A.YILDIZ

BAU 2022

MOBILE APPLICATION CLASSIFICATION USING MACHINE LEARNING

Anıl YILDIZ

MARCH 2022

MOBILE APPLICATION CLASSIFICATION USING MACHINE LEARNING

A THESIS SUBMITTED TO THE

GRADUATE SCHOOL

OF

BAHCESEHIR UNIVERSITY

BY

ANIL YILDIZ

IN PARTIAL FULLFILMENT OF THE REQUIREMENTS

FOR

THE DEGREE OF MASTER OF COMPUTEER ENGINERING

IN THE DEPARTMENT OF APPLIED SCIENCES

MARCH 2022

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as requiredby these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname : Anıl Yıldız

Signature :

ABSTRACT

MOBILE APPLICATION CLASSIFICATION USING MACHINE LEARNING

Yıldız, Anıl

Master's Program in Computer Engineering

Supervisor: Asst. Prof. Dr. Ece GELAL SOYAK

Co-Advisor: Asst. Prof. Cemal Okan ŞAKAR

March 2022, 45 pages

The technique of acquiring and analyzing individual data packets that go across your network is known as packet capture (PCAP) analysis. PCAP files are particularly useful because they may capture multilayer traffic data, recording packets from the data connection layer to the application layer. Network traffic analysis is a critical tool for monitoring network availability and activity in order to spot abnormalities, improve performance, and detect threats.

Although traffic capture is almost as old as internet technology, combining the captured data with machine learning technology is a fairly new technology. In the literature, packet data and machine learning technology have been combined and studies have been made in many areas such as traffic and category forecasting.

However, the category estimation was not made through the packages created by the applications used on the smartphones. In this thesis we are using a subset of the MIRAGE-2019 (Giuseppe Aceto, Domenico Ciuonzo et al. 2019) dataset, category prediction was made using three machine learning algorithms, namely Random Forest, LightGBM, Catboost, and a deep learning algorithm, Tabnet. The performances of these four algorithms were analyzed in detail and compared with each other on a overall and class-wise basis.

Keywords: Machine Learning, Packet Capture, Smart Phone, Multiclass Classification

ÖZ

MAKİNE ÖĞRENİMİ TABANLI AKILLI TELEFON UYGULAMA KATEGORİ TAHMİNİ

Yıldız, Anıl

Bilgisayar Mühendisliği Yüksek Lisans Programı

Tez Danışmanı: Dr. Öğr. Üyesi Ece GELAL SOYAK

Tez Danışmanı: Doç. Dr. Cemal Okan ŞAKAR

Mart 2022, 45 sayfa

Bir ağda bulunan veri paketlerini toplama ve analiz etme tekniği, paket yakalama (PCAP) analizi olarak bilinir. PCAP dosyaları, veri bağlantı katmanından uygulama katmanına kadar paketleri kaydederek çok katmanlı trafik verilerini yakalayabilmeleri sayesinde oldukça kullanışlıdır. Ağ trafiği analizi, anormallikleri tespit etmek, performansı artırmak ve tehditleri tespit etmek için ağ kullanılabilirliğini ve etkinliğini izlemek için kritik bir araçtır.

Trafik yakalama neredeyse internet teknolojisi kadar eski olmasına rağmen, yakalanan verileri makine öğrenimi teknolojisiyle birleştirmek oldukça yeni bir teknolojidir. Literatürde paket verisi ve makine öğrenmesi teknolojisi birleştirilerek trafik ve kategori tahmini gibi birçok alanda çalışmalar yapılmıştır.

Ancak akıllı telefonlarda kullanılan uygulamaların oluşturduğu paketler üzerinden kategori tahmini konusunda bir çalışma bulunmamaktadır. MIRAGE-2019 (Giuseppe Aceto ve diğerleri 2019) veri setinin bir alt kümesini kullandığımız bu tezde, üç makine öğrenme algoritması olan Random Forest, LightGBM, Catboost ve bir derin öğrenme algoritması olan Tabnet olmak üzere toplamda dört algoritma kullanılarak kategori tahmini yapılmıştır. Bu dört algoritmanın performansları ayrıntılı olarak analiz edilmiş, genel ve sınıf bazında birbirleriyle karşılaştırılmıştır.

Anahtar Kelimeler: Makine Öğrenimi, Paket Yakalama, Akıllı Telefon, Çok Sınıflı Sınıflandırma

To My Parents

ACKNOWLEDGMENTS

First and foremost, I would like to express my gratitude to my thesis advisor, Asst. Prof. Ece GELAL SOYAK and my thesis co-adviser, Assoc. Prof. Cemal Okan ŞAKAR, for allowing me to work on this thesis. I am quite appreciative for their assistance, insight, and crucial assistance in the production of my thesis.

I'd also like to thank the BAU Department of Computer Engineering, and my lecturers for all of their help over the years.

Finally, I'd want to thank my family: my parents, Selma and Erdoğan, and my sister Ezgi for their spiritual support throughout the composition of this thesis and my life in general. They are my source of inspiration and motivation. Thank you for making all of these sacrifices and always believing in me.

I'd also like to express my gratitude to everyone who has contributed, directly or indirectly, to this endeavor.

İstanbul, 2022 Anıl YILDIZ

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

CSV	Comma Separated Values
ML	Machine Learning
PCAP	Packet Capture Analysis
GBM	Gradient Boosting
IAT	Inter Arrival Time
LightGBM	Light Gradient Boosting

1. INTRODUCTION

In the past, there was little activity in the networks, but with the development of technology, not only computers but also many devices from phones to household appliances became able to connect to the Internet and produce network packages. As the density in the network traffic has increased, the control of the network by humans has become quite difficult or even impossible without various applications and methods. Therefore, machine and deep learning techniques have started to be applied in the analysis of packets generated in network traffic (E. Biersack, 2013).

In the last two decades, smartphones have become an important medium of communication, connecting mobile users to each other and to a variety of services. From the perspective of the network operator, it may be desirable to provide Quality of Service (QoS) to the end user based on the application types running on their smartphones. In a Wireless Local Area Network (WLAN), it may be desirable to provide context-aware service to connected devices; for example, application-dependent power savings may be applied. For security, it may be desirable to identify normal and abnormal traffic running on the smartphone. Due to these motivations, in this thesis, the traffic packets produced by smart phones have been analyzed and the category of the application has been estimated.

The MIRAGE-2019 (Giuseppe Aceto, Domenico Ciuonzo et al. 2019) dataset was selected for the study. This dataset has been generated by human users under a controlled setup (in a lab environment). The data set consists of per-packet data, per-flow data, and per-flow metadata. We have first formatted the data set and prepared for processing. After preprocessing, one of the most common machine learning algorithms Random Forest, two of the gradient boosting algorithms LightGBM and Catboost, and the deep learning algorithm Tabnet were used sequentially, in order to estimate the traffic type from captured traffic flow characteristics. The analysis shows that while some applications are easily categorized (e.g. entertainment applications such as Spotify or Duolingo), some others have packet flow characteristics that are similar to each other and therefore it is harder to accurately predict these types (e.g. gaming, productivity, lifestyle applications).

Among the algorithms we have used, the highest accuracy was achieved using the Random Forest algorithm. The accuracies with LightGBM, Catboost and Tabnet respectively followed.

The rest of the thesis is structured as follows. In Section II, we discuss the relevant studies and provide state of the art. In Section III, we provide background on the algorithms used in our study. In Section IV, we describe the data set and the preprocessing performed to prepare the data set for analysis. In Section V we demonstrate our results and interpret what they mean for the network traffic flows. Finally, in Section VI we conclude the thesis.

2. RELATED WORK

With advancements in machine learning algorithms, data collection and processing from network traffic has become a popular tool in order to identify flow types, anomalies, security breaches. In this section, we present a brief summary of various previous machine or/and deep learning studies that process collected network traffic.

