Predictive Modeling of Ad Clicks Based on User Cursor Behavior

Mohammad Alirezaeedizicheh

Dissertation submitted to International Business School
for the partial fulfillment of the requirement for the degree of
MASTER OF SCIENCE IN IT FOR BUSINESS DATA ANALYTICS

May 2024

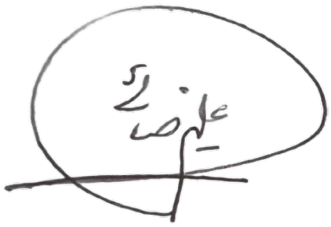
DECLARATION

This dissertation is a product of my own work and is the result of nothing done in collaboration.

I consent to International Business School's free use including/excluding online reproduction, including/excluding electronically, and including/excluding adaptation for teaching and education activities of any whole or part item of this dissertation.

Mohammad Alirezaeedizicheh

Signature



Word length: 13,260 words

ABSTRACT

Purpose:

This coding project is devoted to creating a predictive model that determines the probability of a client to click on a particular advertising after reviewing the data of website interaction of an actual user, particularly this user's initial cursor movements. This way we will use machine learning to supply with some data that can help the ad methods & better the ad positioning.

Design/Methodology/Approach:

The project is following a project management approach of a real-life data science project, and specifically at the beginning, it is mainly analyzing data and investigating. Therefore, we apply in new classes algorithms and methodologies learned in the previous courses, where such approaches are practiced as data preparation, non-neural machine learning, and deep learning techniques. FEATURE ENGINEERING, MODEL TRAINING, AND CLEANING UP THE DATA are all part of our approach. We use classical machine learning and deep learning algorithms to build the model. In addition, we configure data flows to be a key and use diverse aids such as visualization techniques, time series forecasting tools to be able to analyze and predict the changes.

Solution:

The challenge of determining the number of people clicking the ad after visiting the website is explored throughout a detailed examination of user engagement data collected during website visits. We utilize different machine learning models, such as basic algorithms to deep learning architectures, to establish a predictive one that is accurate in spotting the people most likely to click on ads. We offer sequence of improvisations and refinement of the model as the optimal performance is not always achievable.

Limitations/Implications:

We have presented our prediction model which is looked very promising, but there are some limiting factors and consequences that we need to bear in mind. Developing the previously suggested solutions may require other projects that require some adjustment of less serious problems such as feature selection or report manual analysis techniques among others. Not only that but advertising techniques could develop into having one's ads more specifically targeted at the audience and ads positioned where it does not block the content.

Reflections:

Throughout the project, gaining valuable insights was among the most important lessons I obtained from the course and thus I would do wonderfully as a data analyst. Iterative practice shows the importance of think-and-solve skills in data science, illustrating how we should be able to modify or optimize the model to fit the purpose. This encounter will be a catalyst in team collaboration and coding projects in the following work emphasizing the fact that analysis and its thorough investigation are in the center of objective achievements.

Contents

1 INTRODUCTION

1.1 Task and requirements

The coding project, as indicated above, is to figure out the user ad clicks predictive model using cursor mouse movements data. This project fulfills the requirements of the discipline of data science, where the data on users' interactions with the online advertisements is in the main attention of the study.

What our purpose is to create the model that will help us predict if user will click on an ad depending on the cursor movements and other data. This task is being treated like a real-world practical data science project where the ingenuity of developer will be required to discover, interpret and excel in data modeling.

One of the project components of this work is using the tools and methods taught in previous classes including data preparation, machine learning and deep learning. Employing these devices, the project was set to probe into data pipelines with multiple solutions as well as taking into account considerations correlating to data science profession.

the Project is an open-ended one, meaning that one is free to choose between the different variables, encoding methods, and modelling techniques There is a lot of freedom on what one can do with the project: At one end of the spectrum this could involve choosing between different variables or encoding methods and at the other end choosing between different modelling techniques. However, certain necessity, such as gating data to the users who spent at least 5 seconds on the webpage as well as restricting specific variables which are used for the prediction, is set.

It should be observed that the project is imposing a focus on a proper assessment of the available data, a competent data analysis, and a prudent interpretation of the model performance. Review of the steps undertaken and the results obtained is a crucial part of the project which points not just to the outcome of earth system modeling but also to the process of building and implementing models.

In summary, the coding project can be considered a perfect ground for plunging into the intricacies of predictive modelling in the space of online marketing, through the application of theoretical concepts and practical skills addressing real-world problems as well as gaining analytical and prediction experience.

1.2 Data set

The data set that we are going to explore and work on in this project is The Attentive Cursor Dataset gathered (Luis A. Leiva, Ioannis Arapakis, 2020). It contains a groundtruth.tsv file which has four columns:

- `user_id`: (string) Participant's ID.
- `ad_clicked`: (int) Whether the participant clicked on the ad (1) or not (0).
- `attention`: (int) Self-reported attention score, in 1-5 Likert-type scale (1 denotes no attention).
- `log_id columns`: (string) Mouse tracking log ID.

a aparticipants.tsv file which contains :

- `user_id`: (string) Participant's ID.
- `country`: (string) Participant's country, in ISO-3 format.
- `education`: (int) Level of education, according to 6 bins (see table below).
- `age`: (int) Participant's age, according to 9 bins (see table below).
- `income`: (int) Level of income, according to 8 bins (see table below).
- `gender`: (string) Participant's gender.
- `ad_position`: (string) Ad stimulus position.
- `ad_type`: (string) Ad stimulus type.
- `ad_category`: (string) Ad stimulus category.
- `serp_id`: (string) Ad SERP identifier.
- `query`: (string) Ad stimulus query.
- `log_id`: (string) Mouse tracking log ID.

and finally a logs folder which contains csv files related to each user's mouse cursor movement that is recorded by a software named evtrack (Luis A. Leiva, Ioannis Arapakis, 2013) and presented in a csv file that contains eight columns:

- `cursor`: (int) This column is always 0 because all participants used a regular computer mouse.
- `timestamp`: (int) Timestamp of the event, with millisecond precision.
- `xpos`: (float) X position of the mouse cursor.
- `ypos`: (float) Y position of the mouse cursor.
- `event`: (string) Browser's event name; e.g. load, mousemove, click, etc.
- `xpath`: (string) Target element that relates to the event, in XPath notation.
- `attrs`:(string) Optional. Element attributes, if any.
- `extras`: (string) Optional. A JSON string with Euclidean distances to different reference points of the ad's bounding box.

2 FUNCTIONS AND LIBRARIES

In this chapter i will write and put all the functions that i need during the project. I guess it will be easeir to read and find the functions by dedicating a separate chapter. Then whenever the function is called i will reference by the number and name of the function. Also i will create a separate sub-chapter to import all the libraries that are needed.

2.1 Libraries

[101]:
```
import os
import cv2
import math
import random
import zipfile
import pyforest
import requests
import datetime
import warnings
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import matplotlib.patches as patches

from PIL import Image
from skimpy import skim
from nbconvert import PythonExporter
from imblearn.pipeline import Pipeline as ImbPipeline
from imblearn.over_sampling import SMOTE, RandomOverSampler

from sklearn import tree
from sklearn.svm import SVC
from sklearn.tree import plot_tree
from sklearn.pipeline import Pipeline
from sklearn.utils import class_weight
from sklearn.compose import ColumnTransformer
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, roc_auc_score, auc
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import AdaBoostClassifier, BaggingClassifier,␣
↪StackingClassifier
```

```
from sklearn.metrics import precision_score, recall_score, f1_score,␣
 ↪accuracy_score, confusion_matrix,make_scorer

from keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras import backend as K
from tensorflow.keras.regularizers import l2
from tensorflow.keras.models import Sequential
from tensorflow.keras.regularizers import l1_l2
from tensorflow.keras.optimizers import Adam, SGD
from tensorflow.keras.initializers import he_normal
from tensorflow.keras.callbacks import EarlyStopping, LearningRateScheduler
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪Dropout, BatchNormalization
```

[9]:
```
warnings.filterwarnings("ignore")
```

2.2 Functions

2.2.1 Download and extract

[10]:
```
def download_and_extract(url):

    '''
    It has the url as a parameter, gets the Downloads folder path, creates␣
 ↪CodingProject folder,
    Dwonloads the zip file, unzip the content of it, and returns the unzipped␣
 ↪folder
    '''

    # Get the current downloads folder
    downloads_folder = os.path.join(os.path.expanduser("~"), "Downloads")

    # Create the CodingProject folder within the downloads folder
    unzipped_folder = os.path.join(downloads_folder, "CodingProject")
    os.makedirs(unzipped_folder, exist_ok=True)

    # Download the ZIP file
    response = requests.get(url)

    # Save the ZIP file to the downloads folder
    zip_file_path = os.path.join(downloads_folder, "file.zip")
    with open(zip_file_path, "wb") as zip_file:
        zip_file.write(response.content)

    # Extract the contents of the ZIP file to the CodingProject folder
    with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
```

```
        zip_ref.extractall(unzipped_folder)

    # Remove the ZIP file after extraction
    os.remove(zip_file_path)

    # Return the path to the CodingProject folder
    return unzipped_folder
```

2.2.2 ROC and AUC curve

(mo-alrz, 2023)

[11]:
```
def roc_auc(clf, modelname, X, y):
    vals = clf.predict_proba(X)
    n_classes = vals.shape[1]
    fpr = dict()
    tpr = dict()
    roc_auc = dict()

    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y, vals[:, i], pos_label=i)
        roc_auc[i] = auc(fpr[i], tpr[i])

    plt.figure(figsize=(8, 6))
    for i in range(n_classes):
        plt.plot(
            fpr[i],
            tpr[i],
            lw=2,
            label=f"Class {i} (AUC = {roc_auc[i]:.2f})")

    plt.plot([0, 1], [0, 1], color="navy", linestyle="--", label="Random␣
↪classifier")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title(f"{modelname}: ROC curves")
    plt.legend(loc="lower right")
    plt.show()
```

2.2.3 Confusion matrix

(mo-alrz, 2023)

[12]:
```
def conf_matrx(true_labels, predicted_labels, model_type):
    class_labels = ["red", "green", "yellow"]
    cm = confusion_matrix(true_labels, predicted_labels)
    cm_df = pd.DataFrame(cm, index=class_labels[:cm.shape[0]],␣
↪columns=class_labels[:cm.shape[1]])
```

```
    print(f"Confusion Matrix For {model_type}")
    print(cm_df)
```

2.2.4 Randomly plot images

[13]:
```
def plot_random_images(image_dir, num_images=8):

    # Get a list of all image files in the directory
    image_files = [os.path.join(image_dir, file) for file in os.
↪listdir(image_dir)]

    # Randomly select num_images images
    selected_images = random.sample(image_files, min(num_images,␣
↪len(image_files)))

    # Create a 2x4 grid for plotting images
    fig, axes = plt.subplots(2, 4, figsize=(12, 6))

    # Flatten the axes array for easy iteration
    axes = axes.flatten()

    # Plot each image
    for i, image_path in enumerate(selected_images):
        # Load the image using matplotlib
        image = mpimg.imread(image_path)

        # Plot the image on the corresponding axis
        axes[i].imshow(image)
        axes[i].axis('off')  # Hide axis labels

    # Adjust layout to prevent overlapping
    plt.tight_layout()

    # Show the plot
    plt.show()
```

2.2.5 Image Pre-processing

[14]:
```
def preprocess_image(pic_dir, target_size=(150, 150)):
    try:
        # Attempt to read the image
        image = cv2.imread(pic_dir)

        # Check if the image is successfully loaded
        if image is None:
            raise FileNotFoundError("Image file not found or cannot be read.")
```

```
        # Convert the image from BGR to RGB
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        # Resize the image
        image = cv2.resize(image, target_size)

        # Normalize pixel values to be in the range of (0, 1)
        image = image / 255.0

        return image

    except Exception as e:
        # Handle any exceptions that occur during preprocessing
        print(f"Error occurred during image preprocessing: {e}")
        return None
```

2.2.6 Plotting accuracy and loss function

(mo-alrz, 2023)

[15]:
```
def plot_results(results):
  # summarize history for accuracy
  plt.figure(figsize = (12,5))
  plt.subplot(121)
  plt.plot(results.history['accuracy'])
  plt.plot(results.history['val_accuracy'])
  plt.title('model accuracy')
  plt.ylabel('accuracy')
  plt.xlabel('epoch')
  plt.legend(['train', 'val'], loc='lower right')

  # summarize history for loss
  plt.subplot(122)
  plt.plot(results.history['loss'])
  plt.plot(results.history['val_loss'])
  plt.title('model loss')
  plt.ylabel('loss')
  plt.xlabel('epoch')
  plt.legend(['train', 'val'], loc='upper right')

  max_loss = np.max(results.history['loss'])
  min_loss = np.min(results.history['loss'])
  print("Maximum Loss : {:.4f}".format(max_loss))
  print("")
  print("Minimum Loss : {:.4f}".format(min_loss))
  print("")
  print("Loss difference : {:.4f}".format((max_loss - min_loss)))
```

3 DATA PRE-PROCESSING

3.1 Exploring the data

3.1.1 Content

First step in data preparation is to become familiar with data and its characteristics to gain a better insight into the data set and its suitable preparation methods. So first we will download the file and get the basic info using pandas and other libraries.

[ ]:
```
# URL of the ZIP file
data_url = "https://drive.google.com/uc?
 ↪export=download&id=1woOqHNNHLqAZc4QE6CmoedDrFndT1LDA"

# Call the 2 - 2 - 1 download_and_extract function
extracted_folder = download_and_extract(data_url)
```

[154]:
```
# Getting the extracted_folder path
CodingProject_contents = os.listdir(extracted_folder)

# Print the list of contents
for item in CodingProject_contents:
    print(item)
```

```
groundtruth.tsv
logs
participants.tsv
```

3.1.2 groundtruth.tsv

As we can see the delimiter in `groundtruth.tsv` and `participants.tsv` files is tab, so we need to consider it while reading the files.

[ ]:
```
# groundtruth.tsv path
groundtruth_file_path = os.path.join(extracted_folder, "groundtruth.tsv")

# Load the participants.tsv into a DataFrame
groundtruth_data = pd.read_csv(groundtruth_file_path, delimiter='\t')

# Drop the attention column because we are not allowed to use it
groundtruth_data.drop(columns=['user_id','attention'], inplace=True)
```

I use this line of code to convert my dataframes to `LaTeX` format so that we can have a clear output in my PDF file, but to prevent repeating this, i will just write it once here. I put the output of print statement of this code as a markdown cell, so the result in my PDF is a clear and fomratted table. (ExplainHowToSimply, 2023)

[19]:
```
groundtruth_data_head = groundtruth_data.head()
groundtruth_latex = groundtruth_data_head.to_latex()
print(groundtruth_latex)
```

```
\begin{tabular}{lrr}
\toprule
 & ad_clicked & log_id \\
\midrule
0 & 0 & 20181002033126 \\
1 & 1 & 20181001211223 \\
2 & 0 & 20181001170952 \\
3 & 0 & 20181001140754 \\
4 & 0 & 20181001132434 \\
\bottomrule
\end{tabular}
```

	ad_clicked	log_id
0	0	20181002033126
1	1	20181001211223
2	0	20181001170952
3	0	20181001140754
4	0	20181001132434

3.1.3 participants.tsv

[ ]:
```
# participants.tsv path
participants_file_path = os.path.join(extracted_folder, "participants.tsv")

# Load the participants.tsv into a DataFrame
participants_data = pd.read_csv(participants_file_path, delimiter='\t')

# Drop the user_id, education ,age ,income , and gender columns, becasue we are␣
 ↪not allowed to use
participants_data.drop(columns=['user_id', 'education' ,'age' ,'income' ,␣
 ↪'gender'], inplace=True)

# Inspecting the first 5 rows of the DataFrame
participants_data_head = participants_data.head()
participants_latex = participants_data_head.to_latex()
# print(participants_latex)
```

	country	ad_position	ad_type	ad_category	serp_id	query	log_id
0	PHL	top-left	dd	Computers & Electronics	tablets	tablets	20181002033126
1	VEN	top-right	dd	Shop - Luxury Goods	casio-watches	casio watches	20181001211223
2	VEN	top-left	native	Shop - Luxury Goods	chivas-regal	chivas regal	20181001170952
3	VEN	top-right	dd	Shop - Luxury Goods	chivas-regal	chivas regal	20181001140754
4	VEN	top-left	native	Autos & Vehicles	audi-r8-used	audi r8 used	20181001132434

3.1.4 Merging groundtruth and participants

If we take a look at these two data sets, we can see that the `user_id` and `log_id` are common in both data sets, so it can be a good idea to merge (Join) these two data sets on these two columns to have a comprehensive data set that consist of those two data sets. By doing this we can also find out whether all the rows are in common or there are some rows that are not common in those columns. i will trun the indicator of merge function on to see how this can help us. It will make a new column that shows which data set consist of which `on=['user_id', 'log_id']` values. Also log_id seems to be a date plus a numbers, so i will sort the final data set by the log_id column in descending order.

[ ]:
```
# Merging two DataFrames based on SQL inner joins (Join on 'log_id') to see the␣
 ↪mutual records
merged_data = pd.merge(groundtruth_data, participants_data, on=['log_id'],␣
 ↪how='inner', indicator = True)

# Sort by log_id column because in merging process the order will be affected
merged_data.sort_values(by='log_id', inplace=True)

# Splitting the DataFrame into two parts, first contains six columns
first_six_columns = merged_data.iloc[:, :6]
next_six_columns = merged_data.iloc[:, 6:]

# Convert the first six columns to LaTeX format
latex_table_first_six = first_six_columns.head().to_latex()

# Convert the next six columns to LaTeX format
latex_table_after_six = next_six_columns.head().to_latex(index=False)

# Print or save the LaTeX tables
# print(latex_table_first_six)
# print(latex_table_after_six)
```

	user_id	ad_clicked	log_id	country	ad_position	ad_type
1350	2fq3fhu6r693jl2sdb8fhbd190	1	20161214224444	USA	top-right	dd
884	mh0d0jr0oo98kriolus0ml0591	0	20161215173310	USA	top-left	dd
2110	hp3hqt4ifrb3q4f8vgo8e98e65	0	20161217041101	USA	top-left	dd
1498	9k7e1d73n5reh06p705cgp2ke7	0	20161218020519	USA	top-left	dd
105	j25njapljcob43qrg1cfnsg452	0	20161219223751	USA	top-left	native

ad_category	serp_id	query	_merge
Computers & Electronics	galaxy-s7	galaxy s7	both
Computers & Electronics	ipad-pro	ipad pro	both
Games	xbox-one	xbox one	both
Shop - Wholesalers & Liquidatr	iphone-6se	iphone 6se	both
Shop - Luxury Goods	jack-daniels	jack daniels	both

[23]:
```
merged_data['_merge'].value_counts()
```

[23]:
```
_merge
both          2909
left_only        0
right_only       0
Name: count, dtype: int64
```

[ ]:
```
# now i can drop the _merge because i dont it anymore.
merged_data.drop(columns=['_merge'],inplace=True)
```

By observing the result of _merge column and not having a change in length of final data set we can say that the merged data is successful and all the rows had the common `log_id`

3.1.5 Logs folder

Now we will look at one of the csv files inside logs folder, if we look at the csv files names, we can say that each csv is related to a `log_id`, and has the same name as `log_id`. Also the delimiter in this csv files is space that needs to be considered.

[ ]:
```
logs_folder_path = os.path.join(extracted_folder, "logs")

# List all files and directories in the logs folder
logs_folder_contents = os.listdir(logs_folder_path)

# Becasue the merged_data is sorted by logs_id, here we will do the same,␣
 ↪because file names are the same
logs_folder_contents.sort()

# Read the first one to see the content
df = pd.read_csv(f'{logs_folder_path}\\{logs_folder_contents[0]}',sep=' ')

df_latex = df.head().to_latex()
```

```
# print(df_latex)
```

	cursor	timestamp	xpos	ypos	event	xpath	attrs	extras
0	0	1481751971767	0	0	load	/	{}	{}
1	0	1481751977860	0	0	blur	/	{}	{}
2	0	1481751993927	0	0	focus	/	{}	{}
3	0	1481752002419	1342	11	mouseover	//*[@id= 'searchform'] /div/div	{}	{"topRight":136, "topLeft":560, "bottomRight":495, "bottomLeft":735, "middle":436, "inTarget":false}
4	0	1481752002572	1170	171	mousemove	//*[@id= 'rhs_block'] /div/div /div[4]/div[1]	{}	{"topRight":98, "topLeft":386 , "bottomRight":333, "bottomLeft":500 , "middle":201, "inTarget":true}

3.2 Sanity check

For the last step of manipulating data sets, we want to make sure that all the csv files inside logs folder are in the log_id column of our merged_data. This can be done in the following code:

3.3 EDA (Exploratory Data Analysis) and Preparation

Now that our data sets are ready and first stpes of pre processing is done, we want to go deep inside the data to gain some insight about the data and its characteristics. It will help us identfiy which variables are more important and can be a vital feature for the models that we will create.

3.3.1 Filter data based on user duration

According to the task we the only observations that matters for us in this project, are the users who stayed more than 5 seconds in the website, so i will extract the duration that every user spent on the website. I will do it with looping through CVS files and deducting the ending timestamp (Max) and starting timestamp (Min) for each user. I will also add it to the merged_data as a new column because i think it can be a useful feature to use in our our models. I will put the name of csv and duration in a dictionary, and then merge them (on log_id) so that i make sure evey record goes to its relevant observation.