About one and a half decades ago, the research community began to investigate new traffic characterization and classification mechanisms, that would surpass the limitations with traditional approaches based on IP address, port number, and payload analysis (M. Mellia, 2009). T.T. Nguyen and G. Armitage, 2008 reviewed 18 significant works studying the application of ML techniques to IP traffic classification. This study motivated a shift away from the port-based or payload-based traffic classification. It also underlined that each ML algorithm may perform differently for different Internet applications. In A. Dainotti et al., 2010, the authors proposed to use the sign pattern and payload size of the first four packets in each flow, and to incorporate decision trees, to identify traffic flows.

A. A. Al-Subaihin et al., 2016 developed a method for determining app similarity based on the indicated metadata on the app store. They have crawled and collected the metadata of free and paid apps from BlackBerry and Google Play stores. A total of 14,258 apps from 16 different categories were collected from the BlackBerry store, and 3,619 apps from 23 high-level categories from the Google Play store. Information retrieval and ontological analysis were used to extract features, which were then utilized as characteristics to characterize apps.

G. He et al., 2017 developed a methodology for detecting encrypted network traffic in mobile apps using traffic correlation. Motivated by the challenge to classify apps from encrypted network traffic, they have utilized temporal and lexical similarity to cluster correlated DNS requests.

Y. Liu et al., 2018 proposed CFMTC (Cascade Forest for Mobile Traces Categorization), a traffic classification system for mobile networks that uses flow statistical data acquired from mobile traces. CFMTC uses deep learning approaches

and statistical aspects of bidirectional flows of mobile traces, for effective mobile application categorization. Cascade Forest algorithm is trained to classify raw mobile traces. CFMTC is evaluated on mobile network traces generated by Kuwo Music, WeChat, PPTV Live traces. The results demonstrate an average accuracy of about 88.71%.

V. F. Taylor et al., 2018 performed fingerprinting to identify smartphone apps. They have also investigated how app fingerprints change over time, across devices, and across different versions of apps. They have implemented their approach to fingerprint 110 of the most popular apps in the Google Play Store, and they were able to identify them six months later with up to 96% accuracy. One important contribution of this paper is that they have introduced strategies to enable their app classification system to identify and mitigate the effect of ambiguous traffic, i.e., traffic in common among apps, such as advertisement traffic.

Using a multi-classification technique, G. Aceto et al., 2017 improved the classification performance of mobile app classifiers by intelligently integrating judgments from state-of-the-art classifiers offered for mobile and encrypted traffic classification. The proposed method uses five well-known combining approaches, as well as a pool of seven state-of-the-art classifiers that are particular to or suited for mobile traffic. Proposed approach has been evaluated on a dataset describing traffic from 49 apps in Android and 45 apps in iOS devices, and their results demonstrate that classification performance can be improved by up to 8.1% F-measure score with respect to the best base classifier.

The analysis of traffic flows are also used for abnormal traffic detection. In Y. Zhang et al., 2019, the statistical features of a flow are extracted and processed using LeNet-5 CNN and LSTM models in order to distinguish anomalies.

3. BACKGROUND

This section explains how TCP/IP packets are created (conversion of the data contained in these packages into our dataset is explained under the Methodology section) and classification algorithms used in this thesis. The research question that we investigate is a classification problem, and we investigate the performance of various classification algorithms. Before we get into the classification algorithms, let's go over why the research question is a classification problem in the first place.

Classification is a data analysis task in which an attempt is made to create a model for describing and distinguishing data classes and concepts. It is about identifying a set of categories that will serve as the foundation for a data training set. An appcategory attribute is included in subset of the MIRAGE 2019 data set, allowing the problem analysis to be based on a classification model in order to detect other attributes that affect the application category data.

Random Forest Classifier, Light GBM Classifier, Catboost Classifier, and Tabnet are utilized as feature selection and classification methods, and each is detailed in detail below. The aforementioned machine learning algorithms were used with Python on Jupyter.

3.1 TCP/IP PACKET CREATION

A network packet is a tiny piece of data that is transmitted through TCP/IP networks. Ethernet packets are typically 1.5 kilobytes in size. A packet is a unit of data that is routed between an origin and a destination on the internet or another packet-switched network – or networks that transport data in tiny packets.

When any packet, such as an email message, a web site in Hypertext Markup Language, an image in Graphics Interchange Format, or a Uniform Resource Locator request, is transferred over the Internet, it is split down into little pieces. For effective routing over the IP layer, the TCP layer of the TCP/IP protocol suite splits the file into bytes. An IP packet typically contains a maximum of 1,500 bytes of data.

Each packet is uniquely numbered and carries the destination's IP address. Individual packets for a particular file may travel over distinct paths between the source and destination devices over the Internet. When the packets arrive at their destination, the TCP layer at the receiving end reassembles them into the original file. Depending on the network, packets may also be referred to as a block, cell, frame, or segment. (Andrew Zola, 2021)

Network packets are made up of three parts: the header, the payload, and the trailer. All IP packets contain IP headers, which are necessary for the routers in the Internet core to process the packets in the most effective way. The IP header includes information about the data packet. According to RFC: 791 Internet Protocol (1981), the IP header includes:

- Checksum, which detects errors
- 16-bit identification number
- Flags to let a router know if it can fragment a packet
- Fragmentation offsets, which reconstruct fragmented packets
- Destination address
- Number of hops a packet can make
- IP
- Length of the packet -- but not always, as some networks have fixed-length packets
- Size of the header and payload
- Time-to-Live
- Originating address
- Packet number, in relation to the packet sequence
- Protocol or what type of packet is transmitted
- Synchronization or the few bits that enable the packet to match up to the network.

The data included within the packet is referred to as the payload. This is the basic data that the packet sends to the destination. To accommodate a fixed-length packet, the payload is frequently padded with blank data.

Trailers, sometimes known as the footer, are bits that indicate the conclusion of a package. These bits notify the receiving device that the packet has come to a conclusion. A form of error checking procedure may also be included in trailers.

3.2 CLASSIFICATION ALGORITHMS

3.2.1 RANDOM FOREST CLASSIFIER

As the name implies, a random forest is made up of a huge number of individual decision trees that work together as an ensemble. The random forest's trees each spat out a class forecast (Pedregosa et al., 2011).

The random forest algorithm can be summarized as follows:

- Create ntree samples from the original data.
- Create an unpruned classification or regression tree from each sample. At each node, a random sample of the predictors is taken and the optimal split from these variables is chosen.
- Aggregate the predictions of the ntree trees to predict fresh data.

The advantages of the Random Forest classifier that are relevant for our work are:

- Random Forests are effective with both categorical and numerical data. In most cases, no scaling or alteration of variables is required.
- Random Forests execute feature selection implicitly and build uncorrelated decision trees. It accomplishes this by selecting a random collection of characteristics to construct each decision tree. This also makes it an excellent model for dealing with a large number of characteristics in the data.
- Random Forests are not significantly impacted by outliers. This is accomplished by binning the variables.
- Random Forests are capable of handling both linear and non-linear connections.
- Random Forests, in general, give excellent accuracy and a good balancing of the bias-variance trade-off. Because the model's concept is to average the outcomes over the various decision trees it constructs, it also averages the variance.

The shortcomings of the Random Forest classifier are:

- Random Forests are difficult to understand. They give feature importance, but not as much visibility into the coefficients as linear regression provides.
- Random Forests may be computationally demanding when dealing with huge datasets.

For the study in this thesis, the hyper parameters of the Random Forest classifier have been optimized using Grid Search. Specifically, *max_depth* has been set to 32 and *n_estimators* has been set to 512. For other hyper parameters, the default values have been used.