[ ]:
```
duration = {}

for csv in logs_folder_contents:
    # Read the CSV file into a DataFrame
    df = pd.read_csv(os.path.join(logs_folder_path, csv), sep=' ');
```

```
    # Convert timestamps to datetime objects
    df['timestamp'] = pd.to_datetime(df['timestamp'], unit='ms');

    # Calculate the difference between the maximum and minimum times in seconds
    time_diff_seconds = (df['timestamp'].max() - df['timestamp'].min()).
↪total_seconds();

    # Store the time difference in the 'duration' dictionary
    duration[int(csv[:-4])] = round(time_diff_seconds,2);
```

[ ]:
```
# Convert dictionaries to DataFrames
duration_df = pd.DataFrame.from_dict(duration, orient='index',␣
↪columns=['duration'])
duration_df.reset_index(inplace=True)
duration_df.rename(columns={'index': 'log_id'}, inplace=True)

# Merge dataframes
merged_data = pd.merge(merged_data, duration_df, on='log_id', how='left')
```

[ ]:
```
# And finally filter the data with users who spent more than 5 seconds
merged_data = merged_data[merged_data['duration'] >= 5]
# print(merged_data.head().to_latex())
```

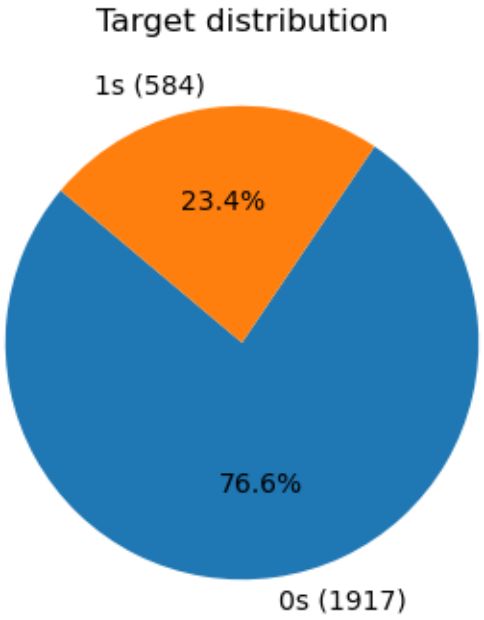
	ad_clicked	log_id	country	ad_position	ad_type	ad_category	serp_id	query	duration
0	1	20161214224444	USA	top-right	dd	Computers & Electronics	galaxy-s7	galaxy s7	127.146000
1	0	20161215173310	USA	top-left	dd	Computers & Electronics	ipad-pro	ipad pro	19.169000
2	0	20161217041101	USA	top-left	dd	Games	xbox-one	xbox one	100.974000
3	0	20161218020519	USA	top-left	dd	Shop - Wholesalers & Liquidatr	iphone-6se	iphone 6se	85.295000
4	0	20161219223751	USA	top-left	native	Shop - Luxury Goods	jack-daniels	jack daniels	21.228000

3.3.2 Class Imbalance

First important thing is our target variable `ad_clicked`, We want to see is if there is a class imbalance because it is one of the most important things when creating a model.

[29]:
```
classes = merged_data['ad_clicked'].value_counts()
plt.figure(figsize=(4, 4))
plt.pie(classes, labels=[f'{label}s ({count})' for label, count in zip(classes.
↪index, classes)], autopct='%1.1f%%', startangle=140)
```

```
plt.title('Target distribution')
plt.show()
```



It is clear there is significant imbalance in our target variable and number of users who did not click on the ad is almost 3 times more than ones who has clicked on the ad

3.3.3 Missing values

[30]:
```
merged_data.isnull().sum()
```

[30]:
```
ad_clicked     0
log_id         0
country        0
ad_position    0
ad_type        0
ad_category    0
serp_id        0
query          0
duration       0
dtype: int64
```

We dont have any missing values so we we do not need to handle or fill anything.

3.3.4 Outliers

In this data set because most of our variables are categorcial, and we are facing a classification scenario and since the classification models are robust to utliers i will use all the data points and

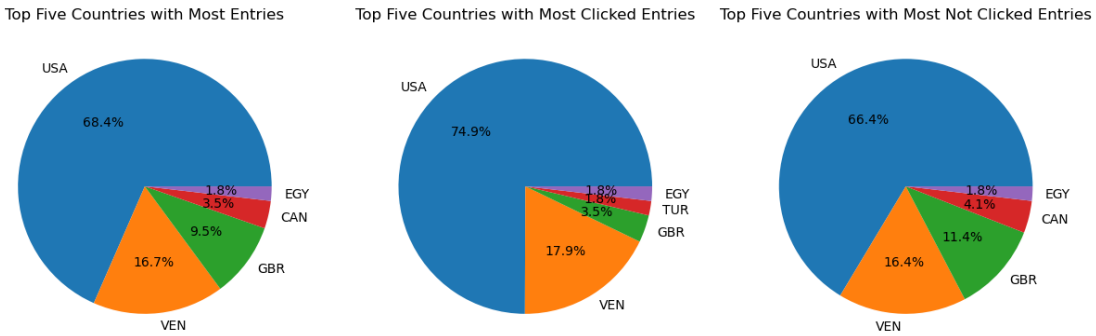
wont deal with utliers. (Verma Y, 2022)

3.3.5 Visualizations

In this part i will make some visualization in order to see if here is any particular trend or relationship between variables.

Top 5 countries with most entries

[31]:
```
# Top 5 countries with most entries
top_5_countries = merged_data['country'].value_counts().head(5)

# Top five countries with most clicked entries
top_clicked_countries = merged_data[merged_data['ad_clicked'] == 1]['country'].
 ↪value_counts().head(5)

# Top five countries with most not clicked entries
top_not_clicked_countries = merged_data[merged_data['ad_clicked'] ==␣
 ↪0]['country'].value_counts().head(5)

# Create figure and subplots
fig, axs = plt.subplots(1, 3, figsize=(15, 5))

# Plot top 5 countries with most entries
axs[0].pie(top_5_countries, labels=top_5_countries.index, autopct='%1.1f%%')
axs[0].set_title('Top Five Countries with Most Entries')

# Plot top five countries with most clicked entries
axs[1].pie(top_clicked_countries, labels=top_clicked_countries.index,␣
 ↪autopct='%1.1f%%')
axs[1].set_title('Top Five Countries with Most Clicked Entries')

# Plot top five countries with most not clicked entries
axs[2].pie(top_not_clicked_countries, labels=top_not_clicked_countries.index,␣
 ↪autopct='%1.1f%%')
axs[2].set_title('Top Five Countries with Most Not Clicked Entries')

# Show plots
plt.show();
```



[32]:
```
merged_data['country'].nunique()
```

[32]: 68

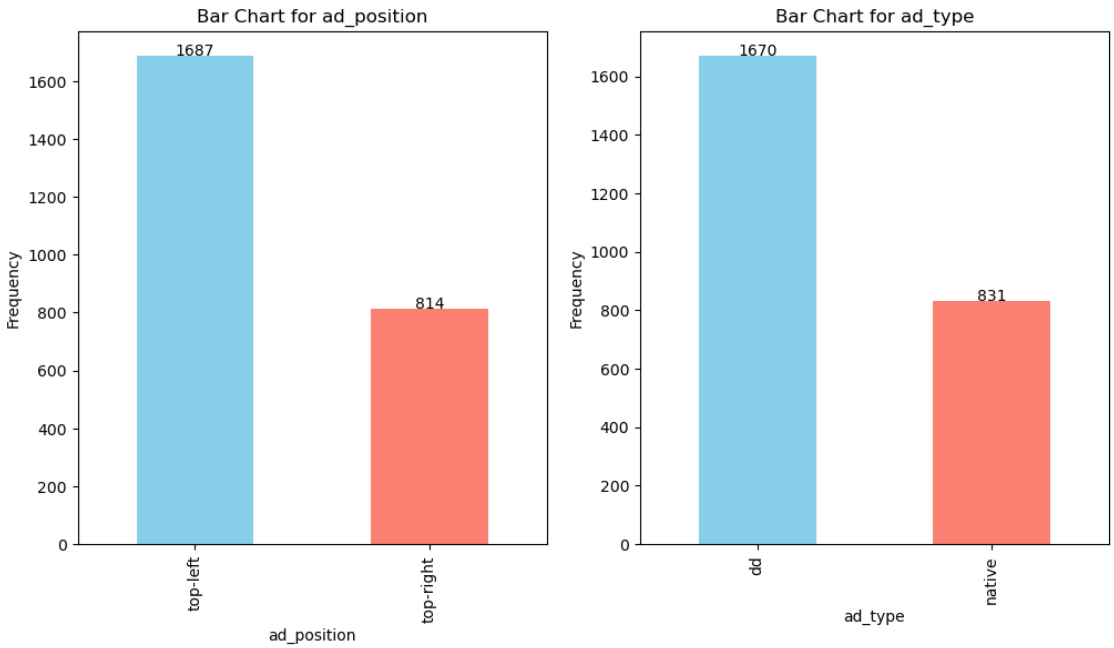
We can see which countries have the most entry, the most clicked entries and most not-clicked entries. But because country column has 68 unique values it wont be a useful feature for us because it wont provide any significant information and in our modeling it may cause overfitting and computational problems. An approach is if this information is vital, use it in broader categoreis like regions, but here i will smiply drop it and wont use it as a feature.

[ ]:
```
merged_data.drop(columns=['country'],inplace=True)
```

Ad position and type

[34]:
```
# Create subplots
fig, axs = plt.subplots(1, 2, figsize=(12, 6))

# Iterate over columns
for idx, column in enumerate(['ad_position', 'ad_type']):
    # Count values
    value_counts = merged_data[column].value_counts()

    # Plot stacked bar chart
    value_counts.plot(kind='bar', ax=axs[idx], color=['skyblue', 'salmon'])

    # Add data labels
    for i, v in enumerate(value_counts):
        axs[idx].text(i, v + 0.1, str(v), ha='center')

    # Set plot title
    axs[idx].set_title(f'Bar Chart for {column}')

    # Set x-axis label
    axs[idx].set_xlabel(column)
```

```
    # Set y-axis label
    axs[idx].set_ylabel('Frequency')

# Show plot
plt.show();
```

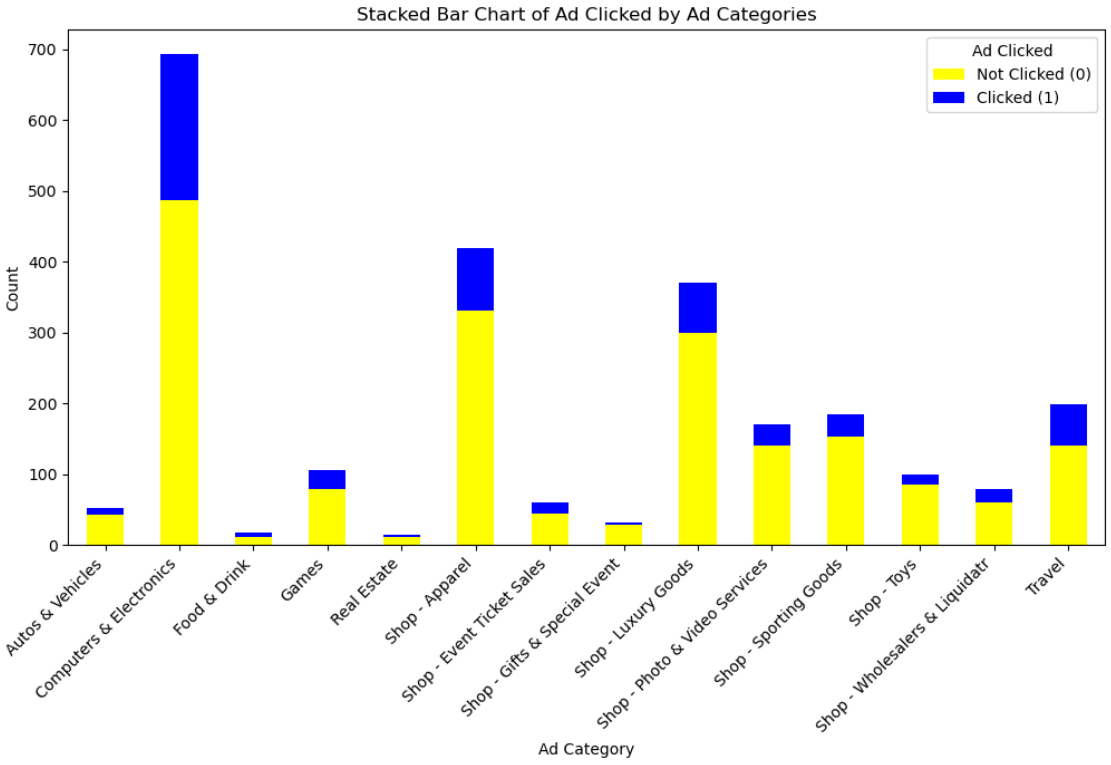


`ad_position` and `ad_type` can be vital because although they are categorical but first of all they only consist of two categories, and about ad_position it shows where the ad is displayed and it may affect how user react to that, for instance if someone is left handed and the ad is being displayed on the left side of the display there will be a higher probability that they click on it, on the other hand close number of `dd : top-left` and `native : top-right` can be a sign that native ads are being displayed in top-right while dd ads are in to-left.

Ad Category

[35]:
```
# Group the data by 'ad_category' and 'ad_clicked' and count the occurrences
category_counts = merged_data.groupby(['ad_category', 'ad_clicked']).size().
 ↪unstack()

# Plot the stacked bar chart
category_counts.plot(kind='bar', stacked=True, figsize=(12, 6), color=['yellow',␣
 ↪'blue'])

# Set plot title and labels
plt.title('Stacked Bar Chart of Ad Clicked by Ad Categories')
plt.xlabel('Ad Category')
plt.ylabel('Count')
```

```
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better␣
↪readability

# Show the plot
plt.legend(title='Ad Clicked', labels=['Not Clicked (0)', 'Clicked (1)'])
plt.show()
```



This variable can be useful because we can see how the number of clicked and not clicked users varies through each category, although an accurate breakthrough and exploring the treatment of our model for each category separately can be very helpful and insightful in our analysis, but again because there are 14 variables and if i want to use them as one of my features, like country column, even one hot coding will be problematic due to generating a huge number of new columns in this case i will transform this to 4 different categories. All the shops in one category : `Shop`, Computer & electronics alongside games : `Digital`, Travl with food and drinks : `Hobby`, Autos & Vehicles with Real Estate : `Investment`.

[ ]:
```
# Define a dictionary with obsolete categories
new_category = {
    'Shop': ['Shop - Apparel', 'Shop - Event Ticket Sales', 'Shop - Gifts &␣
↪Special Event',
             'Shop - Luxury Goods' ,'Shop - Photo & Video Services','Shop -␣
↪Sporting Goods',
```

```
            'Shop - Toys','Shop - Wholesalers & Liquidatr'],
    'Digital': ['Computers & Electronics', 'Games'],
    'Hobby': ['Travel', 'Food & Drink'],
    'Investment': ['Autos & Vehicles', 'Real Estate']
}

# Create a function to change the old categories with new ones
def map_category(category):
    for broad_category, sub_categories in new_category.items():
        if any(sub_category in category for sub_category in sub_categories):
            return broad_category
    return

# Apply map_category function to all values of category column using map␣
 ↪built-in function
merged_data['ad_category'] = merged_data['ad_category'].map(map_category)
```

Serp id and Query These two column show the SERP (Search Engine Results Pages) identifier and stimulus query and i think it is important in terms of SEO and search engine optimization. (Dean Brian, 2023) And for the same reason as country column i will drop this two columns.

[37]:
```
merged_data.drop(columns=['serp_id','query'],inplace=True)
```

4 CREATING FEATURES FROM CSV FILES

4.1 First approach to the task

I will treat this task in two different ways, first i will extract and create some features that i feel necessary and meaningful for my analysis and model building from the csv files in `logs folder`, and with the combination of original features available in the data set (`merged_data`) and my custom extracted features, i will a build a final tabular data set and develop some classic machine learning models using non-neural algorithms to predict the binary target (`ad_clicked`).

There might be thousands of other meaningful and useful features that can be extrated from this data sets that contain wealth of information, but due to time shortage and limtied computational resources i just picked the few ones that i feel can be vital for the project

[38]:
```
print(merged_data['duration'].std())
print(merged_data['duration'].mean())
```

```
108843.01102440049
2234.5462774890043
```

4.1.1 Duraion

Duration shows how long a user spent on the website and already extracted for filtering data fo user who spent more than 5 seconds.

4.1.2 Time of the day / Day of the week

The way I see this data set, other useful features that can be extracted from the CSV (log) files are time of the day and the day of the week in which the user was sepnding time on this website. Because for instance if it is Monday, 2pm probably people are much busier and may be less willing to click on an add to see the content. A very important thing is that because we filtered the data based on duration some of log_id has been deleted from merged_data so if we just calculate the duration for all CSV files in logs folder, we need to consider that, so i will use merge function again to extract values only for the records that are still in log_id column.

[ ]:
```
time_of_day = {}
day_of_week = {}

# Iterate through each CSV in the logs folder
for csv in logs_folder_contents:
    # Read the CSV file into a DataFrame
    df = pd.read_csv(os.path.join(logs_folder_path, csv), sep=' ');

    # Convert timestamps to datetime objects
    df['timestamp'] = pd.to_datetime(df['timestamp'], unit='ms');

    first_timestamp = df['timestamp'].iloc[0]

    time_of_day[int(csv[:-4])] = first_timestamp.hour;
    day_of_week[int(csv[:-4])] = first_timestamp.dayofweek; # Monday=0, Sunday=6
```

[ ]:
```
time_of_day_df = pd.DataFrame.from_dict(time_of_day, orient='index',␣
↪columns=['time_of_day'])
time_of_day_df.reset_index(inplace=True)
time_of_day_df.rename(columns={'index': 'log_id'}, inplace=True)

day_of_week_df = pd.DataFrame.from_dict(day_of_week, orient='index',␣
↪columns=['day_of_week'])
day_of_week_df.reset_index(inplace=True)
day_of_week_df.rename(columns={'index': 'log_id'}, inplace=True)

# Merge with merged_data on 'log_id' column
merged_data = pd.merge(merged_data, time_of_day_df, on='log_id', how='left')
merged_data = pd.merge(merged_data, day_of_week_df, on='log_id', how='left')
```

4.1.3 X and Y average

Next useful feature can be the average of cursor movement in X and Y directions, it will give an insight of how users moved their cursor along the session.

[ ]:
```
positions_x = {}
positions_y = {}
```

```
for csv in logs_folder_contents:
    # Read the CSV file into a DataFrame
    df = pd.read_csv(os.path.join(logs_folder_path, csv), sep=' ')
    positions_x[int(csv[:-4])] = round(df['xpos'].mean(),2)
    positions_y[int(csv[:-4])] = round(df['ypos'].mean(),2);
```

[ ]:
```
x_mean = pd.DataFrame.from_dict(positions_x, orient='index', columns=['x_mean'])
x_mean.reset_index(inplace=True)
x_mean.rename(columns={'index': 'log_id'}, inplace=True)

y_mean = pd.DataFrame.from_dict(positions_y, orient='index', columns=['y_mean'])
y_mean.reset_index(inplace=True)
y_mean.rename(columns={'index': 'log_id'}, inplace=True)

merged_data = pd.merge(merged_data, x_mean, on='log_id', how='left')
merged_data = pd.merge(merged_data, y_mean, on='log_id', how='left')
```

4.1.4 Speed of cursor movement

I filtered out the data sets in a way that each record with `xpos = 0` and `ypos = 0` be eliminated from it and only records that had a movement remains, then defined a function to find the Euclidean distance, then calculated the time difference and finally the speed which is gained from the Euclidean distance divided by time differenc.(Wikipedia Contributors, 2019), At the end calculated the mean speed of each user and added to my data set.(ChatGPT)

[ ]:
```
def distance(x1, y1, x2, y2):
    return np.sqrt((x2 - x1)**2 + (y2 - y1)**2)

speed = {}

for csv in logs_folder_contents:
    # Read the CSV file into a DataFrame
    df = pd.read_csv(os.path.join(logs_folder_path, csv), sep=' ')

    # Filter the data
    df_filtered = df[(df['xpos'] != 0) & (df['ypos'] != 0)]

    # Euclidean distance
    df_filtered['distance'] = round(distance(df_filtered['xpos'].shift(),␣
↪df_filtered['ypos'].shift(), df_filtered['xpos'], df_filtered['ypos']),2)

    # Time difference
    df_filtered['time_diff'] = round((df_filtered['timestamp'] -␣
↪df_filtered['timestamp'].shift()) / 1000,2)

    # Speed
```

```
    df_filtered['speed'] = round( df_filtered['distance'] /␣
↪df_filtered['time_diff'] , 2 )

    # Adding verage speed for each dataset to dictionary
    speed[int(csv[:-4])] = round(df_filtered['speed'].mean(),2);
```

[ ]:
```
speed = pd.DataFrame.from_dict(positions_x, orient='index', columns=['speed'])
speed.reset_index(inplace=True)
speed.rename(columns={'index': 'log_id'}, inplace=True)

merged_data = pd.merge(merged_data, speed, on='log_id', how='left')
# print(merged_data.head().to_latex())
```

	ad_clicked	log_id	ad_position	ad_type	ad_category	duration	time_of_day
0	1	20161214224444	top-right	dd	Digital	127.146000	21
1	0	20161215173310	top-left	dd	Digital	19.169000	16
2	0	20161217041101	top-left	dd	Digital	100.974000	3
3	0	20161218020519	top-left	dd	Shop	85.295000	1
4	0	20161219223751	top-left	native	Shop	21.228000	22

day_of_week	x_mean	y_mean	speed
2	454.190000	109.960000	454.190000
3	116.830000	221.500000	116.830000
5	620.150000	112.390000	620.150000
6	311.080000	359.270000	311.080000
0	183.630000	48.840000	183.630000

So our final data set looks like this , with this columns and features. I will approach this task in two different ways. Once i go with this data set and treat it like classical machine leraning cases were we have tabular data in our classification scenario, and the second time i will extract a picture for each data set which shows the trajectory mouse cursor movement, type of ad (`dd` or `native`) and position of the add (`top-right` or `top-left`). In thenext part i will extract the pictures.(Ioannis Arapakis, Luis A. Leiva, 2020) Then treat this project like a image processing classification with neural netwrok based models.

4.2 Second approach to the task

Second approach is, creating features using images and develop machine learning models using neural network algorithms. For this, We would like to plot the images in a way that it reflects `start point, end point, path, type of ad and ad position`. So i will first add these two columns to each data set, then for `ad_position` i will make a separate box in `top-right` and `top-left` of the images, and for the `ad_type` i will put a different `background colors` for each type.

I also created this dictionary and assigned one particular marker for each event, so event column of each data set will be also reflected to each image.(Consulted with my professor, `Zsofia Gyarmathy`)(Implemented with `ChatGPT` assistance) Considering all these i guess the images will have fairly suffiecient amount of information for building my models.

Since there is always room for develpoment and these are custom images that i created for my project, i guess there might be some other things that can be added to images to have better results, maybe in the future, but for now because i would not be delinquent from my dead line i will go with these

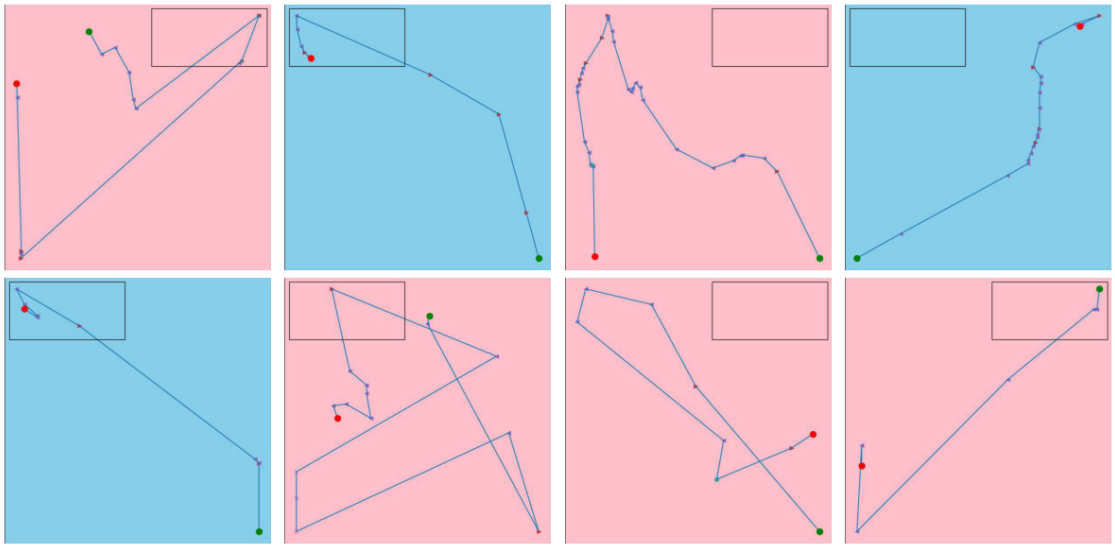
[45]:
```
event_markers = {
    'load': 's',  # square
    'blur': 'D',  # diamond
    'focus': 'P',  # filled plus
    'mouseover': '>',  # triangle right
    'mousemove': '<',  # triangle left
    'mousedown': 'X',  # cross
    'scroll': 'x',  # filled cross
    'mouseup': 'o',  # circle
    'contextmenu': '^',  # triangle up
    'click': 'v',  # triangle down
    'beforeunload': '*',  # star
    'unload': 'd'  # thin diamond
}
```

[ ]:
```
# Iterate through each index in merged_data
for i in merged_data['log_id'].values:

    # Create a new figure and axis object
    fig, ax = plt.subplots(figsize=(6, 6))  # Adjust the figure size as needed

    df = pd.read_csv(f'{logs_folder_path}/{i}.csv', sep=' ')

    ad_type = merged_data.loc[merged_data['log_id'] == i, 'ad_type'].values[0]
    ad_position = merged_data.loc[merged_data['log_id'] == i, 'ad_position'].
↪values[0]

    # Add ad_type and ad_position columns to the DataFrame
    df['ad_type'] = ad_type
    df['ad_position'] = ad_position

    moving_data = df[(df['xpos'] != 0) | (df['ypos'] != 0)]

    # Plot the trajectory of user movement (line plot)
    ax.plot(moving_data['xpos'], moving_data['ypos'], label='User Movement␣
↪Trajectory')

    # Plot markers for the beginning and end points
    beginning_point = moving_data.iloc[0]
    end_point = moving_data.iloc[-1]
    ax.plot(beginning_point['xpos'], beginning_point['ypos'], 'go',␣
↪markersize=10, label='Beginning Point')
```

```
    ax.plot(end_point['xpos'], end_point['ypos'], 'ro', markersize=10,␣
↪label='End Point')

    # Reflecting the marker on the plot
    for event, marker in event_markers.items():
        event_data = df[(df['event'] == event) & ((df['xpos'] != 0) |␣
↪(df['ypos'] != 0))]
        ax.scatter(event_data['xpos'], event_data['ypos'], label=event,␣
↪marker=marker)

    # Set background color based on ad_type for the inner plot area
    if ad_type == 'dd':
        ax.set_facecolor('pink')
    elif ad_type == 'native':
        ax.set_facecolor('skyblue')  # Set background color of the inner plot␣
↪area

    # Remove the white margins outside of the graph
    plt.subplots_adjust(left=0, right=1, top=1, bottom=0)

    # Remove numbers and lines on x and y axes
    ax.set_xticklabels([])
    ax.set_yticklabels([])
    ax.tick_params(axis='both', which='both', length=0)

    # Save the plot as a JPG image
    plt.savefig(f'{extracted_folder}\\trajectories\\{i}.jpg',␣
↪bbox_inches='tight', pad_inches=0)

    # Close the current figure to prevent overplotting
    plt.close(fig)

    # Open the saved image
    im = Image.open(os.path.join(extracted_folder, 'trajectories', f'{i}.jpg'))

    # Create figure and axes with adjusted size
    fig, ax = plt.subplots(figsize=(im.width / 100, im.height / 100))

    # Create rectangles
    if ad_position == 'top-right':
        rect_coordinates = (330, 10)
    else:
        rect_coordinates = (10, 10)
    rect1 = patches.Rectangle(rect_coordinates, 260, 130, linewidth=1,␣
↪edgecolor='black', facecolor='none')
```

```
    # Display the image with the rectangle
    ax.imshow(im)
    ax.add_patch(rect1)

    # Remove white margins
    plt.subplots_adjust(left=0, right=1, top=1, bottom=0)

    # Remove x and y axes
    ax.axis('off')

    # Save the plot with rectangles
    plt.savefig(os.path.join(extracted_folder, 'trajectories', f'{i}.jpg'),␣
↪bbox_inches='tight', pad_inches=0)

    # Close the current figure to prevent overplotting
    plt.close(fig);
```

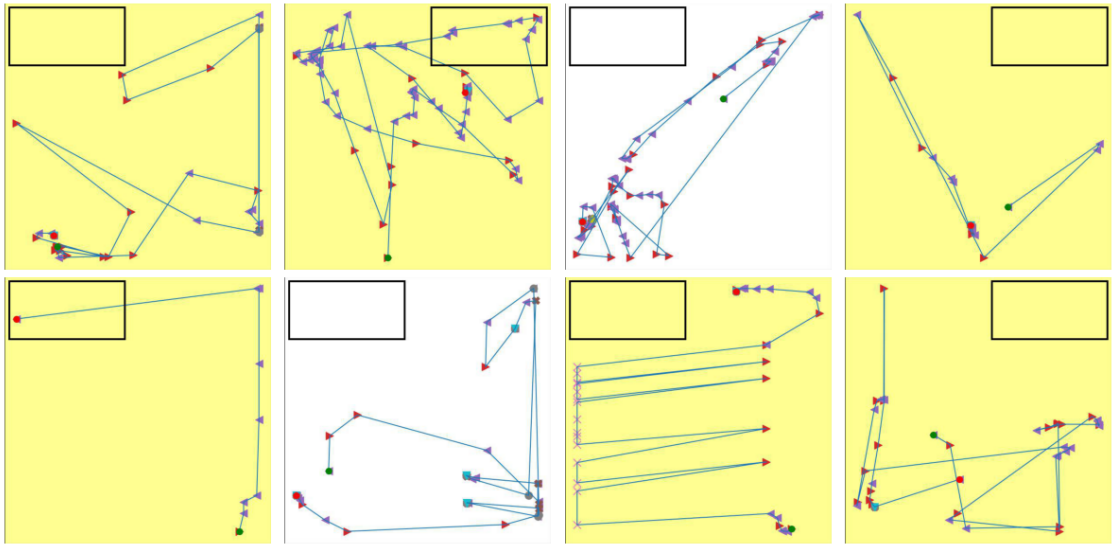
[38]:
```
plot_random_images(f'{extracted_folder}\\trajectories')
```



Since these images are created manually, i do not have a clue that how useful the features in the images were, so i decided to create images with other differences, in the second version, i created the images with the same structure, but increased the thickness of rectangles so that it varies a bit with the lines between the movements, i also changed the background from blue and pink to white and yellow for two reasons, first that i have created markers for events and some of the markers have default colors that might be similar to the background, second thing is that i think pink and blue don't have a high contrast with the pictures illustrations and that might be misinterpereted by the model. I also increased the size of the markers, starting and enging points so that it can be much noticeable. To prevent repeating i just write the relavant parts of code that i changed:

[ ]:
```
for event, marker in event_markers.items():
    event_data = df[(df['event'] == event) & ((df['xpos'] != 0) | (df['ypos'] !=␣
↪0))]

    # Added the s=150 because the default value is too small
    ax.scatter(event_data['xpos'], event_data['ypos'], label=event,␣
↪marker=marker, s=150)

    # Set background color based on ad_type for the inner plot area, th other␣
↪one will be left white
    if ad_type == 'dd':
        ax.set_facecolor('#FFFE91')
```

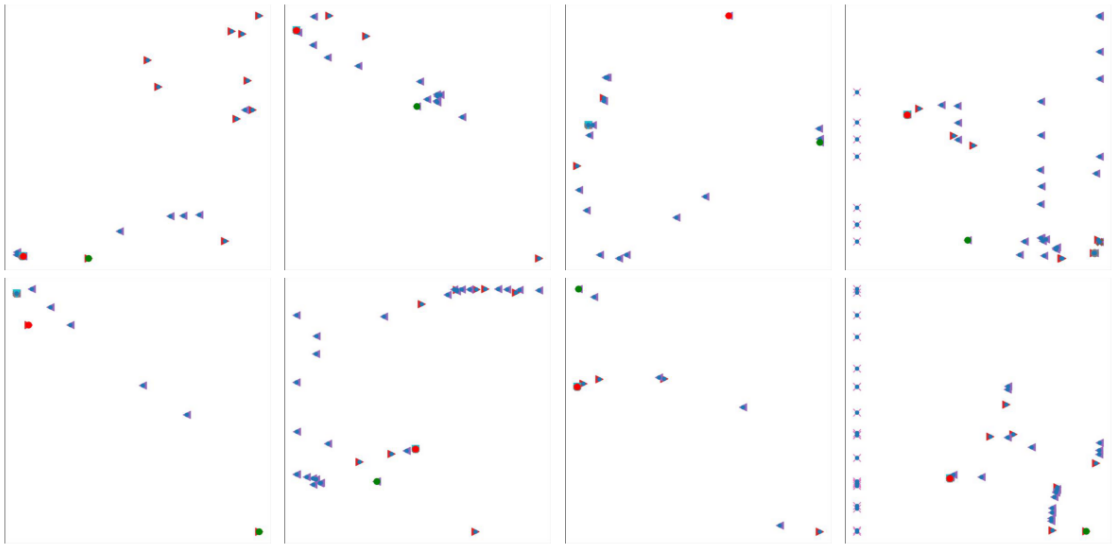
[40]:
```
plot_random_images(f'{extracted_folder}\\trajectories1')
```



For third version of pictures, i completely ignored the information outside the dataset (`ad_type,ad_position`) and also deleted the lines between the cursor movoemenets, i just kept the mouse events, start and ending point in addition to increasing the size of markers. So i just deleted the parts relevant to ad_position and ad_type and for deleting lines between points :

[ ]:
```
# Set the linstyle to None
ax.plot(moving_data['xpos'], moving_data['ypos'], marker='o', linestyle='None',␣
↪label='User Movement Trajectory')
```

[42]:
```
plot_random_images(f'{extracted_folder}\\trajectories2')
```



5 MACHINE LEARNING ALGORITHMS

First thing i do in this step is to drop the `log_id` from my data set because all the sanity checks and feature extractions are done, data pre processing is completed and since it is not a useful variable for us anymore and doesn't contribute to the predictive power of my models and might even introduce noise.

[ ]:
```
imgs_and_labels = merged_data[['ad_clicked','log_id','ad_position','ad_type']]
```

[ ]:
```
merged_data.drop('log_id', axis=1, inplace=True)
# print(merged_data.head().to_latex())
```

	ad_clicked	ad_position	ad_type	ad_category	duration	time_of_day	day_of_week
0	1	top-right	dd	Digital	127.150000	21	2
1	0	top-left	dd	Digital	19.170000	16	3
2	0	top-left	dd	Digital	100.970000	3	5
3	0	top-left	dd	Shop	85.300000	1	6
4	0	top-left	native	Shop	21.230000	22	0

x_mean	y_mean	speed
454.190000	109.960000	454.190000
116.830000	221.500000	116.830000
620.150000	112.390000	620.150000
311.080000	359.270000	311.080000
183.630000	48.840000	183.630000

Next important thing is, as we saw in our EDA step, there is a huge class imbalance in our target variable (See Here) and it is curcial to deal with this imbalance, because when each class has roughly

the same amount of samples, most machine learning algorithms perform at their best. This is so because the majority of algorithms are developed to minimize mistakes and increase accuracy.

In circumstances when the classes in the dataframe are unbalanced, on the other hand, you can predict the majority class with a very high degree of accuracy, but you are unable to capture the minority class, which is typically the main reason for building the model in the first place. For instance, no simple classification model, such as logistic regression or decision tree, would be able to detect the minor class data points if the class distribution indicates that 99% of the data belongs to the majority class.(guest_blog, 2020)

To deal with class imbalance with have couple of options, first thing is that instead of only working with accuracy metrice, make use of: - Confusion Matrix - Precision - Recall - F1 Score - Area Under ROC Curve

Next thing that can spontaneously be a great help in this cases is the righ choice of algorithm, we cane use Penalize Algorithms or Cost-Sensitive Training ones that increase the cost of classification mistakes in the minority class. So i will use Decision trees and SVMs, they can both be effective choices for handling class imbalance, and evaluating them using precision, recall, F1-score, area under ROC curve (AUC-ROC), and confusion matrices will be a good approach.

5.1 Scaling and One hot encoding

[48]:
```
merged_data.describe()
```

[48]:

	ad_clicked	duration	time_of_day	day_of_week	x_mean	\
count	2501.000000	2.501000e+03	2501.000000	2501.000000	2501.000000	
mean	0.233507	2.234546e+03	12.799280	2.145142	333.727477	
std	0.423146	1.088430e+05	6.483926	2.062262	158.093827	
min	0.000000	5.010000e+00	0.000000	0.000000	15.630000	
25%	0.000000	1.402000e+01	9.000000	0.000000	231.200000	
50%	0.000000	2.854000e+01	14.000000	1.000000	307.190000	
75%	0.000000	5.745000e+01	18.000000	4.000000	411.390000	
max	1.000000	5.443292e+06	23.000000	6.000000	1165.830000	

	y_mean	speed
count	2501.000000	2501.000000
mean	238.803926	333.727477
std	151.736566	158.093827
min	2.290000	15.630000
25%	134.240000	231.200000
50%	205.670000	307.190000
75%	303.580000	411.390000
max	1106.250000	1165.830000

Based on the summary statistics of our dataset, our data contains both categorical and numerical variables and our numerical variables look to be quite differently scaled and ranged. This is often a case where the use of scaling techniques can be beneficial in making sure that all features contribute equally in the process of model training. Since i am using decision trees that is not sensitive to feature scaling and SVM that works perfectly with Standard Scaling, i will go with standardization. Standardization is a scaling technique wherein it makes the data scale-free by converting the

statistical distribution of the data into the below format:

- mean : 0
- standard deviation : 1

The standardization formula :

$$Z = \frac{X - \mu}{\sigma} \tag{1}$$

On the other hand i will one-hot encode the categorical variables (`ad_position, ad_type, ad_category`) so that these features can be fed into algorithms to improve prediction accuracy.

[ ]:
```
X = merged_data.drop('ad_clicked',axis = 1)
y = merged_data[['ad_clicked']]
```

[ ]:
```
# Select numerical and categorical columns
X_numeric = X.select_dtypes(include=['float64', 'int64']).columns
X_categoric = X.select_dtypes(include=['object']).columns

# Define preprocessing steps for numerical and categorical columns
numerical_transformer = StandardScaler()
categorical_transformer = OneHotEncoder()

# Define column transformer for preprocessing numerical and categorical columns␣
 ↪separately ,It was also possible to one-hot
# encode and scale manually and then merge them together but this approach looks␣
 ↪way more shorter and easier
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, X_numeric),  # Scale numerical features
        ('cat', categorical_transformer, X_categoric) # One-hot encode␣
 ↪categorical features
    ])

# Define the model pipeline including preprocessing and any model(s) we want to␣
 ↪apply
model_pipeline = Pipeline(steps=[('preprocessor', preprocessor)])

# Fit and transform the preprocessor on the entire dataset
X_processed = model_pipeline.fit_transform(X)

# Fit the OneHotEncoder and transform the categorical features to get feature␣
 ↪names
categorical_transformer.fit(X[X_categoric])
categorical_feature_names = categorical_transformer.
 ↪get_feature_names_out(X_categoric)
```

```
# Convert the processed data back to a DataFrame
X_scaled = pd.DataFrame(X_processed, columns=list(X_numeric) +␣
↪list(categorical_feature_names))
```

5.2 Train-Test split

Train-test splitting is a fundamental technique in machine learning. It involves dividing our dataset into two subsets : a training set (`usually 70-80 % of the data`) and a test set (`usually 30-20 % of the data`). We fit our data on the training data (`train the data`), and we test our model (`make predictions on target variable`) on the test part, and because the test part is not seen by the model the results produced by that will be a good assessment of how well our model performs. A very crucial hyper parameter here is the random state which is used to increase the reproduceability and also it helps us to get same output whenever we run this notebook. It is neseccary to go with the same random state untill the end of our tasks.

[ ]:
```
X_train, X_temp, y_train, y_temp = train_test_split(X_scaled, y, test_size=0.3,␣
↪random_state=42)
X_validation, X_test, y_validation, y_test = train_test_split(X_temp, y_temp,␣
↪test_size=0.5, random_state=42)
```

5.3 SVM

5.3.1 Linear

In SVM linear we will set the `class_weight='balanced'` to deal with the class imbalance.

[ ]:
```
svm_linear = SVC(class_weight='balanced',kernel="linear", probability=True)
svm_linear.fit(X_train,y_train)
svm_linear_preds = svm_linear.predict(X_validation)
```

[ ]:
```
scores_dict = {}
svm_linear_precision = precision_score(y_validation, svm_linear_preds,␣
↪average='weighted')
svm_linear_recall = recall_score(y_validation, svm_linear_preds,␣
↪average='weighted')
svm_linear_f1 = f1_score(y_validation, svm_linear_preds, average='weighted')
svm_linear_accuracy = accuracy_score(y_validation, svm_linear_preds)
```

[ ]:
```
scores_dict["SVM Linear"] = [
    "{:.4f}".format(svm_linear_precision),
    "{:.4f}".format(svm_linear_recall),
    "{:.4f}".format(svm_linear_f1),
    "{:.4f}".format(svm_linear_accuracy)
]
```

5.3.2 Ploynomial

Grid search To find the best hyper parametes for my polynomial SVM, i used the grid search technique, therefore in addition to degrees i will check some other hyperparametes that are effective

in terms of class imbalance. In polynomial version of SVM for dealing with class imbalance i will use SMOTE (Synthetic Minority Oversampling Technique) technique which is most popular technique to cope with the class imbalance.