Table 3.1

Random Forest Classifier Parameters

Name	**Definition**	**Value used in experiments**
n_estimators	The number of trees in the forest.	Changed to "512"
max_depth	The maximum depth of the tree.	Changed to "32"
criterion	The function to measure the quality of a split.	Default "gini"
min_samples_split	The minimum number of samples required to split an internal node.	Default "2"
min_samples_leaf	The minimum number of samples required to be at a leaf node.	Default "1"
min_weight_fraction_leaf	The minimum weighted fraction of the sum of weights required to be at a leaf node.	Default "0.0"
max_features	The number of features to consider when looking for the best split.	Default "auto"
max_leaf_nodes	Grow trees with max_leaf_nodes in best-first fashion.	Default "none"
min_impurity_decrease	A node will be split if this split induces a decrease of the impurity greater than or equal to this value.	(Default "0.0")
bootstrap	Whether bootstrap samples are used when building trees.	Default "true"
oob_score	Whether to use out-of-bag samples to estimate the generalization score.	Default "false"

n_jobs	The number of jobs to run in parallel.	Default “none”
random_state	Controls the randomness of the bootstrapping of the samples used when building trees and the sampling of the features to consider when looking for the best split at each node.	Default “none”
verbose	Controls the verbosity when fitting and predicting.	Default “0”
warm_start	When set to True, reuse the solution of the previous call to fit and add more estimators to the ensemble, otherwise, just fit a whole new forest.	Default “false”
class_weight	Weights associated with classes in the form.	Default “none”
ccp_alpha	Complexity parameter used for Minimal Cost-Complexity Pruning.	Default “0.0”
max_samples	If bootstrap is True, the number of samples to draw from X to train each base estimator.	Default “none”

3.2.2 LIGHTGBM CLASSIFIER

LightGBM is a gradient boosting framework based on decision trees that improves model efficiency while reducing memory usage.

LightGBM employs two innovative techniques: Gradient-based One Side Sampling and Exclusive Feature Bundling (EFB), which overcome the restrictions of the histogram-based approach employed in all GBDT (Gradient Boosting Decision Tree) frameworks. The properties of the LightGBM Algorithm are formed by the two methodologies of GOSS and EFB explained below. They work together to make the model work efficiently and to provide it a competitive advantage over alternative GBDT frameworks (LightGBM Documentation, 2021).

Gradient-based One Side Sampling Technique for LightGBM:

Different data instances play different roles in calculating information gain. Instances having higher gradients (i.e., under-trained instances) will contribute more to knowledge gain. GOSS preserves examples with big gradients (e.g., greater than a predetermined threshold or in the top percentiles) and drops instances with tiny gradients at random to maintain the accuracy of information gain estimation. This method can result in more accurate gain estimation than uniformly random sampling

with the same desired sample rate, particularly when the amount of information gain has a wide range.

Exclusive Feature Bundling Technique for LightGBM:

Because high-dimensional data is typically sparse, we can build a nearly lossless strategy to reduce the number of features. In particular, in a sparse feature space, many features are mutually exclusive, i.e., they never take nonzero values at the same time. The distinctive features can be securely combined into a single feature. As a result, the complexity of histogram construction shifts from O(data^feature) to O(data ^bundle), with bundle<<feature. As a result, the pace of the training framework is increased without sacrificing precision (Singh, 2021).

The advantages of the LightGBM classifier that are relevant for our work are:

- LightGBM employs a histogram-based technique, which buckets continuous feature data into discrete bins to speed up the training phase. As a result, faster training and improved efficiency are achieved.
- LightGBM replaces continuous data with discrete bins, resulting in less memory use. As a result, lower memory utilization is achieved.
- LightGBM builds far more complicated trees by using a leaf-wise split technique rather than a level-wise approach, which is the major element in obtaining better accuracy. As a result, it achieves higher accuracy than other boosting techniques.
- LightGBM is capable of doing similarly well with huge datasets while significantly reducing training time.

The shortcomings of the LightGBM classifier are:

- Because it creates more complicated trees, LightGBM splits the tree leaf-wise, which might lead to overfitting.
- Because LightGBM is vulnerable to overfitting, it can readily overfit tiny data sets (Subham Surana, 2020).

For the study in this thesis, the hyper parameters of the LightGBM classifier have been optimized using Grid Search. Specifically, *num_leaves* was set to 100,

min_child_samples was set to 15, *max_depth* was set to 20, *learning_rate* was 0.2 and *reg_alpha* was set to 0.03. Other hyper parameters have been used as LightGBM's library default parameters.

Table 3.2

LightGBM Classifier Parameters

Name	**Definition**	**Value**
num_leaves	Maximum tree leaves for base learners.	Changed to "100"
max_depth	Maximum tree depth for base learners.	Changed to "20"
min_child_samples	Minimum number of data needed in a child.	Changed to "15"
learning_rate	Boosting learning rate. This parameter can use callbacks parameter of fit method to shrink/adapt learning rate in training using reset_parameter callback. Note, that this will ignore the learning_rate argument in training.	Changed to "0.2"
reg_alpha	L1 regularization term on weights.	Changed to "0.03"
boosting_type	gbdt', traditional Gradient Boosting Decision Tree. 'dart', Dropouts meet Multiple Additive Regression Trees. 'goss', Gradient-based One-Side Sampling. 'rf', Random Forest.	Default "gbdt"
n_estimators	Number of boosted trees to fit.	Default "100"
subsample_for_bin	Number of samples for constructing bins.	Default "200000"
objective	Specify the learning task and the corresponding learning objective or a custom objective function to be used.	Default "None"
class_weight	Weights associated with classes in the form.	Default "None"
min_split_gain	Minimum loss reduction required to make a further partition on a leaf node of the tree	Default "0"
min_child_weight	Minimum sum of instance weight needed in a child.	Default "1e-3"
subsample	Subsample ratio of the training instance.	Default "1"

subsample_freq	Frequency of subsample, <=0 means no enable.	Default “0”
colsample_bytree	Subsample ratio of columns when constructing each tree.	Default “1”
reg_lambda	L2 regularization term on weights.	Default “0”
random_state	Random number seed.	Default “None”
n_jobs	Number of parallel threads to use for training.	Default “-1”
importance_type	The type of feature importance to be filled into feature_importances_.	Default “split”

3.2.3 CATBOOST CLASSIFIER

CatBoost is a recently open-sourced machine learning algorithm from Yandex. It integrates easily with deep learning frameworks such as Google's TensorFlow and Apple's Core ML. It can operate with a variety of data types and has the best-in-class accuracy (Tal Peretz, 2018)

CatBoost is particularly effective, as it yields high accuracy without requiring the rigorous data training that other machine learning approaches require. Second, it delivers excellent support for the more descriptive data formats.

The name "CatBoost" is derived from the phrases "Category" and "Boosting." As this library is based on the gradient boosting library, the name "Boost" is derived from the gradient boosting machine learning algorithm. Gradient boosting is a sophisticated machine learning method that is frequently used to solve a variety of business problems such as fraud detection, recommendation items, and forecasting. It also works well. It can also produce extremely good results with relatively little data, in contrast to DL models, which must learn from enormous amounts of data. Abhishek Mishra (2020)

The advantages of the CatBoost classifier that are relevant for our work are:

- CatBoost excels in circumstances when the data is constantly changing. Its boosting technique operates and provides predictions in a short period of time, resulting in quick model services.

- We can enhance conditions based on weighable parameters by using parameters in Cat boosting. It also allows for rapid monitoring of the Error function.
- CatBoost's stability while altering hyperparameters, especially when employed with large training sets, is one of its most appealing features. It produces the best results using the default values, saving time on parameter customization.

The shortcomings of the CatBoost classifier are:

- Catboost is created using the same methodology and characteristics as the "previous" generation of GBDT devices.
- Catboost's strength rests in its preprocessing of categorical information, prediction time, and model analysis.
- The training and optimization timeframes are Catboost's drawbacks.
- Though Catboost performs well with the default parameters, there are certain factors that when modified, result in a considerable improvement in outcomes.

For the study in this thesis, the hyper parameters of the CatBoost classifier have been optimized using Grid Search. Specifically, depth was set to 16, learning_rate was set to 0.11, iterations was set to 170, l2_leaf_reg was set to 3, border_count was set to 15 and thread_count was set to 4. Other hyper parameters have been used as Catboost's library default parameters.

Table 3.3

Catboost Classifier Parameters

Name	**Definition**	**Value**
depth	Depth of the tree. The range of supported values depends on the processing unit type and the type of the selected loss function.	Changed to "16"
learning_rate	The learning rate. Used for reducing the gradient step.	Changed to "0.11"
iterations	The maximum number of trees that can be built when solving machine learning problems.	Changed to "170"
l2_leaf_reg	Coefficient at the L2 regularization term of the cost function.	Changed to "3"

border_count	The number of splits for numerical features.	Changed to “15”
thread_count	The number of threads to use during the training.	Changed to “4”
bagging_temperature	Defines the settings of the Bayesian bootstrap. It is used by default in classification and regression modes.	Default “1”
sampling_frequency	Frequency to sample weights and objects when building trees.	Default “PerTreeLevel”
random_strength	The amount of randomness to use for scoring splits when the tree structure is selected. Use this parameter to avoid overfitting the model.	Default “1”
grow_policy	Defines how to perform greedy tree construction.	Default “SymmetricTree”
min_data_in_leaf	The minimum number of training samples in a leaf. CatBoost does not search for new splits in leaves with samples count less than the specified value.	Default “1”
max_leaves	The maximum number of leaves in the resulting tree.	Default “31”
class_weights	The values are used as multipliers for the object weights. This parameter can be used for solving binary classification and multiclassification problems.	Default “None” that means weight for all classes is set to 1
best_model_min_trees	The minimal number of trees that the best model should have.	Default “None”
has_time	Use the order of objects in the input data (do not perform random permutations during the Transforming categorical features to numerical features and Choosing the tree structure stages).	Default “False”

3.2.4 TABNET

TabNet takes in raw tabular data with no preprocessing and trains using gradient descent-based optimisation. TabNet use sequential attention to select characteristics at each decision step, enhancing interpretability and learning by allocating learning capacity to the most useful features. Feature selection is done on an individual basis, so it can be different for each row of the training dataset. TabNet uses a single deep

learning architecture for feature selection and reasoning, which is referred to as soft feature selection.

TabNet can enable two types of interpretabilities based on the design choices: local interpretability, which visualizes the importance of features and how they are combined for a single row, and global interpretability, which quantifies the contribution of each feature to the trained model across the dataset. (Adam Shafi, 2021)

The advantages of the Tabnet classifier that are relevant for our work are:

- Tabnet allows you to train a Multiregressor without having to construct distinct models for each class.
- It employs attention to choose the collection of features to focus on for a given data point and even visualizes this to show which portions receive attention for a certain choice. The number of features may be changed based on the concentrating features.
- It makes use of backprop to improve judgments and weights, giving it more control.
- Fine-tuning approaches that have worked for all deep-learning principles such as LR annealing, Custom loss, and so on are available.
- Tabnet automates the feature selection for you, so you don't have to worry about it.

The shortcomings of Tabnet classifier are:

- Tabnet performs well with tabular datasets.
- Tabnet, like the rest of the deep learning algorithms, is rather sophisticated.

- Grid Search and Randomized Search cannot be utilized during feature selection, feature selection must be done manually.
- Layer and neuron values must be assigned correctly, else the accuracies will be relatively low. (Tanul Singh, 2020)

For the study in this thesis, the hyper parameters of the Tabnet classifier have been optimized manually. Specifically, 4 layer and total of 1005 neurons 103, 512, 256, 128 and 6, respectively, were used. 103 and 6 comes from feature and class numbers. Epochs was set to 300, batch_size was set to 32 and learning_rate was set to 0.001. Other Parameters of the classifier are used as Tabnet's library default parameters.

Table 3.4

TABNET Parameters

Name	**Definition**	**Value**
max_epochs	Maximum number of epochs for training.	Changed to "300"
batch_size	Number of examples per batch.	Changed to "32"
learning_rate	Step size at each iteration while moving toward a minimum of a loss function.	Changed to "0.001"
attention_width	Width of the attention embedding for each mask.	Default "8"
virtual_batch_size	Size of the mini batches used for "Ghost Batch Normalization".	Default "128"
momentum	Momentum for batch normalization.	Default "0.02"

4. METHODOLOGY

4.1 DESCRIPTION OF THE DATA SET

The MIRAGE-2019 dataset was collected at the University of Napoli's ARCLAB laboratory. The capture sessions take place between May 2017 and May 2019. Three devices were used to generate mobile traffic: (i) the Xiaomi Mi5, (ii) the Google Nexus 7, and (iii) the Samsung Galaxy A5.

Over 280 volunteers participated in the dataset construction by doing one or two experimental sessions each. Each experimental session lasted no more than two hours. Each experimenter completed 5–10 minutes capture sessions throughout total 12 experimental session. In each capture session, the researcher was asked to do actions that mimicked popular usage of a single app, with the goal of exploring its functions in addition to the initial install, login, and registration. (Giuseppe Aceto Domenico Ciuonzo et al., 2019)

The MIRAGE-2019 dataset was built in a setting that included an IEEE 802.11g access point, which provided connectivity to the mobile devices that generated traffic while human experimenters used the applications. The capture server's access to the public Internet is provided through a wired connection, which performs Network Address Translation (NAT). Notably, the upstream connections to the public Internet do not represent a bandwidth bottleneck, therefore the parameters of the traffic stream passing via the access WLAN are unaffected. Each mobile device is also physically connected to the capture server through a USB hub, allowing the capture server to transmit instructions to the associated devices and receive answers over an off-band channel using the Android Debug Bridge (ADB). In order to correctly conduct the traffic capture operation, the devices must be rooted. Notably, the environment may gather traffic from several devices at the same time. The captured traffic is then processed and produce MIRAGE-2019.

The MIRAGE-2019 dataset collects traffic generated by 40 Android apps from 16 different categories, according to the Google Play app distribution portal (42matters.com, 2021).

To improve compatibility and usability, the MIRAGE-2019 dataset is available in JSON format: One JSON file corresponds to one collected PCAP trace. In particular,

for each biflow identified by its 5-tuple, the following data has been extracted: (i) per-packet data, (ii) per-flow characteristics, and (iii) per-flow metadata. The structure of each JSON file is shown in Figure 4.1.