[56]:
```
# Define the parameter grid for the SVM classifier
param_grid_svm = {
    'svm__degree': [3, 4, 5],
    'svm__class_weight': [None, 'balanced'],
    'svm__C': [0.1, 1, 10],
}

# Define a custom scoring function (weighted F1-score)
scorer = make_scorer(f1_score, average='weighted')

# Define the steps for the pipeline
steps = [
    ('sampling', SMOTE(random_state=42)),  # Apply SMOTE for oversampling
    ('svm', SVC(kernel='poly', probability=True, random_state=42)),
]

# Initialize the pipeline
pipeline = ImbPipeline(steps)

# Initialize GridSearchCV
grid_search = GridSearchCV(pipeline, param_grid_svm, cv=5, scoring=scorer,␣
 ↪n_jobs=-1)

# Fit the grid search to the training data
grid_search.fit(X_train, y_train)

# Get the best estimator
best_model = grid_search.best_estimator_

# Displaying the best model result
best_model
```

[56]:
```
Pipeline(steps=[('sampling', SMOTE(random_state=42)),
                ('svm',
                 SVC(C=10, degree=4, kernel='poly', probability=True,
                     random_state=42))])
```

Implementing Now we make the model with the best_model hyper parameters

[ ]:
```
svm_classifier = SVC(C=10, degree=4, kernel='poly', probability=True,␣
 ↪class_weight='balanced', random_state=42)
```

```
# Fit the SVM classifier on the training data
svm_classifier.fit(X_train, y_train)

# Predict the labels using the SVM classifier
svm_preds = svm_classifier.predict(X_validation)

# Calculate precision, recall, F1-score, and accuracy for the current degree
precision = precision_score(y_validation, svm_preds, average='weighted')
recall = recall_score(y_validation, svm_preds, average='weighted')
f1 = f1_score(y_validation, svm_preds, average='weighted')
accuracy = accuracy_score(y_validation, svm_preds)

# Store the scores in the dictionary
scores_dict['SVM Polynomial'] = [
    "{:.4f}".format(precision),
    "{:.4f}".format(recall),
    "{:.4f}".format(f1),
    "{:.4f}".format(accuracy)
]
```

5.3.3 Decision Tree

Grid search

[58]:
```
# Define the parameter grid
param_grid = {
    'max_depth': [3, 4, 5],
    'min_samples_leaf': [1, 2, 3],
}

# Initialize the decision tree classifier
dec_tree = DecisionTreeClassifier(random_state=42, class_weight="balanced")

# Initialize GridSearchCV
grid_search = GridSearchCV(dec_tree, param_grid, cv=5, scoring='f1_weighted',␣
 ↪n_jobs=-1)

# Fit the grid search to the training data
grid_search.fit(X_train, y_train)

# Get the best estimator
best_model = grid_search.best_estimator_

# Displaying the best model result
best_model
```

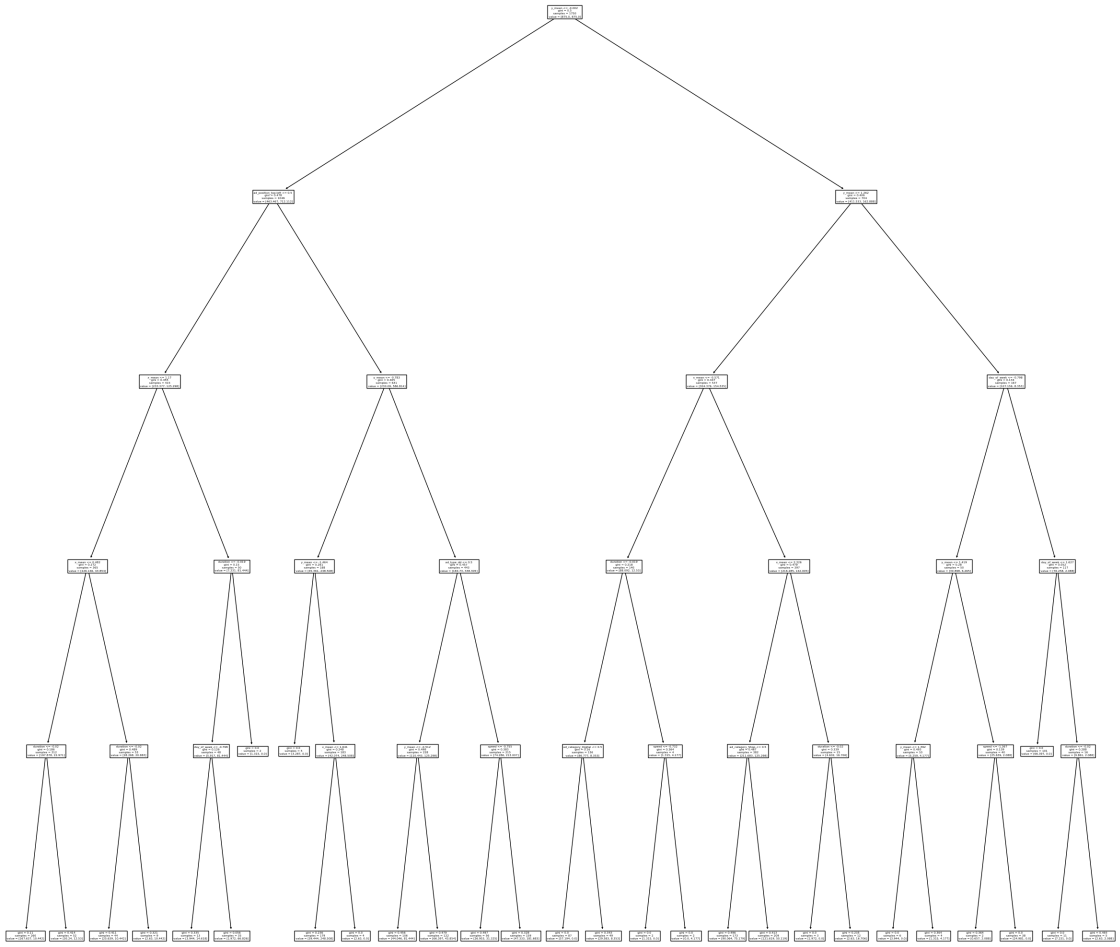
[58]:
```
DecisionTreeClassifier(class_weight='balanced', max_depth=5, min_samples_leaf=2,
                       random_state=42)
```

Implementing

[ ]:
```
dec_tree = DecisionTreeClassifier(class_weight='balanced', max_depth=5,␣
↪min_samples_leaf=2,random_state=42)
dec_tree.fit(X_train,y_train)
dec_tree_preds = dec_tree.predict(X_validation)
```

[ ]:
```
dec_tree_precision = precision_score(y_validation, dec_tree_preds,␣
↪average='weighted')
dec_tree_recall = recall_score(y_validation, dec_tree_preds, average='weighted')
dec_tree_f1 = f1_score(y_validation, dec_tree_preds, average='weighted')
dec_tree_accuracy = f1_score(y_validation, dec_tree_preds, average='weighted')

scores_dict["Decision Tree"] = ["{:.4f}".format(dec_tree_precision),"{:.4f}".
↪format(dec_tree_recall),
                                "{:.4f}".format(dec_tree_f1),"{:.4f}".
↪format(dec_tree_accuracy)]
```

[61]:
```
plt.figure(figsize=(30,30))
tree.plot_tree(dec_tree, feature_names=dec_tree.feature_names_in_)
plt.show()
```



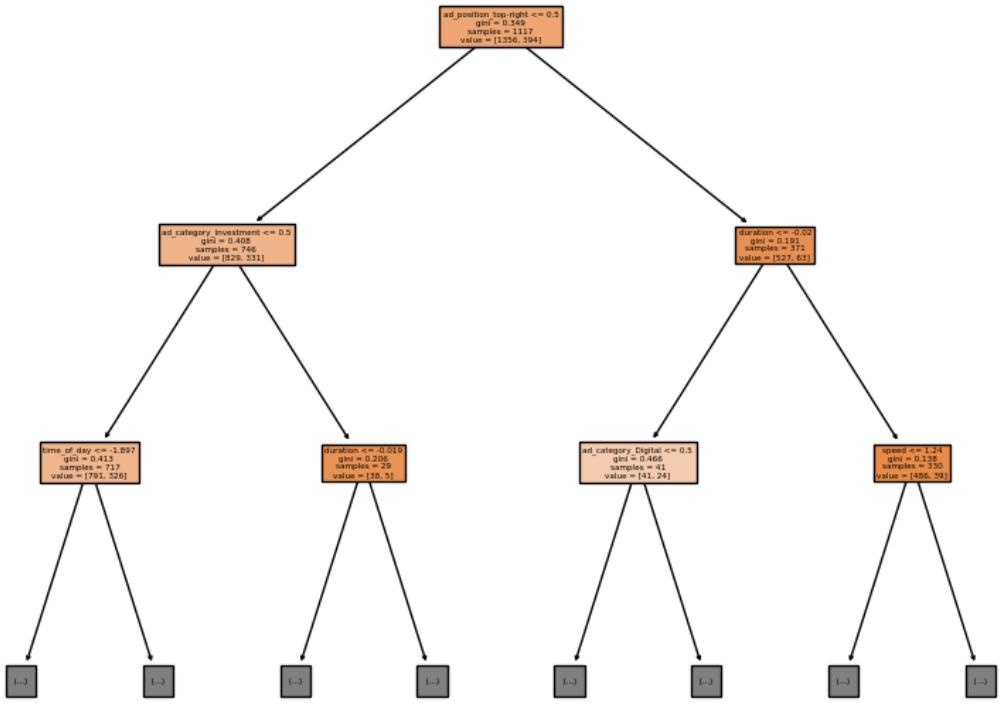
5.4 Random Forest

[ ]:
```
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
rf_preds = rf.predict(X_validation)
```

[ ]:
```
rf_precision = precision_score(y_validation, rf_preds, average='weighted')
rf_recall = recall_score(y_validation, rf_preds, average='weighted')
rf_f1 = f1_score(y_validation, rf_preds, average='weighted')
rf_accuracy = f1_score(y_validation, rf_preds, average='weighted')

scores_dict["Random Forest"] = ["{:.4f}".format(rf_precision),"{:.4f}".
 ↪format(rf_recall),
```

```
                                        "{:.4f}".format(rf_f1),"{:.4f}".
↪format(rf_accuracy)]
```

[73]:
```
# Visualizing first tree in the forest
tree = rf.estimators_[0]

plt.figure(figsize=(10, 8))
plot_tree(tree, feature_names=X_train.columns, filled=True, max_depth=2)
plt.show()
```



5.5 Model Evaluation

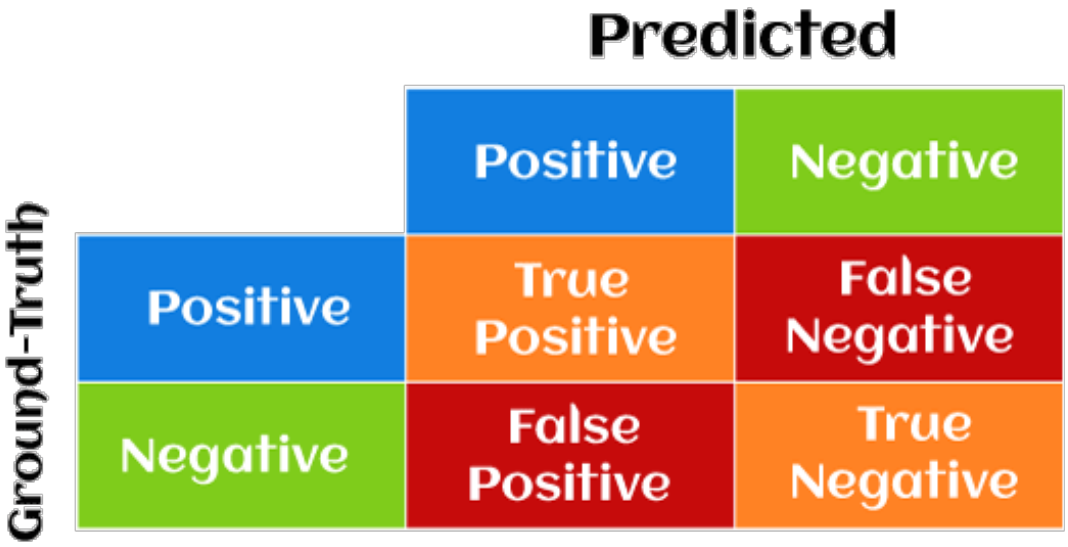
5.5.1 Accuracy Metrices

First evaluation tool is our accuracy metrices that i stored in scores_dict. I convert it to a dataframe for a better readability and shown as below:

[ ]:
```
scores_df = pd.DataFrame(scores_dict,index=['Precision', 'Recall',␣
↪'F1','Accuracy'])
# print(scores_df.to_latex())
```

	SVM Linear	SVM Polynomial	Decision Tree	Random Forest
Precision	0.7774	0.7749	0.7686	0.7734
Recall	0.6853	0.7093	0.7360	0.7893
F1	0.7092	0.7289	0.7481	0.7784
Accuracy	0.6853	0.7093	0.7481	0.7784

- **Precision :** As we can see in terms of precision approximately all the models have similar performance, while SVMs are a bit better, the difference is less than one percent so we can say all are performing similarly, precision refers to the number of true positives divided by the total number of positive predictions.
- `Precision = True Positive / ( True Positive + False Positive )`
- **Recall :** Random forest has the highest recall, then decision tree, then polynomial version of svm and then linear version, Recall is determined by dividing the total number of Positive samples by the number of Positive samples that were accurately categorized as Positive. The recall gauges how well the model can identify positive samples. Positive samples are found in greater numbers the higher the recall.(www.javatpoint.com, n.d.)
- `Recall = True Positive/ ( True Positive + False Negative )`
- **F1 Score :** F1 score of Random forest is the higher again, F1 score is calculated as the mean of accuracy and recall, where the F1 score is the same as the relative contribution of these two measures. The F1 score has a maximum value of 1 and a minimum value of 0. What does this signify? An F1 score of 1 indicates that all of the predictions were accurate, indicating a flawless model, that of course doesnt happen in reality.(Sharma, 2023)
- `F1 = 2 * Precision * Recall / ( Precision + Recall )`
- **Accuracy :** In terms of accuracy again random forest is our best model, In classification issues, accuracy is mostly utilized to determine the right percentage of models being employed and how much of it is accurately predicted.(www.javatpoint.com, n.d.)
- `Accuracy = Number of correct predictions / Total number of predictions`



5.5.2 Confusion Matrix

A confusion matrix is a table that is used to describe the performance of a classification model on a set of test data for which the true values are known. It allows us to visualize the model's performance by showing how many instances were correctly or incorrectly classified. Below we will see the confusion matrix for each model. Another usage of this matrix is that we can manually calculate the accuracy metrics for our models using the Ture/Flase positives and True/False negatives. I defined a function and called it here for each model (See Here)

[76]:
```
conf_matrx(y_validation,svm_linear_preds,"SVM Linear")
```

```
<IPython.core.display.Javascript object>

Confusion Matrix For SVM Linear
       red  green
red    195     94
green   24     62
```

[77]:
```
conf_matrx(y_validation,svm_preds,"SVM Linear")
```

```
<IPython.core.display.Javascript object>

Confusion Matrix For SVM Linear
       red  green
red    208     81
green   28     58
```

[78]:
```
conf_matrx(y_validation,dec_tree_preds,"SVM Linear")
```

```
<IPython.core.display.Javascript object>

Confusion Matrix For SVM Linear
       red  green
```

```
red     225      64
green    35      51
```

[79]:
```
conf_matrx(y_validation,rf_preds,"Random Forest")
```

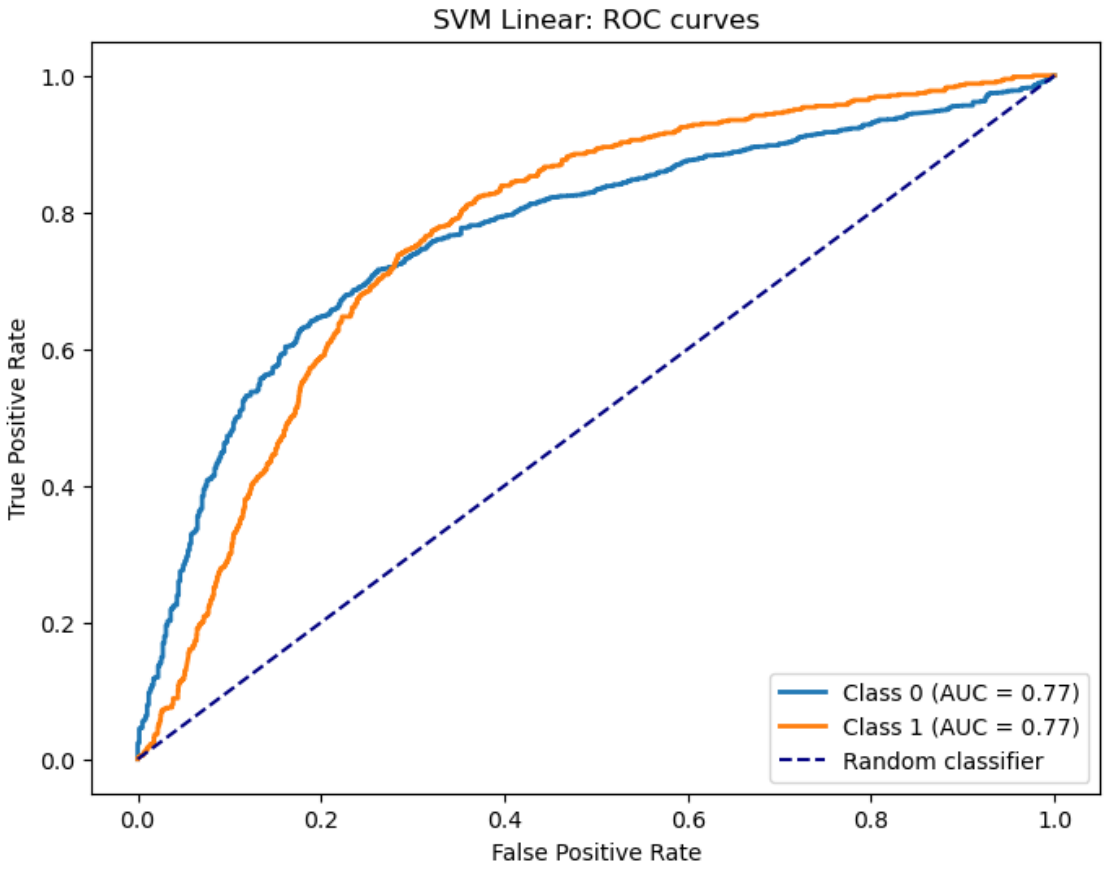
```
<IPython.core.display.Javascript object>
```

```
Confusion Matrix For Random Forest
       red  green
red    266     23
green   50     36
```

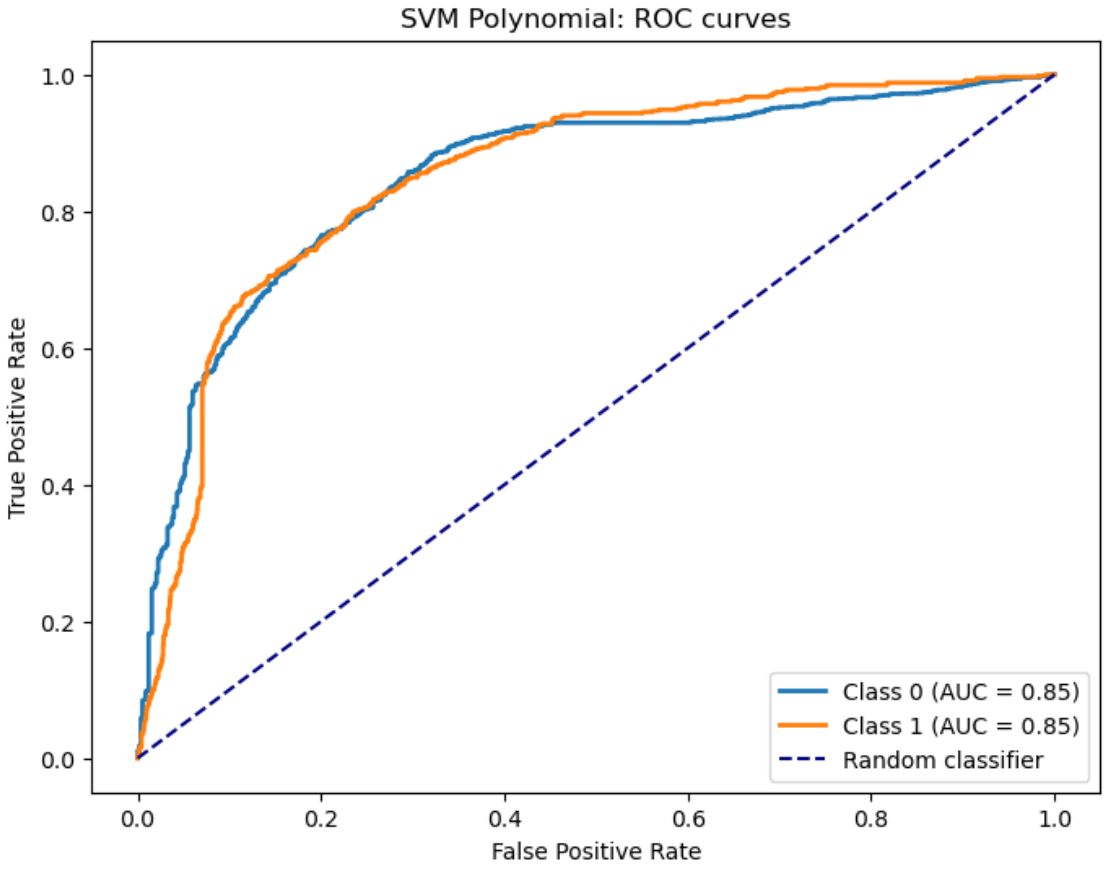
5.5.3 ROC AUC curve

For plotting ROC AUC i defined a function (See Here) and called it here for each model, A ROC curve is a graph that shows a binary classification model's performance at each classification threshold. The true positive rate (TPR) and false positive rate (FPR) are two metrics that are plotted on this curve at various categorization criteria. TPR is shown on the Y-axis and FPR is shown on the X-axis of a ROC curve graph. When the classification threshold is decreased, more observations are classified as positive, which increases TPR and FPR. The AUC that is shown in the legends shows the area under curve, a higher AUC indicates a better performance.