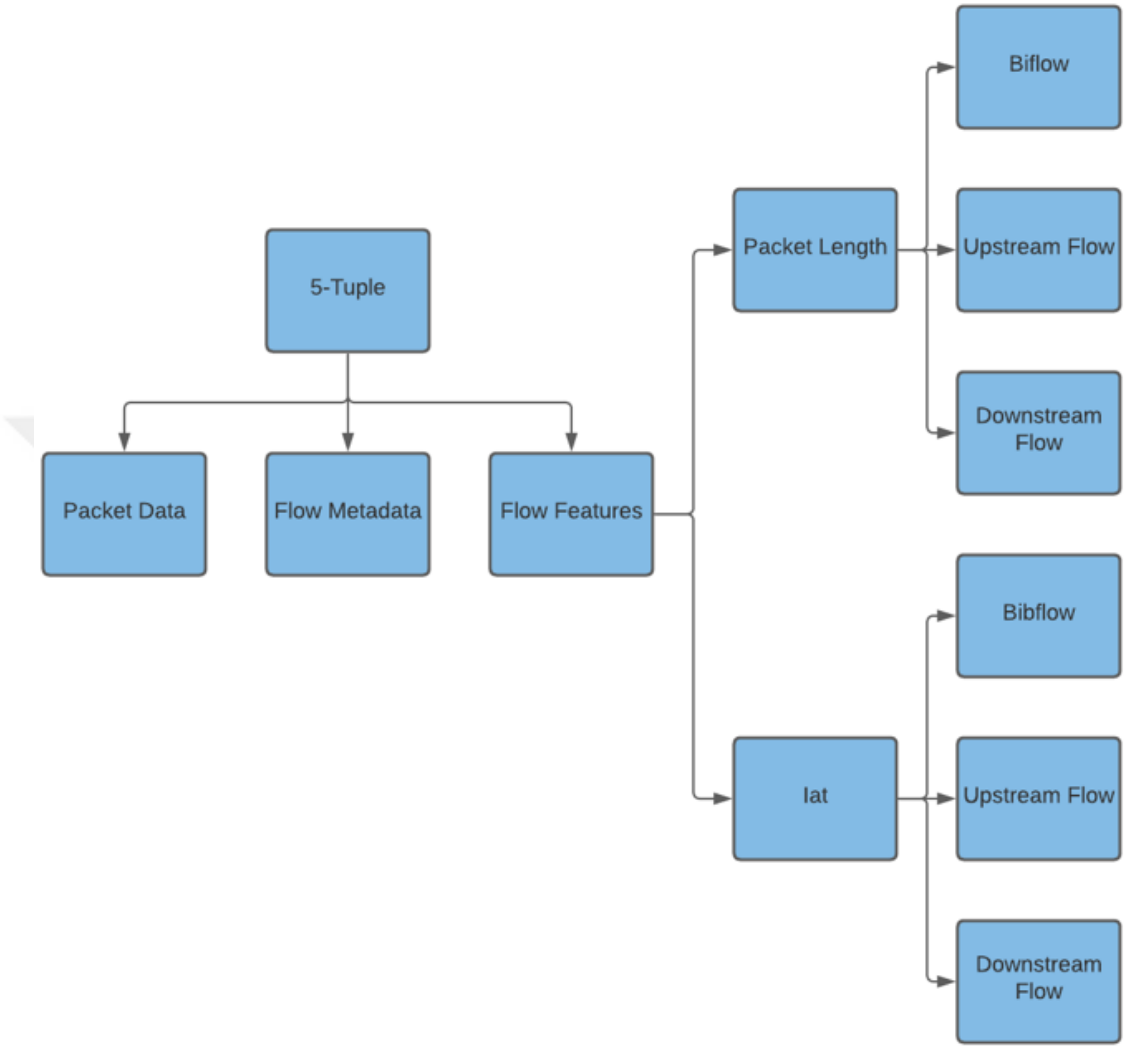

Figure 4.1: The structure of the JSON files that comprise MIRAGE-2019

Per-packet data: Per-packet data extracted from the first 32 packets of each biflow. Each biflow includes 7 informative fields and the L4 payload of the first 32 packets of each biflow. Each header field and description as shown in Table 4.1.

Table 4.1

Packet Data Fields

Field	**Description**
src_port	Source transport-layer port
dst_port	Destination transport-layer port
packet_dir	Packet direction (0 upstream, 1 downstream)
L4_payload_bytes	Number of bytes in L4 payload
iat	Inter-arrival time
TCP_win_size	TCP window size (0 for UDP packets)
L4_raw_payload	Byte-wise raw L4 payload (integer 2 [0; 255])

Per-flow Metadata: Per-flow metadata complementing per-flow features, being also related to complete biflow and upstream/downstream flows. Each header field and description as shown in Table 4.2.

Table 4.2

Metadata Fields

Field	**Description**
BF_label	Android-package name
BF_labeling_type	Exact or most-common labeling
{BF,UF,DF}_num_packets	Number of packets
{BF,UF,DF}_IP_packet_bytes	Total bytes in IP packets
{BF,UF,DF}_L4_payload_bytes	Total bytes in L4 payloads
{BF,UF,DF}_duration	(Bi)flow duration in seconds

4.2 DATA PREPROCESSING AND FEATURE ENGINEERING

In MIRAGE-2019 dataset, data is presented in JSON format. They are converted to the .xls format for each packet by using a Java application. After this conversion dataset exported as .csv from Excel.

During the data processing, packet data and flow metadata had to be eliminated since they did not impact category selection data in the dataset. After this elimination data which is under the flow features is used. In addition, the application name BF_label from flow_metadata was used.

The subset of the dataset used in this thesis has the main components shown in Figure 4.2.

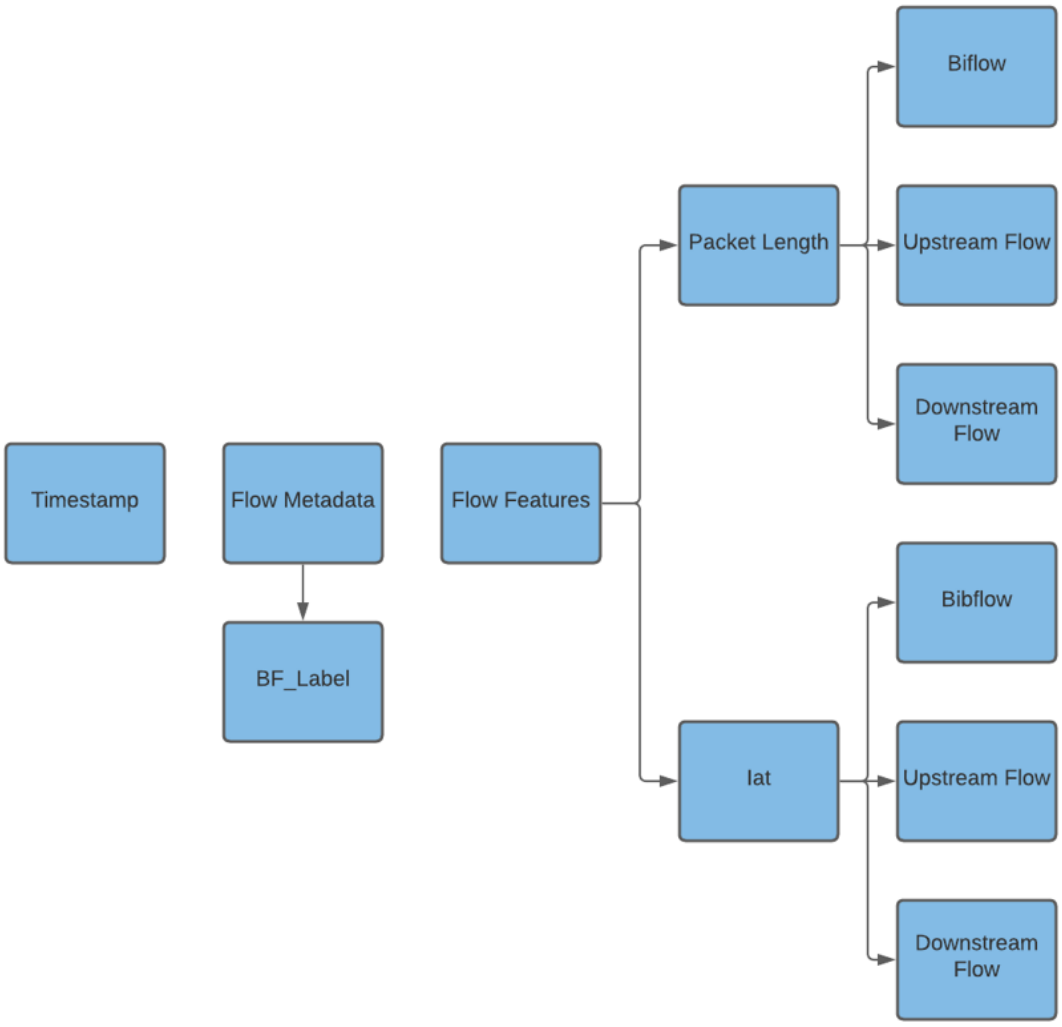

Figure 4.2: The subset of the MIRAGE 2019

Per-flow features: To provide information on the whole biflow and corresponding upstream and downstream flows, 17 statistical features have been selected and computed on the sets of upstream, downstream, and complete (i.e. both of them) IP packet lengths and inter-arrival times, for a total of 102 per-flow features. Previous works in the field of traffic classification via machine learning successfully leveraged these features to feed the classification algorithms they devised (Giuseppe Aceto et al., 2019, V. F. Taylor et al., 2018, V. Carela-Español et al., 2010).

Flow features include two data types which are packet length and iat. These two types include biflow, upstream flow and downstream flow. Each flow data includes min, max, mean, std, var, mad, skew, kurtosis and q_percentile these flow data are statistics of the packet size distribution and are shown in Table 4.3.

Table 4.3

Flow data features

Feature Name	**Description**
min	Minimum
max	Maximum
mean	Arithmetic Mean
std	Standard Deviation
var	Variance
mad	Mean Absolute Deviation
skew	Unbiased Sample Skweness
kurtosis	Unbiased Fisher Kurtosis
q_percentile	q^th percentile (q in [10 : 10 : 90])

In this thesis, we categorize the application names from the BF_Label in the dataset into a target attribute called Appcategory. The original data set suggests 20 application

types, however, in order to make the multi-class classification easier, we have provided a mapping into a total of 6 classes, which are depicted in Table 4.4.

Table 4.4

Application Name Appcategory Change

Application Name	**Appcategory**
com.contextlogic.wish	Entertainment
com.duolingo	Entertainment
com.iconology.comics	Entertainment
com.spotify.music	Entertainment
de.motain.iliga	Entertainment
it.subito	Entertainment
com.facebook.katana	Social
com.facebook.orca	Social
com.google.android.youtube	Social
com.pinterest	Social
com.twitter.android	Social
com.joelapenna.foursquared	Social
com.waze	Utility
com.accuweather.android	Utility
com.dropbox.android	Utility
com.groupon	Lifestyle
com.tripadvisor.tripadvisor	Lifestyle
com.trello	Productivity
com.viber.voip	Productivity
air.com.hypah.io.slither	Game

17 statistical characteristics computed on upstream, downstream, and biflow. Since there are 6 different flows in the dataset, the total number of features is 17 so 6*17 = 102 and there is also a timestamp feature, and as a result, it consists of a total of 103 features. A sample of our data set is shown in Figure 4.3.

	pl_biflow_min	pl_biflow_max	pl_biflow_mean	pl_biflow_std	pl_biflow_var	...	iat_downstream_flow_90_percentile	timestamp	appcategory
0	52.0	1420.0	325.857143	412.000050	169744.040816	...	2.930874	1494596682	Game
1	52.0	1470.0	408.533333	514.797116	265016.071111	...	69.745482	1494596682	Game
2	40.0	1420.0	246.761905	376.542909	141784.562358	...	96.013842	1494596682	Game
3	52.0	1470.0	232.764706	368.126182	135516.885813	...	0.036781	1494596682	Game
4	52.0	1500.0	1007.067161	685.864928	470410.699954	...	0.009008	1494596682	Game
...	...	...	...	...	...	...	...	...	...
112786	40.0	1500.0	269.692308	442.148667	195495.443787	...	0.128733	1558955908	Social
112787	40.0	1500.0	580.493506	638.022892	407073.210997	...	3.343453	1558955908	Social
112788	52.0	1500.0	357.772727	524.137988	274720.630165	...	5.702617	1558955908	Social
112789	52.0	1500.0	305.909091	481.566186	231905.991736	...	4.856056	1558955908	Social
112790	52.0	1470.0	319.230769	478.131404	228609.639053	...	0.111941	1558955908	Social

Figure 4.3: Dataset Example

Timestamp type is int64, category type is object rest of the columns data types are float64. Type of features is shown in Figure 4.4.

```
pl_biflow_min                          float64
pl_biflow_max                          float64
pl_biflow_mean                         float64
pl_biflow_std                          float64
pl_biflow_var                          float64
                                         ...
iat_downstream_flow_70_percentile      float64
iat_downstream_flow_80_percentile      float64
iat_downstream_flow_90_percentile      float64
timestamp                                int64
appcategory                             object
```

Figure 4.4: Feature Types

After these processes, a total of 103 features and 112813 samples were created. After eliminating packets with null data, 112791 (It should be noted that the number of rows begins with 0) samples remained. The number of samples were distributed as follows; Entertainment (44566), Social (27616), Utility (27130), Lifestyle (5469), Productivity (5068), Game (2942). The distribution is plotted in Figure 4.5.

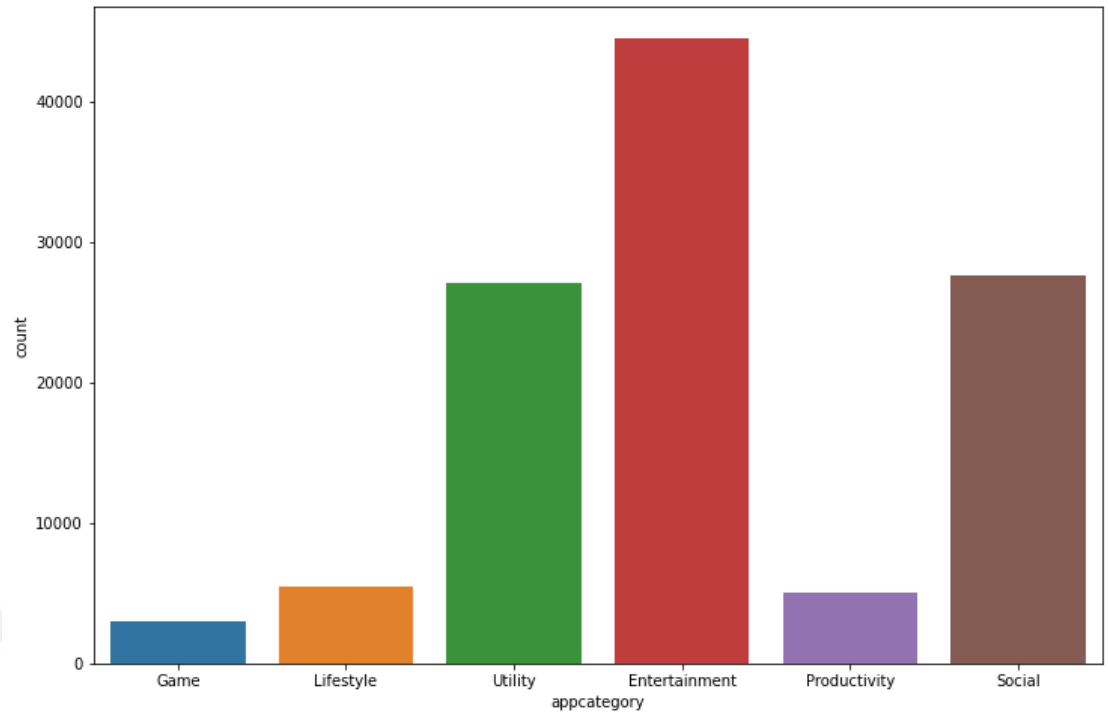

Figure 4.5: Class Distribution of Features

Furthermore, because the LightGBM, Catboost, and Tabnet algorithms need digitization of the targeted feature while classifying, a change has been applied to dataset based on the category numerical value shown in Table 4.5.