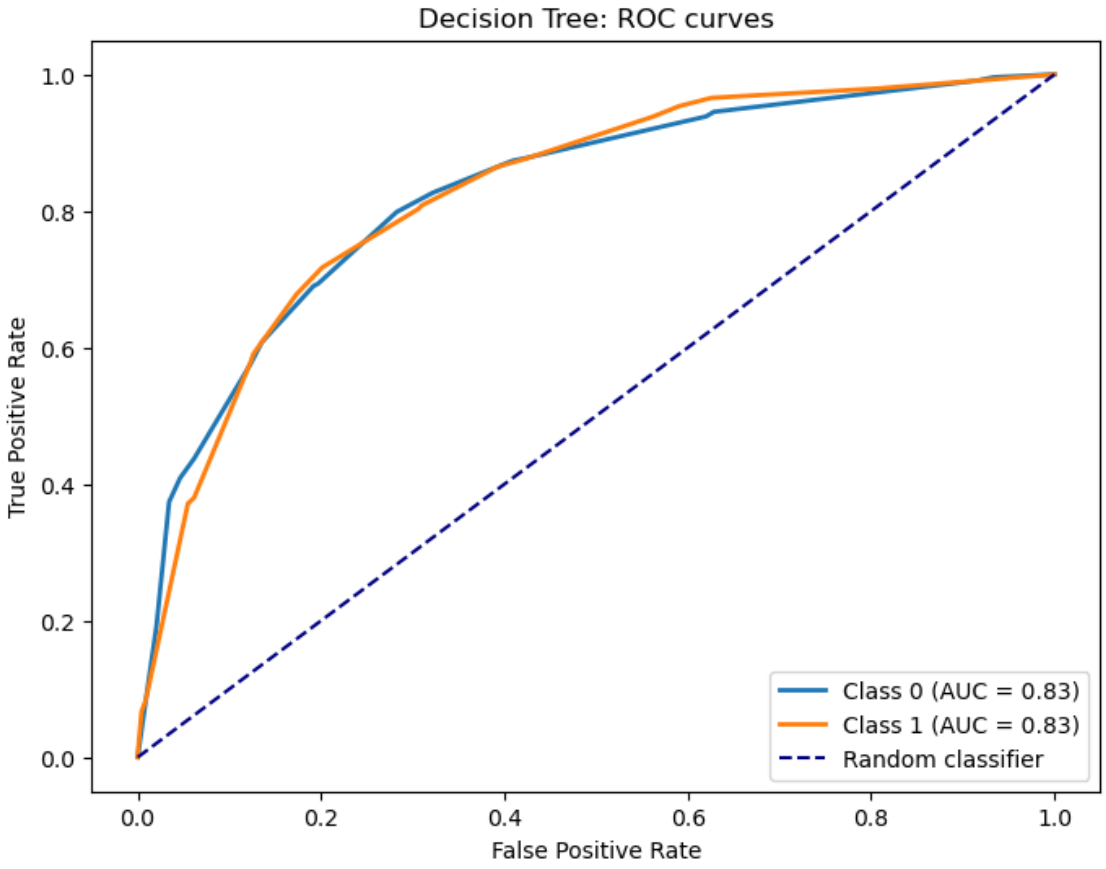
[80]:
```
roc_auc(svm_linear,"SVM Linear",X_scaled,y)
```



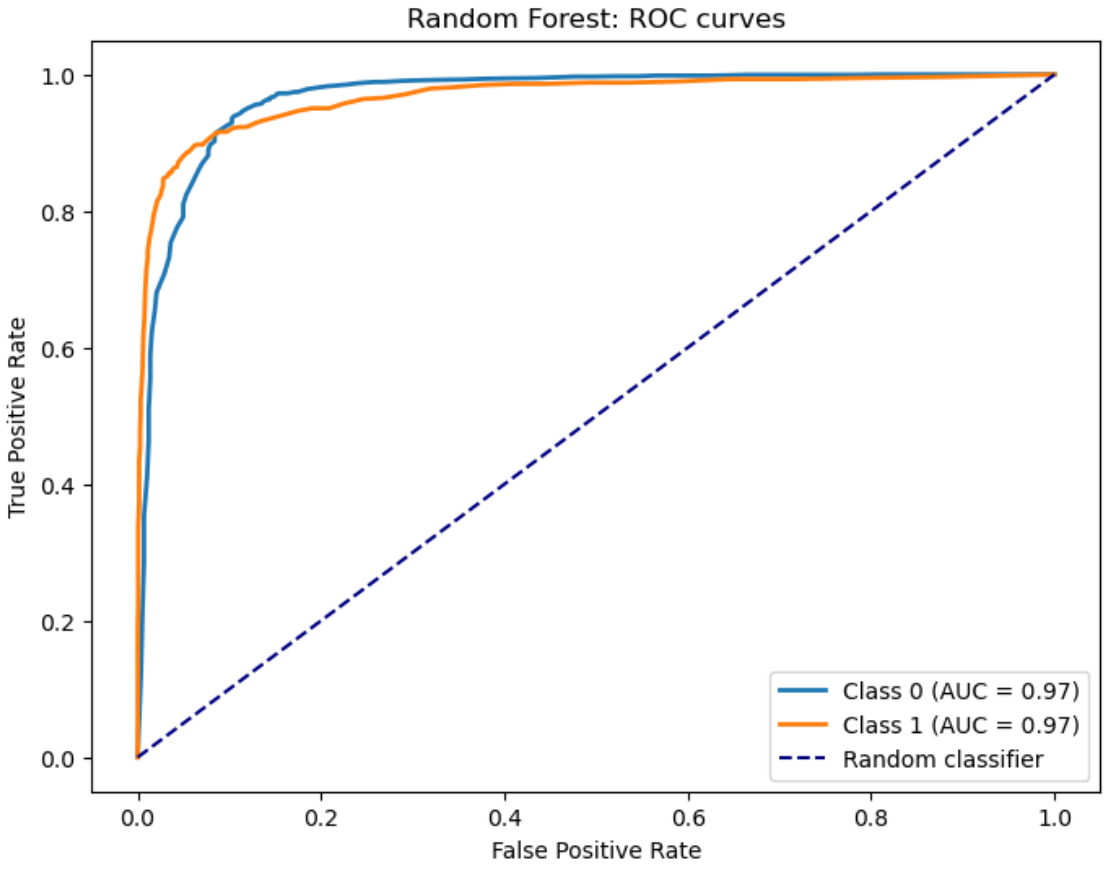
[81]:
```
roc_auc(svm_classifier,"SVM Polynomial",X_scaled,y)
```



[82]:
```
roc_auc(dec_tree,"Decision Tree",X_scaled,y)
```



[83]:
```
roc_auc(rf,"Random Forest",X_scaled,y)
```



We can see in term of area under curve again it is the random forest that has the best performance with a number of 0.97 which is so close to one.

6 DEEP LEARNING ALGORITHMS

The most common algorithm to make a model for image classification in deep learning is convolutional neural network (`CNN`). I will make a basic CNN model with two hidden layers without any dropout, batch normalization and ... to test on three different image types that i created to see which one is much more suitable for the task (If any of them is) then continue to develop it to gain the best result.

6.1 Normalization and Preprocessing

Due to computational limits that i have, and since the resolution of pictures that i created is `600x600` i will resize the pictures to `150x150` and also normalize the pixel values dividing by 255. I will call the function that i created for this. (See here) Reducing the image size also makes the training faster and easier and helps having a less complex architecture.

6.2 Dealing with Class Imbalance

Important thing here is our class imbalance (See Class Imbalance), we need to compute class weights and in fitting step pass it to our model so that the model doesn't learn only the majority class. We can do it in two ways, first we can do it manually by making a dictionary of classes with its weights

weight_i = (total number of samples) / (number of samples in class i)

or use the built in function compute_class_weight to to penalize misclassifications of the minority class more than the majority class, thereby encouraging the model to pay more attention to the minority class.

Another way to deal with this imbalance is using techniques like over sampling and under sampling or the combination of both, i prefer to use over sampling to generate new samples in the minority classes because we have only a few hundred in our minority class. We can implement this in python with imblearn library. The only thing we need to consider is to reshape the input to 2D and then pass it into fit_resample built in function of rus instant that i will create, and then again reshape it to 4D to feed into the CNN model.(guest_blog, 2020)

i will first use the oversampling method because its much more likely that due to our small dataset it can be more helpful.

6.3 First images group

[ ]:
```
# Path to pictures folders
trajectories = f'{extracted_folder}\\trajectories'

# Iterating through log_id column (pics names) instead of each folder content,␣
↪to make sure about the
# sequence using a comprehension
image_files = [os.path.join(trajectories, f'{img}.jpg') for img in␣
↪imgs_and_labels['log_id']]

# Call the function for each image and make a list using a comprehension
images = [preprocess_image(image) for image in image_files]

# Converting to Numpy Array
images = np.array(images)

# Labels
labels = imgs_and_labels['ad_clicked']
```

[ ]:
```
# Train-test split
X_train, X_temp, y_train, y_temp = train_test_split(images, labels, test_size=0.
↪3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5,␣
↪random_state=42)

# Reshape the training images to 2D
```

```
X_train_reshaped = X_train.reshape(X_train.shape[0], -1)

# Instanciate the RandomOverSampler
rus = RandomOverSampler(random_state=42)


# Resample the training dataset
X_train_resampled, y_train_resampled = rus.fit_resample(X_train_reshaped,␣
 ↪y_train)

# Reshape the resampled training images back to 4D
num_samples_resampled = X_train_resampled.shape[0]
X_train_resampled_reshaped = X_train_resampled.reshape(num_samples_resampled,␣
 ↪150, 150, 3)
```

Building my model using keras sequential API, we know for binary classification, output layer activation should be sigmoid because the sigmoid activation function will squash the output between 0 and 1, making it suitable for binary classification. and the loss function should be binary crossentropy. I also picked normal activation function since it is common for these types of tasks. For the optimizer i am using an adaptive one (`Adam`) with the learning rate of `0.00001`.

[86]:
```
print(X_train_resampled_reshaped.shape)
print(y_train_resampled.shape)
print(X_val.shape)
print(y_val.shape)
```

```
(2662, 150, 150, 3)
(2662,)
(375, 150, 150, 3)
(375,)
```

[ ]:
```
model = Sequential()

# Very important to clear the session to avoid mixing up with and being affetced␣
 ↪by previous computations
tf.keras.backend.clear_session()

# First convolutional layers gets the input which is resolution and channels in␣
 ↪our case -> (150,150,3)
# Using MaxPooling => reduce the spatial dimensions of the input,while retaining␣
 ↪the most important information
model.add(Conv2D(64, 3, kernel_initializer='normal', activation='relu',␣
 ↪input_shape=(150,150,3)))
model.add(MaxPooling2D(pool_size=(3, 3), strides=2))

model.add(Conv2D(64, 3, kernel_initializer='normal', activation='relu'))
model.add(MaxPooling2D(pool_size=(3, 3), strides=2))
```

```
model.add(Flatten())

# Adding the dense layer
model.add(Dense(128, activation='relu'))

model.add(Dense(1, activation='sigmoid'))
```

[83]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 148, 148, 64)      1792

 max_pooling2d (MaxPooling2  (None, 73, 73, 64)        0
 D)

 conv2d_1 (Conv2D)           (None, 71, 71, 64)        36928

 max_pooling2d_1 (MaxPoolin  (None, 35, 35, 64)        0
 g2D)

 flatten (Flatten)           (None, 78400)             0

 dense (Dense)               (None, 128)               10035328

 dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 10074177 (38.43 MB)
Trainable params: 10074177 (38.43 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[100]:
```
model.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.00001),␣
 ↪metrics=['accuracy'])

history = model.fit(X_train_resampled_reshaped,
                    y_train_resampled,
                    epochs=20,
                    batch_size=32,
                    validation_data=(X_val, y_val))
```

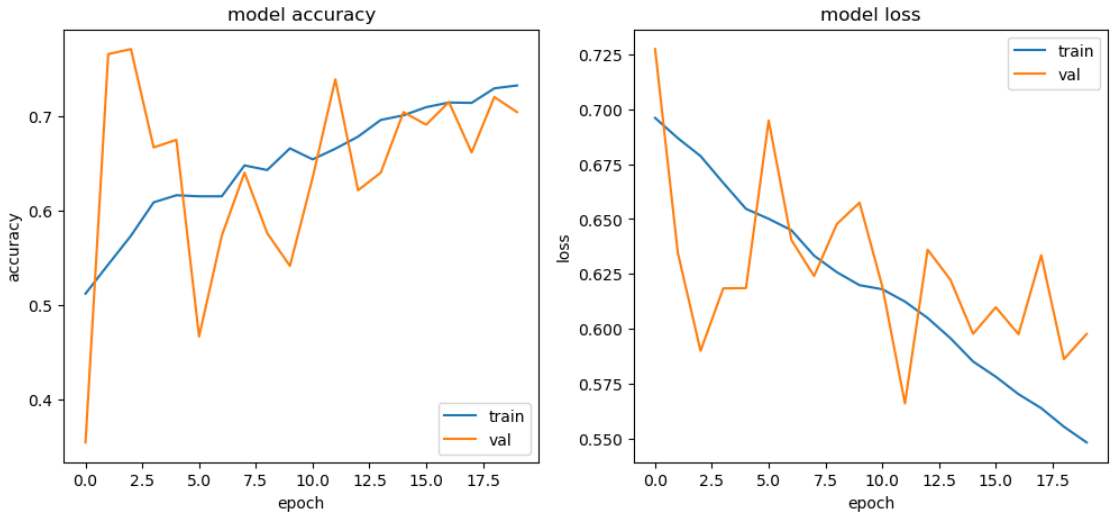
```
Epoch 1/20
84/84 [==============================] - 44s 503ms/step - loss: 0.6960 -
accuracy: 0.5120 - val_loss: 0.7273 - val_accuracy: 0.3547
Epoch 2/20
```

```
84/84 [==============================] - 39s 459ms/step - loss: 0.6868 -
accuracy: 0.5428 - val_loss: 0.6346 - val_accuracy: 0.7653
Epoch 3/20
84/84 [==============================] - 41s 488ms/step - loss: 0.6786 -
accuracy: 0.5733 - val_loss: 0.5900 - val_accuracy: 0.7707
Epoch 4/20
84/84 [==============================] - 40s 476ms/step - loss: 0.6664 -
accuracy: 0.6086 - val_loss: 0.6184 - val_accuracy: 0.6667
Epoch 5/20
84/84 [==============================] - 41s 493ms/step - loss: 0.6546 -
accuracy: 0.6161 - val_loss: 0.6186 - val_accuracy: 0.6747
Epoch 6/20
84/84 [==============================] - 45s 540ms/step - loss: 0.6501 -
accuracy: 0.6150 - val_loss: 0.6948 - val_accuracy: 0.4667
Epoch 7/20
84/84 [==============================] - 42s 506ms/step - loss: 0.6449 -
accuracy: 0.6150 - val_loss: 0.6405 - val_accuracy: 0.5733
Epoch 8/20
84/84 [==============================] - 42s 496ms/step - loss: 0.6333 -
accuracy: 0.6476 - val_loss: 0.6241 - val_accuracy: 0.6400
Epoch 9/20
84/84 [==============================] - 42s 497ms/step - loss: 0.6258 -
accuracy: 0.6427 - val_loss: 0.6477 - val_accuracy: 0.5760
Epoch 10/20
84/84 [==============================] - 40s 481ms/step - loss: 0.6199 -
accuracy: 0.6657 - val_loss: 0.6575 - val_accuracy: 0.5413
Epoch 11/20
84/84 [==============================] - 41s 493ms/step - loss: 0.6181 -
accuracy: 0.6540 - val_loss: 0.6194 - val_accuracy: 0.6347
Epoch 12/20
84/84 [==============================] - 39s 470ms/step - loss: 0.6124 -
accuracy: 0.6653 - val_loss: 0.5661 - val_accuracy: 0.7387
Epoch 13/20
84/84 [==============================] - 40s 481ms/step - loss: 0.6050 -
accuracy: 0.6781 - val_loss: 0.6360 - val_accuracy: 0.6213
Epoch 14/20
84/84 [==============================] - 40s 475ms/step - loss: 0.5958 -
accuracy: 0.6957 - val_loss: 0.6224 - val_accuracy: 0.6400
Epoch 15/20
84/84 [==============================] - 39s 471ms/step - loss: 0.5852 -
accuracy: 0.7006 - val_loss: 0.5978 - val_accuracy: 0.7040
Epoch 16/20
84/84 [==============================] - 39s 460ms/step - loss: 0.5783 -
accuracy: 0.7092 - val_loss: 0.6099 - val_accuracy: 0.6907
Epoch 17/20
84/84 [==============================] - 39s 466ms/step - loss: 0.5703 -
accuracy: 0.7141 - val_loss: 0.5976 - val_accuracy: 0.7147
Epoch 18/20
```

```
84/84 [==============================] - 39s 460ms/step - loss: 0.5640 -
accuracy: 0.7137 - val_loss: 0.6335 - val_accuracy: 0.6613
Epoch 19/20
84/84 [==============================] - 40s 478ms/step - loss: 0.5555 -
accuracy: 0.7292 - val_loss: 0.5863 - val_accuracy: 0.7200
Epoch 20/20
84/84 [==============================] - 41s 487ms/step - loss: 0.5484 -
accuracy: 0.7322 - val_loss: 0.5977 - val_accuracy: 0.7040
```

[101]:
```
plot_results(history)
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

Maximum Loss : 0.6960

Minimum Loss : 0.5484

Loss difference : 0.1476
```



We can see the loss decreases with increasing epochs, showing that the model is learning to predict better.The accuracy is increasing with epochs, which means that the model is becoming more accurate at classifying the images correctly. Training loss decreases, and training accuracy increases, which means that the model is learning from the training data.

Validation Loss and Accuracy are used to estimate how the model is generalizing to unseen data. The validation loss keeps decreasing but appears to be fluctuating after some epochs, and the accuracy of the validation data keeps increasing in the beginning and then stabilizes. The model achieves about 70.40% validation accuracy at the end of the training.

Overall the model has a validation accuracy of about 70.40% after 20 epochs, At least it is doing better than randomly guessing the classes (since it is a binary classification, 50% accuracy). How-

ever, it might not be at the level expected from this model. Validation accuracy is higher than the training accuracy, which means that the model is not overfitting the training data.

The validation accuracy could be further improved by maybe adjusting the architecture of the model, its hyperparameters, or increasing the size and variety in the training data. Also, a possibility is that additional regularization techniques might assist the model.

On the whole, even though the model is learning from the data, it is still far from perfect to get a very good accuracy on the validation set.

6.4 Second images group

[ ]:
```
# Path to pictures folders
trajectories1 = f'{extracted_folder}\\trajectories1'

# Iterating through log_id column (pics names) instead of each folder content,␣
↪to make sure about the
# sequence using a comprehension
image_files1 = [os.path.join(trajectories1, f'{img}.jpg') for img in␣
↪imgs_and_labels['log_id']]

# Call the function for each image and make a list using a comprehension
images1 = [preprocess_image(file_path) for file_path in image_files]

# Converting to Numpy Array
images1 = np.array(images1)
```

[103]:
```
tf.keras.backend.clear_session()

# Train-test split
X_train, X_temp, y_train, y_temp = train_test_split(images1, labels, test_size=0.
↪3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5,␣
↪random_state=42)

# Reshape the training images to 2D
X_train_reshaped = X_train.reshape(X_train.shape[0], -1)

# Instanciate the RandomOverSampler
rus = RandomOverSampler(random_state=42)


# Resample the training dataset
X_train_resampled, y_train_resampled = rus.fit_resample(X_train_reshaped,␣
↪y_train)

# Reshape the resampled training images back to 4D
num_samples_resampled = X_train_resampled.shape[0]
```

```
X_train_resampled_reshaped = X_train_resampled.reshape(num_samples_resampled,␣
↪150, 150, 3)
```

<IPython.core.display.Javascript object>

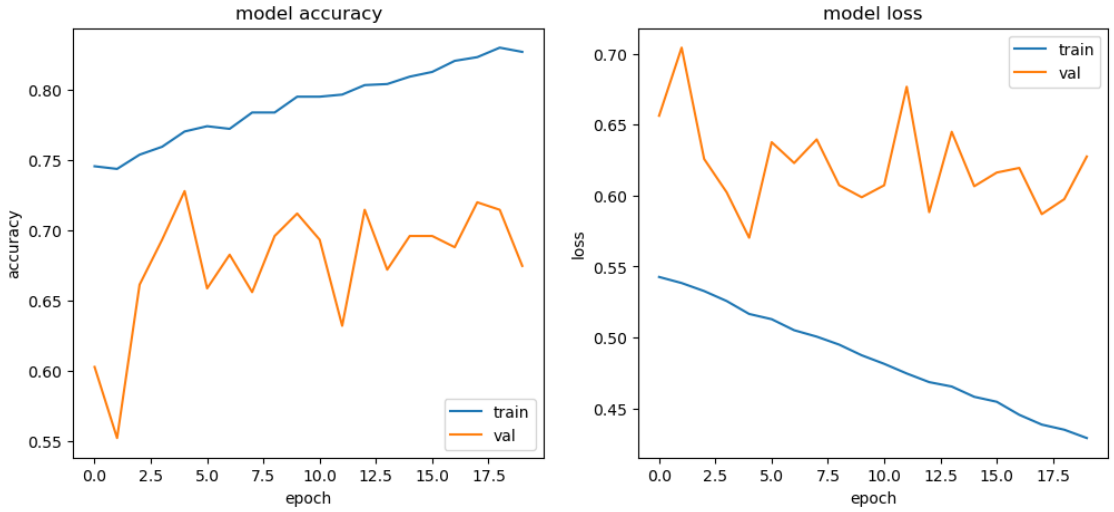
[104]:
```
model.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.00001),␣
↪metrics=['accuracy'])

history = model.fit(X_train_resampled_reshaped,
                    y_train_resampled,
                    epochs=20,
                    batch_size=32,
                    validation_data=(X_val, y_val))
```

```
Epoch 1/20
84/84 [==============================] - 44s 484ms/step - loss: 0.5427 -
accuracy: 0.7457 - val_loss: 0.6562 - val_accuracy: 0.6027
Epoch 2/20
84/84 [==============================] - 40s 477ms/step - loss: 0.5384 -
accuracy: 0.7438 - val_loss: 0.7042 - val_accuracy: 0.5520
Epoch 3/20
84/84 [==============================] - 41s 484ms/step - loss: 0.5327 -
accuracy: 0.7539 - val_loss: 0.6258 - val_accuracy: 0.6613
Epoch 4/20
84/84 [==============================] - 41s 489ms/step - loss: 0.5257 -
accuracy: 0.7596 - val_loss: 0.6023 - val_accuracy: 0.6933
Epoch 5/20
84/84 [==============================] - 44s 523ms/step - loss: 0.5166 -
accuracy: 0.7705 - val_loss: 0.5703 - val_accuracy: 0.7280
Epoch 6/20
84/84 [==============================] - 41s 489ms/step - loss: 0.5129 -
accuracy: 0.7742 - val_loss: 0.6376 - val_accuracy: 0.6587
Epoch 7/20
84/84 [==============================] - 40s 482ms/step - loss: 0.5051 -
accuracy: 0.7724 - val_loss: 0.6229 - val_accuracy: 0.6827
Epoch 8/20
84/84 [==============================] - 41s 488ms/step - loss: 0.5006 -
accuracy: 0.7840 - val_loss: 0.6395 - val_accuracy: 0.6560
Epoch 9/20
84/84 [==============================] - 39s 468ms/step - loss: 0.4950 -
accuracy: 0.7840 - val_loss: 0.6073 - val_accuracy: 0.6960
Epoch 10/20
84/84 [==============================] - 39s 468ms/step - loss: 0.4875 -
accuracy: 0.7953 - val_loss: 0.5988 - val_accuracy: 0.7120
Epoch 11/20
84/84 [==============================] - 39s 466ms/step - loss: 0.4815 -
accuracy: 0.7953 - val_loss: 0.6072 - val_accuracy: 0.6933
Epoch 12/20
```

```
84/84 [==============================] - 40s 471ms/step - loss: 0.4747 -
accuracy: 0.7968 - val_loss: 0.6766 - val_accuracy: 0.6320
Epoch 13/20
84/84 [==============================] - 41s 487ms/step - loss: 0.4686 -
accuracy: 0.8035 - val_loss: 0.5883 - val_accuracy: 0.7147
Epoch 14/20
84/84 [==============================] - 39s 465ms/step - loss: 0.4655 -
accuracy: 0.8043 - val_loss: 0.6449 - val_accuracy: 0.6720
Epoch 15/20
84/84 [==============================] - 40s 473ms/step - loss: 0.4582 -
accuracy: 0.8095 - val_loss: 0.6066 - val_accuracy: 0.6960
Epoch 16/20
84/84 [==============================] - 41s 486ms/step - loss: 0.4547 -
accuracy: 0.8129 - val_loss: 0.6162 - val_accuracy: 0.6960
Epoch 17/20
84/84 [==============================] - 40s 473ms/step - loss: 0.4456 -
accuracy: 0.8208 - val_loss: 0.6194 - val_accuracy: 0.6880
Epoch 18/20
84/84 [==============================] - 39s 465ms/step - loss: 0.4387 -
accuracy: 0.8234 - val_loss: 0.5869 - val_accuracy: 0.7200
Epoch 19/20
84/84 [==============================] - 40s 481ms/step - loss: 0.4351 -
accuracy: 0.8302 - val_loss: 0.5975 - val_accuracy: 0.7147
Epoch 20/20
84/84 [==============================] - 39s 462ms/step - loss: 0.4292 -
accuracy: 0.8272 - val_loss: 0.6275 - val_accuracy: 0.6747
```

[106]:
```
plot_results(history)
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

Maximum Loss : 0.5427

Minimum Loss : 0.4292

Loss difference : 0.1135
```



In general loss decreases with an increase in epochs, which means the model is improving in minimizing the errors. Accuracy increases with an increase in epochs, but since the validation loss is higher than train loss in every point and the difference is too high it can be a clear sign of over fitting

While the model is learning from the data, there seems to be some room for improvement to reach higher accuracy on the validation set. The validation accuracy is higher than random guessing—which is 50% accuracy—indicating the model learned some meaningful patterns in the data.

Overall in comparison with first category images i would pick the first category for further work and improving the performance.

6.5 Third images group

[ ]:
```
# Path to pictures folders
trajectories2 = f'{extracted_folder}\\trajectories2'

# Iterating through log_id column (pics names) instead of each folder content,␣
 ↪to make sure about the
# sequence using a comprehension
image_files2 = [os.path.join(trajectories2, f'{img}.jpg') for img in␣
 ↪imgs_and_labels['log_id']]

# Call the function for each image and make a list using a comprehension
images2 = [preprocess_image(file_path) for file_path in image_files]

# Converting to Numpy Array
images2 = np.array(images2)
```

[ ]:
```
tf.keras.backend.clear_session()

# Train-test split
X_train, X_temp, y_train, y_temp = train_test_split(images2, labels, test_size=0.
↪3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5,␣
↪random_state=42)

# Reshape the training images to 2D
X_train_reshaped = X_train.reshape(X_train.shape[0], -1)

# Instanciate the RandomOverSampler
rus = RandomOverSampler(random_state=42)


# Resample the training dataset
X_train_resampled, y_train_resampled = rus.fit_resample(X_train_reshaped,␣
↪y_train)

# Reshape the resampled training images back to 4D
num_samples_resampled = X_train_resampled.shape[0]
X_train_resampled_reshaped = X_train_resampled.reshape(num_samples_resampled,␣
↪150, 150, 3)
```

[110]:
```
model.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.00001),␣
↪metrics=['accuracy'])

history = model.fit(X_train_resampled_reshaped,
                    y_train_resampled,
                    epochs=20,
                    batch_size=32,
                    validation_data=(X_val, y_val))
```

```
Epoch 1/20
84/84 [==============================] - 39s 434ms/step - loss: 0.4263 -
accuracy: 0.8264 - val_loss: 0.6605 - val_accuracy: 0.6560
Epoch 2/20
84/84 [==============================] - 42s 499ms/step - loss: 0.4192 -
accuracy: 0.8347 - val_loss: 0.6321 - val_accuracy: 0.6800
Epoch 3/20
84/84 [==============================] - 38s 447ms/step - loss: 0.4131 -
accuracy: 0.8437 - val_loss: 0.6076 - val_accuracy: 0.7120
Epoch 4/20
84/84 [==============================] - 37s 436ms/step - loss: 0.4070 -
accuracy: 0.8381 - val_loss: 0.7140 - val_accuracy: 0.6080
Epoch 5/20
84/84 [==============================] - 36s 427ms/step - loss: 0.4064 -
```

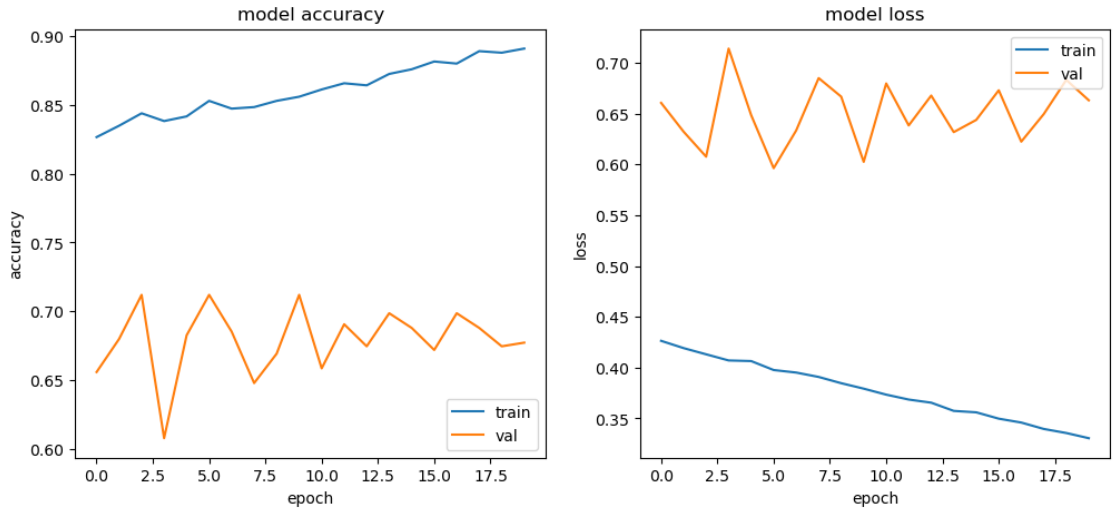
```
accuracy: 0.8415 - val_loss: 0.6485 - val_accuracy: 0.6827
Epoch 6/20
84/84 [==============================] - 36s 427ms/step - loss: 0.3976 -
accuracy: 0.8527 - val_loss: 0.5962 - val_accuracy: 0.7120
Epoch 7/20
84/84 [==============================] - 36s 429ms/step - loss: 0.3951 -
accuracy: 0.8471 - val_loss: 0.6333 - val_accuracy: 0.6853
Epoch 8/20
84/84 [==============================] - 37s 437ms/step - loss: 0.3908 -
accuracy: 0.8482 - val_loss: 0.6849 - val_accuracy: 0.6480
Epoch 9/20
84/84 [==============================] - 36s 432ms/step - loss: 0.3846 -
accuracy: 0.8527 - val_loss: 0.6666 - val_accuracy: 0.6693
Epoch 10/20
84/84 [==============================] - 40s 477ms/step - loss: 0.3793 -
accuracy: 0.8557 - val_loss: 0.6024 - val_accuracy: 0.7120
Epoch 11/20
84/84 [==============================] - 40s 475ms/step - loss: 0.3734 -
accuracy: 0.8610 - val_loss: 0.6796 - val_accuracy: 0.6587
Epoch 12/20
84/84 [==============================] - 35s 417ms/step - loss: 0.3685 -
accuracy: 0.8655 - val_loss: 0.6383 - val_accuracy: 0.6907
Epoch 13/20
84/84 [==============================] - 36s 431ms/step - loss: 0.3655 -
accuracy: 0.8640 - val_loss: 0.6677 - val_accuracy: 0.6747
Epoch 14/20
84/84 [==============================] - 32s 378ms/step - loss: 0.3574 -
accuracy: 0.8723 - val_loss: 0.6317 - val_accuracy: 0.6987
Epoch 15/20
84/84 [==============================] - 29s 344ms/step - loss: 0.3560 -
accuracy: 0.8757 - val_loss: 0.6438 - val_accuracy: 0.6880
Epoch 16/20
84/84 [==============================] - 27s 325ms/step - loss: 0.3497 -
accuracy: 0.8813 - val_loss: 0.6729 - val_accuracy: 0.6720
Epoch 17/20
84/84 [==============================] - 30s 361ms/step - loss: 0.3459 -
accuracy: 0.8798 - val_loss: 0.6223 - val_accuracy: 0.6987
Epoch 18/20
84/84 [==============================] - 32s 382ms/step - loss: 0.3396 -
accuracy: 0.8888 - val_loss: 0.6495 - val_accuracy: 0.6880
Epoch 19/20
84/84 [==============================] - 32s 381ms/step - loss: 0.3356 -
accuracy: 0.8877 - val_loss: 0.6831 - val_accuracy: 0.6747
Epoch 20/20
84/84 [==============================] - 33s 388ms/step - loss: 0.3305 -
accuracy: 0.8907 - val_loss: 0.6630 - val_accuracy: 0.6773
```

[111]:
```
plot_results(history)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

```
Maximum Loss : 0.4263

Minimum Loss : 0.3305

Loss difference : 0.0958
```



Third category is exactly the same as the second category and i think the final decision is to pick the first category and start improving the model by hyper parameter tuning and modifying the architecture

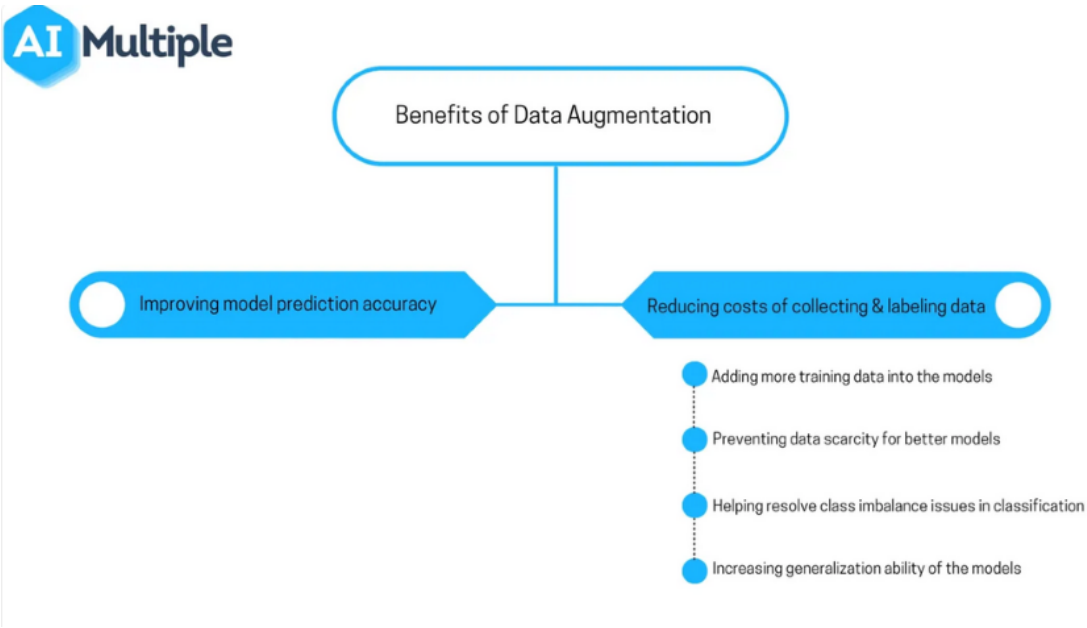
6.6 Model improvement

First i need to do the train test split again because it has been overwritten by the second and third category data.

[132]:
```
# Train-test split
X_train, X_temp, y_train, y_temp = train_test_split(images, labels, test_size=0.
↪3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5,␣
↪random_state=42)

# Reshape the training images to 2D
X_train_reshaped = X_train.reshape(X_train.shape[0], -1)

# Instanciate the RandomOverSampler
```

```
rus = RandomOverSampler(random_state=42)


# Resample the training dataset
X_train_resampled, y_train_resampled = rus.fit_resample(X_train_reshaped,␣
 ↪y_train)

# Reshape the resampled training images back to 4D
num_samples_resampled = X_train_resampled.shape[0]
X_train_resampled_reshaped = X_train_resampled.reshape(num_samples_resampled,␣
 ↪150, 150, 3)
```

First improvement for me is to try data augmentation, because i think huge part of performance of machine learning and deep learning models depends on the quality and sufficiency of the data, in our case the data set is too small and we can easily add more samples for our model to learn from. I will implement this using python ImageDataGenerator library.(Takimoglu, 2021)



[86]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 148, 148, 64)      1792

 max_pooling2d (MaxPooling2  (None, 73, 73, 64)        0
 D)

 conv2d_1 (Conv2D)           (None, 71, 71, 64)        36928
```

```
 max_pooling2d_1 (MaxPoolin  (None, 35, 35, 64)        0
 g2D)

 flatten (Flatten)           (None, 78400)             0

 dense (Dense)               (None, 128)               10035328

 dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 10074177 (38.43 MB)
Trainable params: 10074177 (38.43 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[91]:
```
# Compile the model
model.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.00001),␣
 ↪metrics=['accuracy'])

# Create an instance of ImageDataGenerator for data augmentation
datagen = ImageDataGenerator(
    rotation_range=20,      # Randomly rotate images in the range (degrees, 0 to␣
 ↪180)
    width_shift_range=0.1,  # Randomly shift images horizontally (fraction of␣
 ↪total width)
    height_shift_range=0.1, # Randomly shift images vertically (fraction of␣
 ↪total height)
    horizontal_flip=True,   # Randomly flip images horizontally
    vertical_flip=False     # Don't flip images vertically
)

# Compute quantities required for feature-wise normalization
# (std, mean, and principal components if ZCA whitening is applied)
datagen.fit(X_train_resampled_reshaped)

# Train the model with augmented data
history = model.fit_generator(
    datagen.flow(X_train_resampled_reshaped, y_train_resampled, batch_size=32),
    steps_per_epoch=len(X_train_resampled_reshaped) / 32,
    epochs=20,
    validation_data=(X_val, y_val)
)
```

```
Epoch 1/20
WARNING:tensorflow:From C:\Users\malir\anaconda3\lib\site-
packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue
is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.
```

```
WARNING:tensorflow:From C:\Users\malir\anaconda3\lib\site-
packages\keras\src\engine\base_layer_utils.py:384: The name
tf.executing_eagerly_outside_functions is deprecated. Please use
tf.compat.v1.executing_eagerly_outside_functions instead.

83/83 [==============================] - 44s 503ms/step - loss: 0.6968 -
accuracy: 0.5041 - val_loss: 0.7450 - val_accuracy: 0.2293
Epoch 2/20
83/83 [==============================] - 44s 530ms/step - loss: 0.6959 -
accuracy: 0.5128 - val_loss: 0.7162 - val_accuracy: 0.5413
Epoch 3/20
83/83 [==============================] - 42s 504ms/step - loss: 0.6961 -
accuracy: 0.5131 - val_loss: 0.7104 - val_accuracy: 0.5093
Epoch 4/20
83/83 [==============================] - 44s 523ms/step - loss: 0.6928 -
accuracy: 0.5244 - val_loss: 0.7062 - val_accuracy: 0.5707
Epoch 5/20
83/83 [==============================] - 45s 543ms/step - loss: 0.6915 -
accuracy: 0.5267 - val_loss: 0.7186 - val_accuracy: 0.5307
Epoch 6/20
83/83 [==============================] - 44s 533ms/step - loss: 0.6935 -
accuracy: 0.5274 - val_loss: 0.6496 - val_accuracy: 0.5787
Epoch 7/20
83/83 [==============================] - 45s 533ms/step - loss: 0.6913 -
accuracy: 0.5308 - val_loss: 0.7062 - val_accuracy: 0.5493
Epoch 8/20
83/83 [==============================] - 38s 452ms/step - loss: 0.6906 -
accuracy: 0.5263 - val_loss: 0.6794 - val_accuracy: 0.5787
Epoch 9/20
83/83 [==============================] - 38s 460ms/step - loss: 0.6933 -
accuracy: 0.5237 - val_loss: 0.7104 - val_accuracy: 0.5147
Epoch 10/20
83/83 [==============================] - 44s 528ms/step - loss: 0.6913 -
accuracy: 0.5293 - val_loss: 0.6890 - val_accuracy: 0.5787
Epoch 11/20
83/83 [==============================] - 42s 500ms/step - loss: 0.6913 -
accuracy: 0.5237 - val_loss: 0.6458 - val_accuracy: 0.5787
Epoch 12/20
83/83 [==============================] - 41s 496ms/step - loss: 0.6927 -
accuracy: 0.5342 - val_loss: 0.7352 - val_accuracy: 0.3067
Epoch 13/20
83/83 [==============================] - 41s 495ms/step - loss: 0.6906 -
accuracy: 0.5338 - val_loss: 0.6715 - val_accuracy: 0.5760
Epoch 14/20
83/83 [==============================] - 40s 482ms/step - loss: 0.6901 -
accuracy: 0.5379 - val_loss: 0.7458 - val_accuracy: 0.2293
Epoch 15/20
```

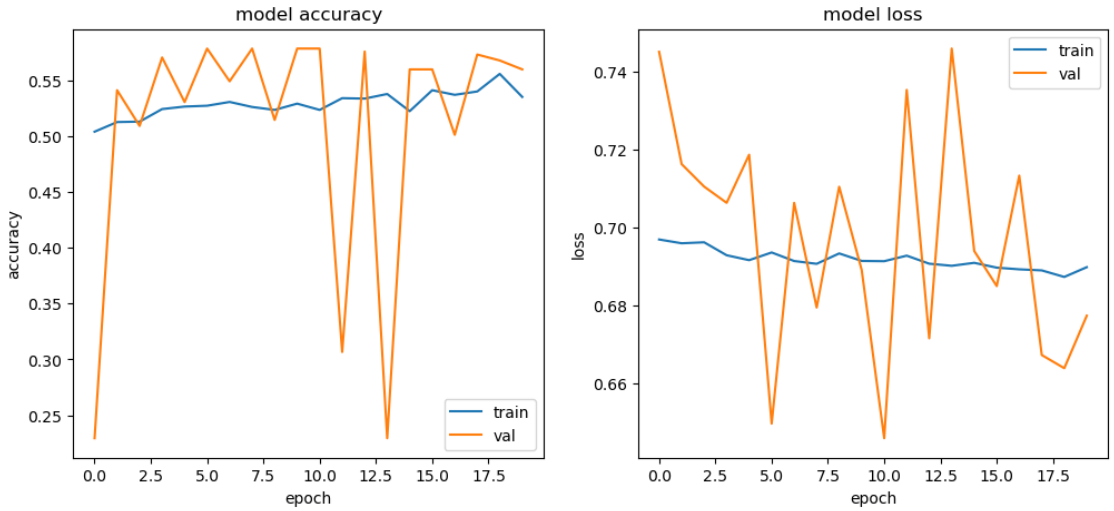
```
83/83 [==============================] - 41s 486ms/step - loss: 0.6908 -
accuracy: 0.5225 - val_loss: 0.6939 - val_accuracy: 0.5600
Epoch 16/20
83/83 [==============================] - 41s 496ms/step - loss: 0.6896 -
accuracy: 0.5413 - val_loss: 0.6849 - val_accuracy: 0.5600
Epoch 17/20
83/83 [==============================] - 41s 488ms/step - loss: 0.6892 -
accuracy: 0.5372 - val_loss: 0.7132 - val_accuracy: 0.5013
Epoch 18/20
83/83 [==============================] - 42s 498ms/step - loss: 0.6889 -
accuracy: 0.5402 - val_loss: 0.6672 - val_accuracy: 0.5733
Epoch 19/20
83/83 [==============================] - 43s 519ms/step - loss: 0.6872 -
accuracy: 0.5560 - val_loss: 0.6638 - val_accuracy: 0.5680
Epoch 20/20
83/83 [==============================] - 43s 517ms/step - loss: 0.6897 -
accuracy: 0.5353 - val_loss: 0.6773 - val_accuracy: 0.5600
```

[92]:
```
plot_results(history)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

```
Maximum Loss : 0.6968

Minimum Loss : 0.6872

Loss difference : 0.0096
```



We can see although the train accuracy is in creasing a bit but validation accuracy is fluctuating and we don't know what will be behaviour if the training continues, same for the loss function,

although we can see generally it has been decreased for both train and validation, the fluctuation problem still persists, that can be a sign of possible overfitting, so in addition to data augmentation, we need to consider more modifications.