Table 4.5

Appcategory Numerical Value Change

Appcategory	Numerical Value
Entertainment	0
Social	1
Utility	2
Lifestyle	3
Productivity	4
Game	5

5. RESULTS

The performance of each machine learning algorithm used, namely the Random Forest, LightGBM, Catboost, and Tabnet, on category classification accuracy is examined independently in this chapter.

5.1 RANDOM FOREST

As mentioned in the Background section, max_depth was set to 32 and n_estimators was set to 512. On parameter selection process, GridsearchCV was used and parameters that negatively affect the success rate when any changes are made are the default value. In addition to this within GridsearchCV cross validation is set to 2 prevent overfitting.

5.1.1 RANDOM FOREST ACCURACY RESULT

A total of ten runs were completed. The overall and class-wise success rates were computed. Finally, the average of each success rate was calculated. Random Forest results is shown in Table 5.1.

Table 5.1

Random Forest Accuracy Result

Run Number	Overall Accuracy	Entertainment	Social	Utility	Lifestyle	Productivity	Game
1	0,791	0,898	0,779	0,816	0,37	0,377	0,668
2	0,796	0,898	0,79	0,821	0,361	0,365	0,621
3	0,794	0,9	0,783	0,813	0,372	0,374	0,669
4	0,794	0,891	0,791	0,818	0,369	0,365	0,638
5	0,792	0,899	0,777	0,826	0,36	0,385	0,6
6	0,796	0,899	0,788	0,817	0,359	0,383	0,633
7	0,793	0,902	0,778	0,81	0,354	0,383	0,631
8	0,791	0,894	0,77	0,826	0,359	0,374	0,633
9	0,796	0,902	0,785	0,82	0,379	0,381	0,624
10	0,792	0,897	0,774	0,818	0,357	0,381	0,639
Average	0,7935	0,898	0,7815	0,8185	0,364	0,3768	0,6356

With the Random Forest algorithm, the highest and lowest overall accuracies achieved were 0,796 and 0,791, yielding a difference of 0.5% among ten independent runs of the algorithm.

Among the individual categories, the highest accuracy was consistently achieved for Entertainment category, followed by Utility, Social and Game in each run. The highest accuracy achieved for Entertainment is 90.2%. The Lifestyle and Productivity categories yield quite low accuracies, below 40%. The highest variation in accuracy (6.9%) was observed for the Game category, while highly accurate Entertainment category classification yielded the lowest variations of 1.1%.

5.1.2 RANDOM FOREST CONFUSION MATRIX

Random Forest Confusion matrix is shown in Figure 5.1.

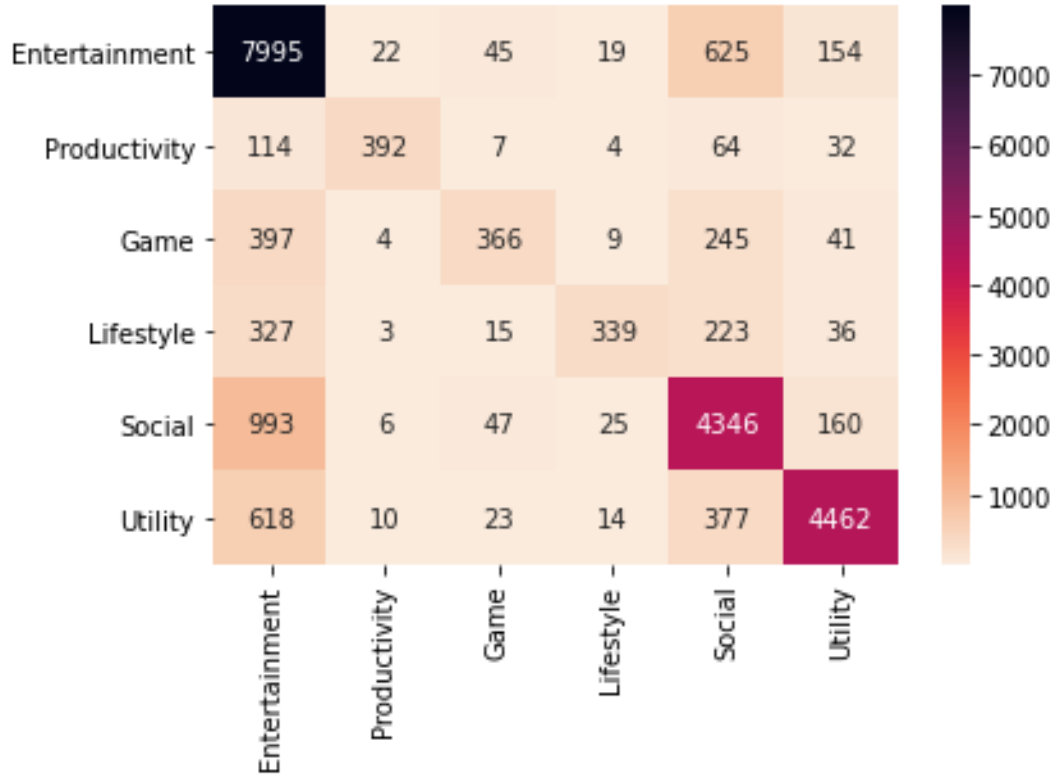

Figure 5.1: Random Forest Confusion Matrix

The Entertainment feature was frequently mistaken for the Social feature, and vice versa. Specifically, in 7.8% of the cases, Entertainment app was categorized as a Social

app; and in 14.3% of the cases, Social was categorized as Entertainment. To identify the underlying reasons, we observed feature importance.

5.1.3 RANDOM FOREST FEATURE IMPORTANCE

The feature_importances_ function used to calculate 20 most important features and the matplotlib.pyplot library was used to visualize the results. Random Forest Feature Importances shown in Figure 5.2.

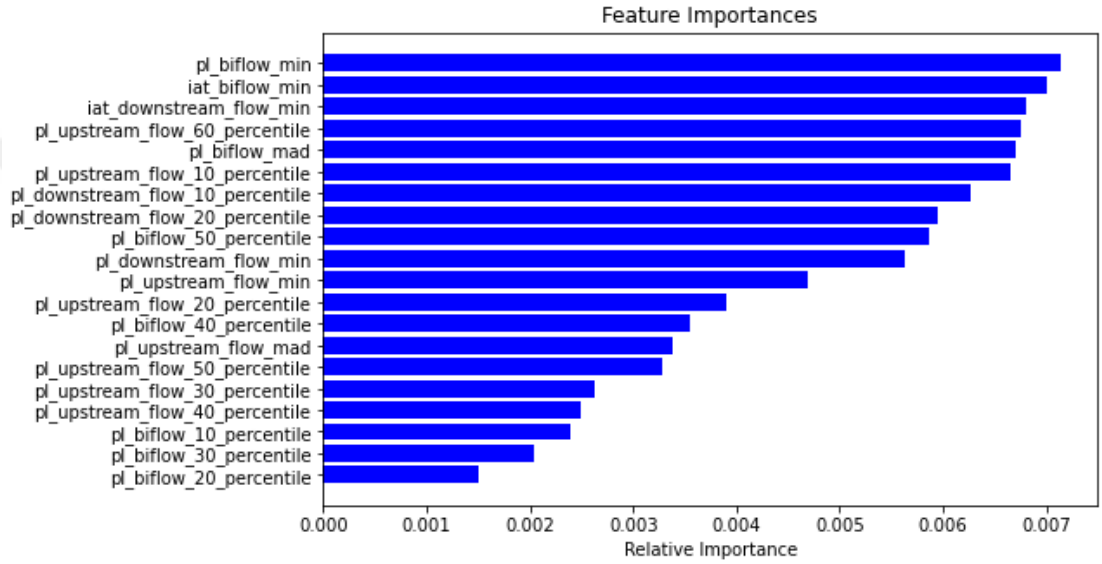

Figure 5.2: Random Forest Feature Importance

5.1.3.1 RANDOM FOREST FEATURE IMPORTANCE

The top 3 features used by the Random Forest algorithm are the minimum bi-flow packet length, minimum inter-arrival time of the bi-flow and the minimum inter-arrival time of the downstream. Table 5.2 depicts the ranges of these values for the 6 app categories. The table shows that the mean values for bi-flow packet length and bi-flow minimum inter-arrival time are very close for these two apps.