The main purpose is to make the loss function constantly decreasing and the accurcy constantly increasing and get make the accuracy as high as possible while making the loss as low as possible.To achive this we already know that playing around ith some hyperparametes like smaller batch size, callback methods such as early stopping, batch normalization,adding L1 and L2 regulizers and increasing dropouts are the techniques that can help us to decrease over fitting.

So for the first modification i will add one other convolutional layer and add batch normalization to each layer, one dense layer with drop out with 0.5 rate to each layer, from the callbacks i will pick early stopping with patience=5.

[103]:
```
# Clearing session
tf.keras.backend.clear_session()

# Data Augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')

validation_datagen = ImageDataGenerator(rescale=1./255)

# Flow training images in batches of 32 using train_datagen generator
train_generator = train_datagen.flow(X_train_resampled_reshaped,␣
 ↪y_train_resampled, batch_size=32)

# Flow validation images in batches of 32 using validation_datagen generator
validation_generator = validation_datagen.flow(X_val, y_val, batch_size=32)

# Model Architecture
model = Sequential()

model.add(Conv2D(64, 3, kernel_initializer='he_uniform', activation='relu',␣
 ↪input_shape=(150,150,3)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(3, 3), strides=2))

model.add(Conv2D(64, 3, kernel_initializer='he_uniform', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(3, 3), strides=2))
```

```
model.add(Conv2D(128, 3, kernel_initializer='he_uniform', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(3, 3), strides=2))

model.add(Flatten())

model.add(Dense(256, activation='relu', kernel_initializer='he_uniform'))
model.add(Dropout(0.5))

model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))

model.add(Dense(1, activation='sigmoid'))

# Compile model
model.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.0001),␣
 ↪metrics=['accuracy'])

# Early stopping to prevent overfitting
early_stop = EarlyStopping(monitor='val_loss', patience=5,␣
 ↪restore_best_weights=True)

# Training the model
history = model.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=30,
    validation_data=validation_generator,
    validation_steps=len(validation_generator),
    callbacks=[early_stop]
)
```

```
<IPython.core.display.Javascript object>
```

```
Epoch 1/30
84/84 [==============================] - 51s 586ms/step - loss: 1.6749 -
accuracy: 0.5210 - val_loss: 1.3862 - val_accuracy: 0.2293
Epoch 2/30
84/84 [==============================] - 49s 579ms/step - loss: 1.0851 -
accuracy: 0.5180 - val_loss: 1.6340 - val_accuracy: 0.2293
Epoch 3/30
84/84 [==============================] - 50s 594ms/step - loss: 0.8782 -
accuracy: 0.5192 - val_loss: 2.1211 - val_accuracy: 0.2293
Epoch 4/30
84/84 [==============================] - 50s 591ms/step - loss: 0.8144 -
accuracy: 0.5041 - val_loss: 0.7816 - val_accuracy: 0.2880
Epoch 5/30
84/84 [==============================] - 50s 593ms/step - loss: 0.7689 -
```

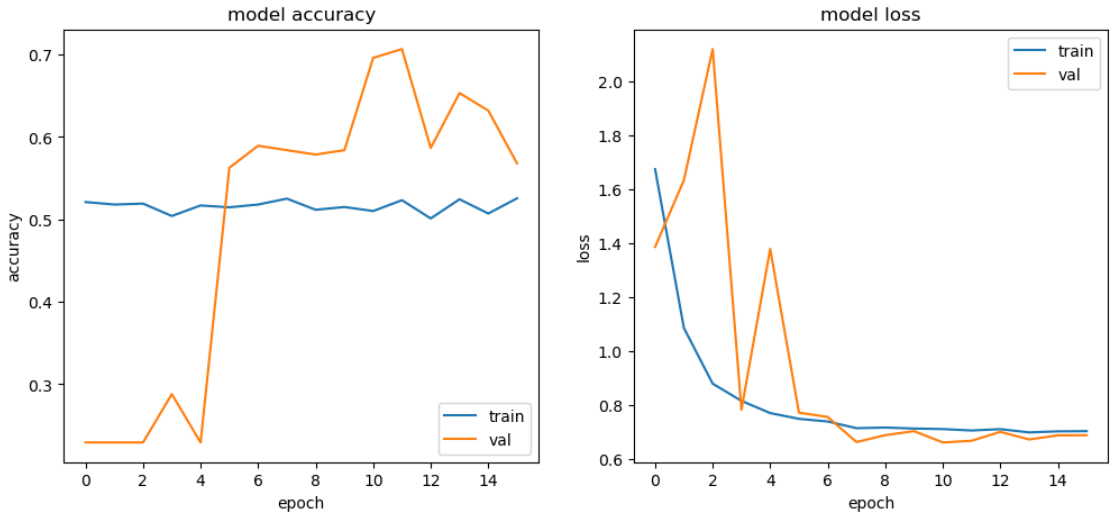
```
accuracy: 0.5169 - val_loss: 1.3789 - val_accuracy: 0.2293
Epoch 6/30
84/84 [==============================] - 49s 579ms/step - loss: 0.7476 -
accuracy: 0.5147 - val_loss: 0.7705 - val_accuracy: 0.5627
Epoch 7/30
84/84 [==============================] - 49s 587ms/step - loss: 0.7376 -
accuracy: 0.5180 - val_loss: 0.7545 - val_accuracy: 0.5893
Epoch 8/30
84/84 [==============================] - 50s 596ms/step - loss: 0.7129 -
accuracy: 0.5252 - val_loss: 0.6614 - val_accuracy: 0.5840
Epoch 9/30
84/84 [==============================] - 52s 611ms/step - loss: 0.7151 -
accuracy: 0.5116 - val_loss: 0.6869 - val_accuracy: 0.5787
Epoch 10/30
84/84 [==============================] - 51s 599ms/step - loss: 0.7115 -
accuracy: 0.5150 - val_loss: 0.7019 - val_accuracy: 0.5840
Epoch 11/30
84/84 [==============================] - 54s 641ms/step - loss: 0.7095 -
accuracy: 0.5101 - val_loss: 0.6598 - val_accuracy: 0.6960
Epoch 12/30
84/84 [==============================] - 49s 586ms/step - loss: 0.7043 -
accuracy: 0.5233 - val_loss: 0.6660 - val_accuracy: 0.7067
Epoch 13/30
84/84 [==============================] - 50s 592ms/step - loss: 0.7092 -
accuracy: 0.5011 - val_loss: 0.6993 - val_accuracy: 0.5867
Epoch 14/30
84/84 [==============================] - 49s 579ms/step - loss: 0.6971 -
accuracy: 0.5244 - val_loss: 0.6709 - val_accuracy: 0.6533
Epoch 15/30
84/84 [==============================] - 50s 597ms/step - loss: 0.7010 -
accuracy: 0.5071 - val_loss: 0.6862 - val_accuracy: 0.6320
Epoch 16/30
84/84 [==============================] - 50s 592ms/step - loss: 0.7019 -
accuracy: 0.5255 - val_loss: 0.6866 - val_accuracy: 0.5680
```

[104]:
```
plot_results(history)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

```
Maximum Loss : 1.6749

Minimum Loss : 0.6971

Loss difference : 0.9778
```



We can see our model after 16 epochs started to have a better loss function and the accuracy for validation increased to some extent but still have the same problem, so i need to figure something out to fix the accuracy for train and validation. I will add `kernel_regularizer=l2(0.001)` to the layers as we mentioned earlier to prevent overfitting, and add the drop out with the same rate to the second dense layer as well.

[112]:
```
# Clearing session
tf.keras.backend.clear_session()

# Data Augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')

validation_datagen = ImageDataGenerator(rescale=1./255)

# Flow training images in batches of 32 using train_datagen generator
train_generator = train_datagen.flow(X_train_resampled_reshaped,␣
 ↪y_train_resampled, batch_size=32)
# Flow validation images in batches of 32 using validation_datagen generator
validation_generator = validation_datagen.flow(X_val, y_val, batch_size=32)

# Model Architecture
model = Sequential()
```

```
model.add(Conv2D(64, 3, kernel_initializer='he_uniform', activation='relu',␣
 ↪input_shape=(150,150,3),kernel_regularizer=l2(0.001)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(3, 3), strides=2))

model.add(Conv2D(64, 3, kernel_initializer='he_uniform',␣
 ↪activation='relu',kernel_regularizer=l2(0.001)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(3, 3), strides=2))

model.add(Conv2D(128, 3, kernel_initializer='he_uniform',␣
 ↪activation='relu',kernel_regularizer=l2(0.001)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(3, 3), strides=2))

model.add(Flatten())

model.add(Dense(256, activation='relu',␣
 ↪kernel_initializer='he_uniform',kernel_regularizer=l2(0.001)))
model.add(Dropout(0.5))

model.add(Dense(128, activation='relu',␣
 ↪kernel_initializer='he_uniform',kernel_regularizer=l2(0.001)))
model.add(Dropout(0.5))

model.add(Dense(1, activation='sigmoid'))

# Compile model
model.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.0001),␣
 ↪metrics=['accuracy'])

# Early stopping to prevent overfitting
early_stop = EarlyStopping(monitor='val_loss', patience=5,␣
 ↪restore_best_weights=True)

# Training the model
history = model.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=30,
    validation_data=validation_generator,
    validation_steps=len(validation_generator),
    callbacks=[early_stop]
)
```

```
<IPython.core.display.Javascript object>
```

```
Epoch 1/30
84/84 [==============================] - 50s 580ms/step - loss: 3.4407 -
accuracy: 0.4940 - val_loss: 2.9792 - val_accuracy: 0.2293
Epoch 2/30
84/84 [==============================] - 48s 567ms/step - loss: 2.4440 -
accuracy: 0.5113 - val_loss: 3.2243 - val_accuracy: 0.2293
Epoch 3/30
84/84 [==============================] - 50s 596ms/step - loss: 2.1041 -
accuracy: 0.4947 - val_loss: 2.7453 - val_accuracy: 0.2293
Epoch 4/30
84/84 [==============================] - 48s 572ms/step - loss: 2.0285 -
accuracy: 0.5207 - val_loss: 1.9982 - val_accuracy: 0.3920
Epoch 5/30
84/84 [==============================] - 48s 565ms/step - loss: 1.9938 -
accuracy: 0.5252 - val_loss: 2.3253 - val_accuracy: 0.2427
Epoch 6/30
84/84 [==============================] - 49s 586ms/step - loss: 1.9828 -
accuracy: 0.5207 - val_loss: 2.0272 - val_accuracy: 0.4987
Epoch 7/30
84/84 [==============================] - 48s 575ms/step - loss: 1.9588 -
accuracy: 0.5038 - val_loss: 2.4187 - val_accuracy: 0.5067
Epoch 8/30
84/84 [==============================] - 48s 570ms/step - loss: 1.9426 -
accuracy: 0.5023 - val_loss: 1.9241 - val_accuracy: 0.5787
Epoch 9/30
84/84 [==============================] - 49s 584ms/step - loss: 1.9393 -
accuracy: 0.4996 - val_loss: 1.9085 - val_accuracy: 0.6293
Epoch 10/30
84/84 [==============================] - 48s 571ms/step - loss: 1.9171 -
accuracy: 0.4962 - val_loss: 1.8983 - val_accuracy: 0.7253
Epoch 11/30
84/84 [==============================] - 50s 597ms/step - loss: 1.9043 -
accuracy: 0.5038 - val_loss: 1.8848 - val_accuracy: 0.7467
Epoch 12/30
84/84 [==============================] - 54s 640ms/step - loss: 1.8940 -
accuracy: 0.5131 - val_loss: 1.8795 - val_accuracy: 0.7573
Epoch 13/30
84/84 [==============================] - 48s 569ms/step - loss: 1.8882 -
accuracy: 0.4955 - val_loss: 1.8644 - val_accuracy: 0.7573
Epoch 14/30
84/84 [==============================] - 48s 567ms/step - loss: 1.8776 -
accuracy: 0.5098 - val_loss: 1.8595 - val_accuracy: 0.7547
Epoch 15/30
84/84 [==============================] - 49s 579ms/step - loss: 1.8632 -
accuracy: 0.4962 - val_loss: 1.8466 - val_accuracy: 0.7707
Epoch 16/30
84/84 [==============================] - 49s 578ms/step - loss: 1.8516 -
accuracy: 0.4966 - val_loss: 1.8367 - val_accuracy: 0.7680
```

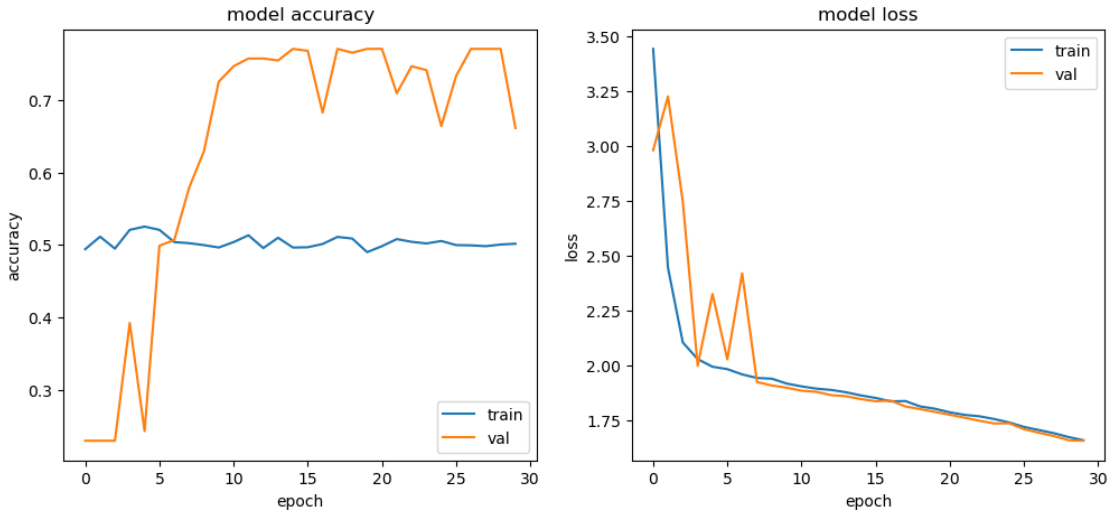
```
Epoch 17/30
84/84 [==============================] - 49s 575ms/step - loss: 1.8363 -
accuracy: 0.5011 - val_loss: 1.8397 - val_accuracy: 0.6827
Epoch 18/30
84/84 [==============================] - 50s 590ms/step - loss: 1.8375 -
accuracy: 0.5109 - val_loss: 1.8131 - val_accuracy: 0.7707
Epoch 19/30
84/84 [==============================] - 57s 681ms/step - loss: 1.8129 -
accuracy: 0.5086 - val_loss: 1.8008 - val_accuracy: 0.7653
Epoch 20/30
84/84 [==============================] - 54s 643ms/step - loss: 1.8025 -
accuracy: 0.4899 - val_loss: 1.7881 - val_accuracy: 0.7707
Epoch 21/30
84/84 [==============================] - 49s 585ms/step - loss: 1.7862 -
accuracy: 0.4981 - val_loss: 1.7749 - val_accuracy: 0.7707
Epoch 22/30
84/84 [==============================] - 54s 637ms/step - loss: 1.7747 -
accuracy: 0.5079 - val_loss: 1.7617 - val_accuracy: 0.7093
Epoch 23/30
84/84 [==============================] - 55s 657ms/step - loss: 1.7681 -
accuracy: 0.5041 - val_loss: 1.7477 - val_accuracy: 0.7467
Epoch 24/30
84/84 [==============================] - 52s 616ms/step - loss: 1.7556 -
accuracy: 0.5019 - val_loss: 1.7355 - val_accuracy: 0.7413
Epoch 25/30
84/84 [==============================] - 49s 579ms/step - loss: 1.7401 -
accuracy: 0.5053 - val_loss: 1.7363 - val_accuracy: 0.6640
Epoch 26/30
84/84 [==============================] - 50s 592ms/step - loss: 1.7196 -
accuracy: 0.4996 - val_loss: 1.7095 - val_accuracy: 0.7333
Epoch 27/30
84/84 [==============================] - 66s 790ms/step - loss: 1.7056 -
accuracy: 0.4992 - val_loss: 1.6931 - val_accuracy: 0.7707
Epoch 28/30
84/84 [==============================] - 52s 613ms/step - loss: 1.6913 -
accuracy: 0.4981 - val_loss: 1.6783 - val_accuracy: 0.7707
Epoch 29/30
84/84 [==============================] - 59s 703ms/step - loss: 1.6736 -
accuracy: 0.5004 - val_loss: 1.6592 - val_accuracy: 0.7707
Epoch 30/30
84/84 [==============================] - 51s 609ms/step - loss: 1.6593 -
accuracy: 0.5015 - val_loss: 1.6568 - val_accuracy: 0.6613
```

[113]:
```
plot_results(history)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

```
Maximum Loss : 3.4407

Minimum Loss : 1.6593

Loss difference : 1.7814
```



We can see model is performing better in terms of loss and acuracy, but still has the same problem on the training data and needs to be modified to achive the proper plots. I will learning scheduler that is another callback tool to find and train the model with the best learning rate, first i need to define a function to calculate the new learning rate and the pass it to the callback while fitting.

[137]:
```
# Define a learning rate schedule function
def lr_schedule(epoch, lr):
    """
    Returns a custom learning rate that decreases as epochs progress.
    """
    initial_lr = 0.0001
    decay = 0.9
    lr = initial_lr * math.pow(decay, epoch)
    return lr

# Clearing session
tf.keras.backend.clear_session()

# Data Augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
```

```
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')

validation_datagen = ImageDataGenerator(rescale=1./255)

# Flow training images in batches of 32 using train_datagen generator
train_generator = train_datagen.flow(X_train_resampled_reshaped,␣
↪y_train_resampled, batch_size=32)
# Flow validation images in batches of 32 using validation_datagen generator
validation_generator = validation_datagen.flow(X_val, y_val, batch_size=32)

# Model Architecture
model = Sequential()

model.add(Conv2D(64, 3, kernel_initializer='he_uniform', activation='relu',␣
↪input_shape=(150,150,3),kernel_regularizer=l2(0.001)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(3, 3), strides=2))

model.add(Conv2D(64, 3, kernel_initializer='he_uniform',␣
↪activation='relu',kernel_regularizer=l2(0.001)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(3, 3), strides=2))

model.add(Conv2D(128, 3, kernel_initializer='he_uniform',␣
↪activation='relu',kernel_regularizer=l2(0.001)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(3, 3), strides=2))

model.add(Flatten())

model.add(Dense(256, activation='relu',␣
↪kernel_initializer='he_uniform',kernel_regularizer=l2(0.001)))
model.add(Dropout(0.5))

model.add(Dense(128, activation='relu',␣
↪kernel_initializer='he_uniform',kernel_regularizer=l2(0.001)))
model.add(Dropout(0.5))

model.add(Dense(1, activation='sigmoid'))

# Compile model
model.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.0001),␣
↪metrics=['accuracy'])
```

```
# Early stopping to prevent overfitting
early_stop = EarlyStopping(monitor='val_loss', patience=5,␣
 ↪restore_best_weights=True)

# Learning rate scheduler
lr_scheduler = LearningRateScheduler(lr_schedule)

# Training the model with learning rate scheduler
history = model.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=20,
    validation_data=validation_generator,
    validation_steps=len(validation_generator),
    callbacks=[early_stop, lr_scheduler]
)
```

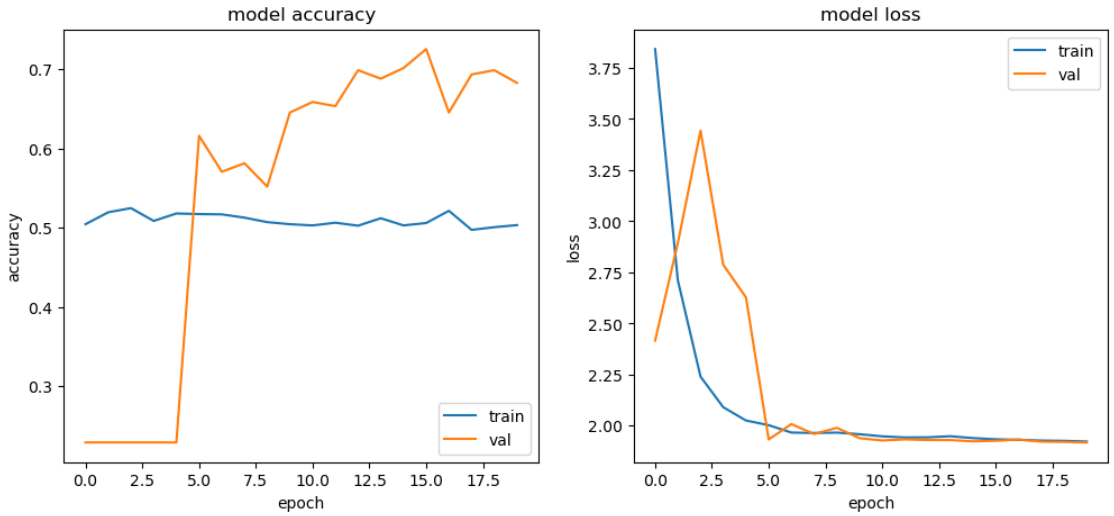
```
<IPython.core.display.Javascript object>
```

```
Epoch 1/20
84/84 [==============================] - 77s 852ms/step - loss: 3.8420 -
accuracy: 0.5045 - val_loss: 2.4166 - val_accuracy: 0.2293 - lr: 1.0000e-04
Epoch 2/20
84/84 [==============================] - 71s 846ms/step - loss: 2.7112 -
accuracy: 0.5195 - val_loss: 2.8934 - val_accuracy: 0.2293 - lr: 9.0000e-05
Epoch 3/20
84/84 [==============================] - 73s 869ms/step - loss: 2.2402 -
accuracy: 0.5248 - val_loss: 3.4431 - val_accuracy: 0.2293 - lr: 8.1000e-05
Epoch 4/20
84/84 [==============================] - 74s 877ms/step - loss: 2.0907 -
accuracy: 0.5086 - val_loss: 2.7870 - val_accuracy: 0.2293 - lr: 7.2900e-05
Epoch 5/20
84/84 [==============================] - 73s 860ms/step - loss: 2.0257 -
accuracy: 0.5180 - val_loss: 2.6278 - val_accuracy: 0.2293 - lr: 6.5610e-05
Epoch 6/20
84/84 [==============================] - 71s 840ms/step - loss: 2.0028 -
accuracy: 0.5173 - val_loss: 1.9330 - val_accuracy: 0.6160 - lr: 5.9049e-05
Epoch 7/20
84/84 [==============================] - 71s 841ms/step - loss: 1.9669 -
accuracy: 0.5169 - val_loss: 2.0082 - val_accuracy: 0.5707 - lr: 5.3144e-05
Epoch 8/20
84/84 [==============================] - 70s 829ms/step - loss: 1.9647 -
accuracy: 0.5128 - val_loss: 1.9595 - val_accuracy: 0.5813 - lr: 4.7830e-05
Epoch 9/20
84/84 [==============================] - 70s 831ms/step - loss: 1.9665 -
accuracy: 0.5071 - val_loss: 1.9900 - val_accuracy: 0.5520 - lr: 4.3047e-05
Epoch 10/20
```

```
84/84 [==============================] - 70s 827ms/step - loss: 1.9587 -
accuracy: 0.5045 - val_loss: 1.9393 - val_accuracy: 0.6453 - lr: 3.8742e-05
Epoch 11/20
84/84 [==============================] - 71s 836ms/step - loss: 1.9483 -
accuracy: 0.5030 - val_loss: 1.9282 - val_accuracy: 0.6587 - lr: 3.4868e-05
Epoch 12/20
84/84 [==============================] - 72s 852ms/step - loss: 1.9429 -
accuracy: 0.5064 - val_loss: 1.9334 - val_accuracy: 0.6533 - lr: 3.1381e-05
Epoch 13/20
84/84 [==============================] - 71s 840ms/step - loss: 1.9435 -
accuracy: 0.5026 - val_loss: 1.9303 - val_accuracy: 0.6987 - lr: 2.8243e-05
Epoch 14/20
84/84 [==============================] - 70s 829ms/step - loss: 1.9489 -
accuracy: 0.5120 - val_loss: 1.9301 - val_accuracy: 0.6880 - lr: 2.5419e-05
Epoch 15/20
84/84 [==============================] - 70s 834ms/step - loss: 1.9395 -
accuracy: 0.5030 - val_loss: 1.9237 - val_accuracy: 0.7013 - lr: 2.2877e-05
Epoch 16/20
84/84 [==============================] - 71s 836ms/step - loss: 1.9339 -
accuracy: 0.5060 - val_loss: 1.9266 - val_accuracy: 0.7253 - lr: 2.0589e-05
Epoch 17/20
84/84 [==============================] - 70s 834ms/step - loss: 1.9313 -
accuracy: 0.5214 - val_loss: 1.9329 - val_accuracy: 0.6453 - lr: 1.8530e-05
Epoch 18/20
84/84 [==============================] - 69s 820ms/step - loss: 1.9277 -
accuracy: 0.4974 - val_loss: 1.9223 - val_accuracy: 0.6933 - lr: 1.6677e-05
Epoch 19/20
84/84 [==============================] - 69s 819ms/step - loss: 1.9262 -
accuracy: 0.5008 - val_loss: 1.9213 - val_accuracy: 0.6987 - lr: 1.5009e-05
Epoch 20/20
84/84 [==============================] - 69s 817ms/step - loss: 1.9229 -
accuracy: 0.5034 - val_loss: 1.9184 - val_accuracy: 0.6827 - lr: 1.3509e-05
```

[138]:
```
plot_results(history)
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

Maximum Loss : 3.8420

Minimum Loss : 1.9229

Loss difference : 1.9191
```



Train accuracy is not increasing whatever change i make to the hyperparameters or the architecture, the only thing left is that i make the acrchitecture a bit more complex to be able to capture more information and learn better from the training data, so my last attempt is to add one other convolutional layer to the model to see if there is a positive change, and if not this is the last modification.

[146]:
```
# Clearing session
tf.keras.backend.clear_session()

# Data Augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')

validation_datagen = ImageDataGenerator(rescale=1./255)

# Flow training images in batches of 32 using train_datagen generator
train_generator = train_datagen.flow(X_train_resampled_reshaped,␣
 ↪y_train_resampled, batch_size=32)
# Flow validation images in batches of 32 using validation_datagen generator
validation_generator = validation_datagen.flow(X_val, y_val, batch_size=32)

# Model Architecture
model = Sequential()
```

```
model.add(Conv2D(64, 3, kernel_initializer='he_uniform', activation='relu',␣
 ↪input_shape=(150,150,3),kernel_regularizer=l2(0.001)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(3, 3), strides=2))

model.add(Conv2D(64, 3, kernel_initializer='he_uniform',␣
 ↪activation='relu',kernel_regularizer=l2(0.001)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(3, 3), strides=2))

model.add(Conv2D(128, 3, kernel_initializer='he_uniform',␣
 ↪activation='relu',kernel_regularizer=l2(0.001)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(3, 3), strides=2))

model.add(Conv2D(128, 3, kernel_initializer='he_uniform',␣
 ↪activation='relu',kernel_regularizer=l2(0.001)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(3, 3), strides=2))

model.add(Flatten())

model.add(Dense(256, activation='relu',␣
 ↪kernel_initializer='he_uniform',kernel_regularizer=l2(0.001)))
model.add(Dropout(0.5))

model.add(Dense(128, activation='relu',␣
 ↪kernel_initializer='he_uniform',kernel_regularizer=l2(0.001)))
model.add(Dropout(0.5))

model.add(Dense(1, activation='sigmoid'))

# Compile model
model.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.0001),␣
 ↪metrics=['accuracy'])

# Early stopping to prevent overfitting
early_stop = EarlyStopping(monitor='val_loss', patience=5,␣
 ↪restore_best_weights=True)

# Learning rate scheduler
lr_scheduler = LearningRateScheduler(lr_schedule)

# Training the model with learning rate scheduler
history = model.fit(
```

```
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=20,
    validation_data=validation_generator,
    validation_steps=len(validation_generator),
    callbacks=[early_stop, lr_scheduler]
)
```

```
<IPython.core.display.Javascript object>
```

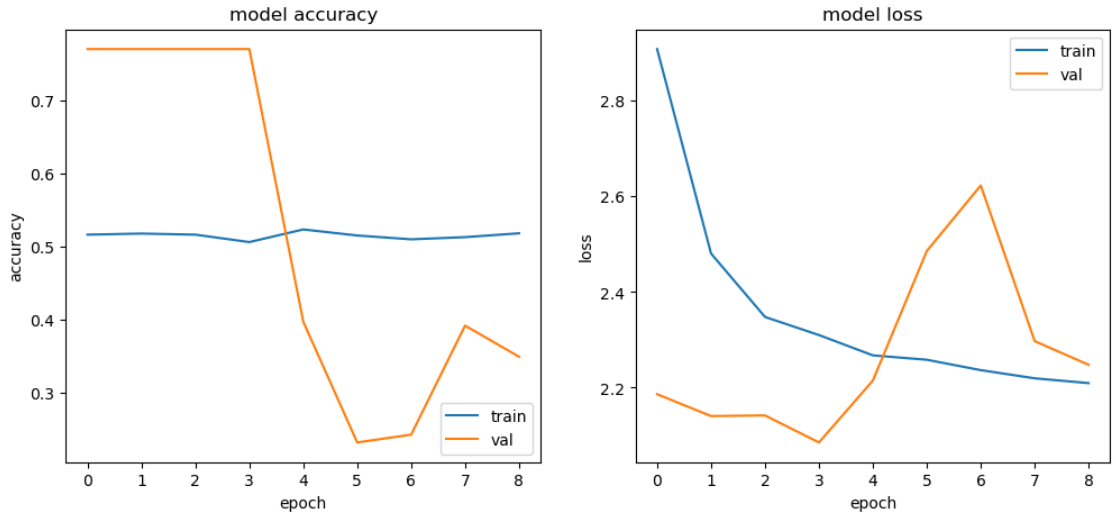
```
Epoch 1/20
84/84 [==============================] - 56s 631ms/step - loss: 2.9084 -
accuracy: 0.5165 - val_loss: 2.1861 - val_accuracy: 0.7707 - lr: 1.0000e-04
Epoch 2/20
84/84 [==============================] - 54s 636ms/step - loss: 2.4805 -
accuracy: 0.5180 - val_loss: 2.1404 - val_accuracy: 0.7707 - lr: 9.0000e-05
Epoch 3/20
84/84 [==============================] - 53s 625ms/step - loss: 2.3479 -
accuracy: 0.5165 - val_loss: 2.1418 - val_accuracy: 0.7707 - lr: 8.1000e-05
Epoch 4/20
84/84 [==============================] - 51s 607ms/step - loss: 2.3100 -
accuracy: 0.5064 - val_loss: 2.0853 - val_accuracy: 0.7707 - lr: 7.2900e-05
Epoch 5/20
84/84 [==============================] - 50s 599ms/step - loss: 2.2675 -
accuracy: 0.5237 - val_loss: 2.2148 - val_accuracy: 0.3973 - lr: 6.5610e-05
Epoch 6/20
84/84 [==============================] - 50s 586ms/step - loss: 2.2583 -
accuracy: 0.5154 - val_loss: 2.4859 - val_accuracy: 0.2320 - lr: 5.9049e-05
Epoch 7/20
84/84 [==============================] - 49s 579ms/step - loss: 2.2366 -
accuracy: 0.5101 - val_loss: 2.6230 - val_accuracy: 0.2427 - lr: 5.3144e-05
Epoch 8/20
84/84 [==============================] - 50s 590ms/step - loss: 2.2195 -
accuracy: 0.5131 - val_loss: 2.2973 - val_accuracy: 0.3920 - lr: 4.7830e-05
Epoch 9/20
84/84 [==============================] - 48s 574ms/step - loss: 2.2094 -
accuracy: 0.5184 - val_loss: 2.2477 - val_accuracy: 0.3493 - lr: 4.3047e-05
```

[147]:
```
plot_results(history)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

```
Maximum Loss : 2.9084

Minimum Loss : 2.2094

Loss difference : 0.6990
```



It was the last modification for this project, there are many possible reasons for this issue, from the architecture and hyper parametes to the quality of data and the pictures that i created. It is possible that if we make some changes to the images we can get better results. Even the other two categoreis that i crated could be a potent high quality data if i we had the time to try different networks on it and monitor the results.

6.7 Future work

In this study, I generated 2500 images derived from datasets containing users' mouse cursor movement data. These images were categorized into three distinct groups, each undergoing several modifications to enhance classification accuracy. The primary objective was to develop a Convolutional Neural Network (CNN) model for image classification to predict user interaction with advertisements based on cursor behavior. However, initial results fell short of expectations, indicating the need for further investigation and refinement. Moving forward, the proposed work entails exploring various avenues for improvement. This includes creating additional synthetic images to diversify the dataset, employing interpretability techniques such as Local Interpretable Model-Agnostic Explanations (LIME) and Shapley Additive Explanations (SHAP) to discern influential image features, and iteratively refining both the dataset and model architecture to achieve better performance. By iterating on both the dataset and model design, this research aims to enhance the accuracy and efficacy of the CNN-based classification system for predicting user interactions with online advertisements.(ChatGPT)

7 ENSEMBLE LEARNING

Ensemble learning is a generic meta-approach to machine learning that aims to improve predictive performance by merging the predictions from several models.

While the number of ensembles we may create for our predictive modeling challenge seems limitless, the discipline of ensemble learning is dominated by three techniques. That it is a topic of study that has given rise to many more specific approaches rather than algorithms per se. `Bagging, stacking`,

and `boosting` are the three primary classes of ensemble learning techniques.

- Boosting involves adding ensemble members sequentially that correct the predictions made by prior models and outputs a weighted average of the predictions.
- Bagging involves fitting many decision trees on different samples of the same dataset and averaging the predictions.
- Stacking involves fitting many different models types on the same data and using another model to learn how to best combine the predictions. (Brownlee, 2021)

[ ]:
```
# Define base estimators
base_estimators = [
    ('SVM linear', svm_linear),
    ('SVM polynomial', svm_classifier),
    ('Decision tree', dec_tree),
    ('Random Forest', rf)
]
```

7.1 Boosting classifier

[ ]:
```
adaboost_classifier = AdaBoostClassifier(base_estimator=None, n_estimators=50,␣
↪random_state=42)
adaboost_classifier.fit(X_train, y_train)
y_pred_adaboost = adaboost_classifier.predict(X_validation)
```

[ ]:
```
accuracy_adaboost = accuracy_score(y_validation, y_pred_adaboost)
precision_adaboost = precision_score(y_validation, y_pred_adaboost)
recall_adaboost = recall_score(y_validation, y_pred_adaboost)
f1_adaboost = f1_score(y_validation, y_pred_adaboost)
```

[ ]:
```
scores_dict["AdaBoost"] = [
    "{:.4f}".format(accuracy_adaboost),
    "{:.4f}".format(precision_adaboost),
    "{:.4f}".format(recall_adaboost),
    "{:.4f}".format(f1_adaboost)
]
```

7.2 Bagging classifier

[ ]:
```
bagging_classifier = BaggingClassifier(base_estimator=DecisionTreeClassifier(),␣
↪n_estimators=10, random_state=42)
bagging_classifier.fit(X_train, y_train)
y_pred_bagging = bagging_classifier.predict(X_validation)
```

[ ]:
```
accuracy_bagging = accuracy_score(y_validation, y_pred_bagging)
precision_bagging = precision_score(y_validation, y_pred_bagging)
recall_bagging = recall_score(y_validation, y_pred_bagging)
f1_bagging = f1_score(y_validation, y_pred_bagging)
```

[ ]:
```
scores_dict["Bagging"] = [
    "{:.4f}".format(accuracy_bagging),
    "{:.4f}".format(precision_bagging),
    "{:.4f}".format(recall_bagging),
    "{:.4f}".format(f1_bagging)
]
```

7.3 Stacking classifier

[ ]:
```
meta_learner = LogisticRegression()
stacking_classifier = StackingClassifier(estimators=base_estimators,␣
↪final_estimator=meta_learner)
stacking_classifier.fit(X_train, y_train)
y_pred_stacking = stacking_classifier.predict(X_validation)
```

[ ]:
```
accuracy_stacking = accuracy_score(y_validation, y_pred_stacking)
precision_stacking = precision_score(y_validation, y_pred_stacking)
recall_stacking = recall_score(y_validation, y_pred_stacking)
f1_stacking = f1_score(y_validation, y_pred_stacking)
```

[ ]:
```
scores_dict["Stacking"] = [
    "{:.4f}".format(accuracy_stacking),
    "{:.4f}".format(precision_stacking),
    "{:.4f}".format(recall_stacking),
    "{:.4f}".format(f1_stacking)
]
```

[130]:
```
scores_df = pd.DataFrame(scores_dict,index=['Precision', 'Recall',␣
↪'F1','Accuracy'])
# print(scores_df.to_latex())
```

```
<IPython.core.display.Javascript object>
```

	SVM Linear	SVM Polynomial	Decision Tree	Random Forest	Bagging	Stacking	AdaBoost
Precision	0.7774	0.7749	0.7686	0.7887	0.7520	0.7947	0.7680
Recall	0.6853	0.7093	0.7360	0.8053	0.4462	0.5652	0.4872
F1	0.7092	0.7289	0.7481	0.7916	0.3372	0.4535	0.2209
Accuracy	0.6853	0.7093	0.7481	0.7916	0.3841	0.5032	0.3040

It looks like the ensemble methods didn't really enhance the performance much compared to using classifiers. Here are some key points based on the scores provided:

- The individual classifiers, Random Forest achieved good precision, recall, F1 score and accuracy when compared to the ensemble methods.
- The AdaBoost Classifier exhibited a decrease in precision, recall, F1 score and accuracy when compared to the classifiers.
- The Bagging Classifier also displayed a decrease in precision, recall, F1 score and accuracy when compared to the classifiers.

- As for the Stacking Classifier it had results. Higher precision but lower recall, F1 score and accuracy compared to using individual classifiers.

Based on these findings it seems like using methods didn't yield improvements over using individual classifiers for this specific task and dataset. It's crucial to take into account the data characteristics, the variety of base classifiers used and the parameters of methods when exploring learning techniques. Further analysis and experimentation might be necessary to determine the approach, for our particular issue.

8 RESOURCES

Mohammad (2023). mo-alrz/Classification-Project. [online] GitHub. Available at: https://github.com/mo-alrz/Classification-Project [Accessed 16 May 2024].

---

www.youtube.com. (n.d.). Create Latex table from pandas DataFrame in Python. [online] Available at: https://www.youtube.com/watch?v=gLalZyodYqs&ab_channel=ExplainHowToSimply [Accessed 19 Apr. 2024].

L.A. and Vivó, R. (2023). *evtrack*. [online] GitHub. Available at: https://github.com/luileito/evtrack. [Accessed 14 Apr. 2024].

Luis A. Leiva, Ioannis Arapakis. (2020) The Attentive Cursor Dataset [online]. Front. Hum. Neurosci. 14. Available At:https://gitlab.com/iarapakis/the-attentive-cursor-dataset. [Accessed 14 Apr. 2024].

Verma, Y. (2022). How to detect and treat outliers in categorical data? [online] Analytics India Magazine. Available at: https://analyticsindiamag.com/how-to-detect-and-treat-outliers-in-categorical-data/.

Dean, Brian. "What Are SERPs and Why Are They Important for SEO? [Here's the Answer]." Backlinko, 16 Nov. 2023, backlinko.com/hub/seo/serps.

---

www.javatpoint.com. (n.d.). Precision and Recall in Machine Learning - Javatpoint. [online] Available at: https://www.javatpoint.com/precision-and-recall-in-machine-learning.

Sharma, N. (2023). Understanding and Applying F1 Score: A Deep Dive with Hands-On Coding. [online] Arize AI. Available at: https://arize.com/blog-course/f1-score/.

Wikipedia Contributors (2019). Euclidean distance. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Euclidean_distance.

Chat GPT, 2024, https://chat.openai.com/share/e33b111c-d8a9-4c0b-9f67-a315a8048b31

Ioannis Arapakis, Luis A. Leiva. (2020). Learning Efficient Representations of Mouse Movements to Predict User Attention. Proc. Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval (SIGIR). Available at: https://arxiv.org/abs/2006.01644 [Accessed 21 Apr. 2024].

Chat GPT, 2024, https://chat.openai.com/share/776bf365-0bfc-41b1-a4c1-e2c2ea9123fc

guest_blog (2020). Imbalanced Classification | Handling Imbalanced Data using Python. [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2020/07/10-techniques-to-deal-with-class-imbalance-in-machine-learning/.

Takimoglu, A. (2021). What is Data Augmentation? Techniques, Benefit and Examples. [online] research.aimultiple.com. Available at: https://research.aimultiple.com/data-augmentation/.

Chat GPT, 2024, https://chat.openai.com/share/b9157613-0158-4c56-a33a-730e5bfa6b62

Brownlee, J. (2021). A Gentle Introduction to Ensemble Learning Algorithms. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/tour-of-ensemble-learning-algorithms/.

9 LETTER OF APPRECIATION

I would like to express my sincere appreciation to Professors Zsofia Gyarmathy and Adam Franyo for their invaluable guidance, support, and mentorship throughout the completion of this thesis.

Their profound understanding of the subject matter and their ability to simplify complex concepts have been instrumental in expanding my knowledge horizons. Their encouragement to explore innovative approaches and their constructive feedback have enriched my academic perspective.

Together, their collective expertise and mentorship have not only guided me through the challenges of thesis writing but have also empowered me to grow both academically and personally. I am deeply grateful for the time, effort, and wisdom they have generously shared with me.

In conclusion, I extend my heartfelt gratitude to both Professors for their unwavering support, encouragement, and guidance. Their belief in my potential has been a constant source of motivation, propelling me to strive for excellence.

Mohammad Alirezaee Spring 2024 IBS Univeristy

10 WORD COUNT

Im counting the words of my document with this code, it counts only the markdown cells content, so the outputs of my codes and models are not included.

[155]:
```
# Convert the notebook to a script
notebook_file = r"C:\Users\malir\CAPB351_24S.ipynb"
script, _ = PythonExporter().from_filename(notebook_file)

# Count the words in the script
word_count = len(script.split())
print("Word count:", word_count)
```

```
Word count: 13260
```

[ ]:
```

```