Table 5.2

Random Forest Feature Overview

[App]_featureID	MIN	MAX	MEAN	STDEV
E_pl_biflow_min	40	64	46.21	5.98
E_iat_biflow_min	0.000001	85.91	0.23	3.68

E_iat_downstream_flow_min	0.000001	240.02	0.52	5.84
S_pl_biflow_min	30	92	46.55	6.66
S_iat_biflow_min	0.000001	85.91	0.233850	3.68
S_iat_downstream_flow_min	0.000001	230.07	0.9	7.2
U_pl_biflow_min	40	60	48.83	5.27
U_iat_biflow_min	0.000001	60.58	0.17	3.19
U_iat_downstream_flow_min	0.000001	240.007	0.51	5.49
L_pl_biflow_min	40	64	46.08	5.97
L_iat_biflow_min	0.000001	60.58	0.17	3.19
L_iat_downstream_flow_min	0.000001	240.007	0.51	5.49
P_pl_biflow_min	36	60	45.76	5.96
P_iat_biflow_min	0.000001	61.47	0.63	6.08
P_iat_downstream_flow_min	0.000002	230.06	1.59	11.06
G_pl_biflow_min	40	56	47.88	5.66
G_iat_biflow_min	0.000001	63.2	0.42	4.96
G_iat_downstream_flow_min	0.000002	230.05	0.93	8.83

Mean values for bi-flow packet length and bi-flow minimum inter-arrival time are very close for Entertainment and Social.

5.2 CATBOOST

As mentioned in the Background section, depth was set to 16, learning_rate was set to 0.11, iterations was set to 170, l2_leaf_reg was set to 3, border_count was set to 15 and thread_count was set to 4. On parameter selection process, GridsearchCV was used and parameters that negatively affect the success rate when any changes are made are the default value. In addition to this within GridsearchCV cross validation is set to 2 prevent overfitting.

5.2.1 CATBOOST ACCURACY RESULT

A total of ten runs were completed. The overall and class-wise success rates were computed. Finally, the average of each success rate was calculated. Results are shown in Table 5.3.

Table 5.3

Catboost Accuracy Result

Run Number	Overall Accuracy	Entertainment	Social	Utility	Lifestyle	Productivity	Game
1	0,76	0,872	0,757	0,78	0,372	0,392	0,6
2	0,76	0,867	0,768	0,779	0,379	0,363	0,577
3	0,763	0,871	0,753	0,79	0,385	0,369	0,633
4	0,762	0,873	0,751	0,782	0,348	0,368	0,632
5	0,763	0,872	0,766	0,786	0,365	0,369	0,6
6	0,759	0,868	0,762	0,779	0,359	0,377	0,616
7	0,759	0,859	0,754	0,792	0,34	0,36	0,6
8	0,759	0,871	0,752	0,78	0,35	0,373	0,633
9	0,764	0,872	0,749	0,792	0,341	0,355	0,65
10	0,762	0,877	0,759	0,78	0,38	0,38	0,6
Average	0,7611	0,8702	0,7571	0,784	0,3619	0,3706	0,6141

With the Catboost algorithm, the highest and lowest overall accuracies achieved were 0,764 and 0,759, yielding a difference of 0.5% among ten independent runs of the algorithm.

Among the individual categories, the highest accuracy was consistently achieved for Entertainment category, followed by Utility, Social and Game in each run. The highest accuracy achieved for Entertainment is 87.7%. The Lifestyle and Productivity categories yield quite low accuracies, below 40%. The highest variation in accuracy (7.3%) was observed for the Game category, while Utility category classification yielded the lowest variations of 1.3%.

5.2.2 CATBOOST CONFUSION MATRIX

The sklearn library was used to calculate the confusion matrix, and the seaborn library was used to visualize the results. Catboost Confusion matrix shown in Figure 5.3.

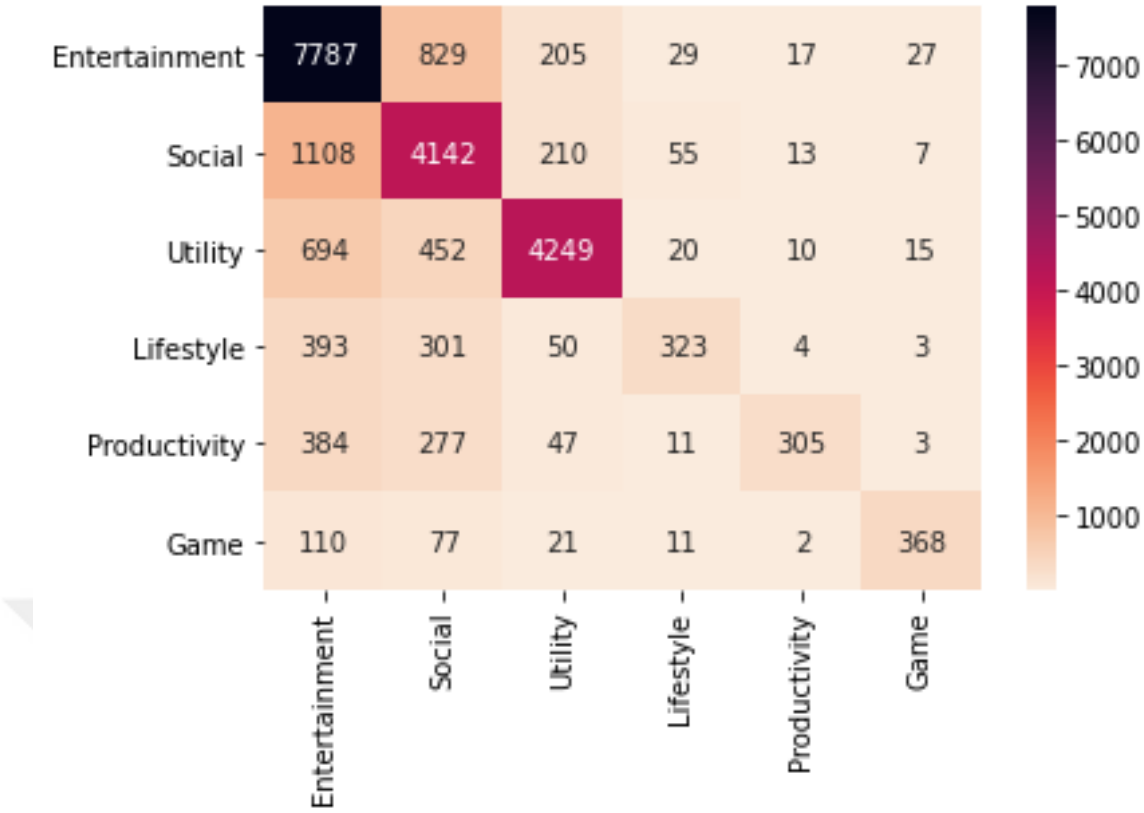

Figure 5.3: Catboost Confusion Matrix

The Entertainment feature was frequently mistaken for the Social feature, and vice versa. Specifically, in 12.5% of the cases, Entertainment app was categorized as a Social app; and in 16.6% of the cases, Social was categorized as Entertainment. To identify the underlying reasons, we observed feature importance.

5.2.3 CATBOOST FEATURE IMPORTANCE

In addition, the feature_importances_ function used to calculate 20 most important features and the matplotlib.pyplot library was used to visualize the results. Catboost Feature Importances shown in Figure 5.4.

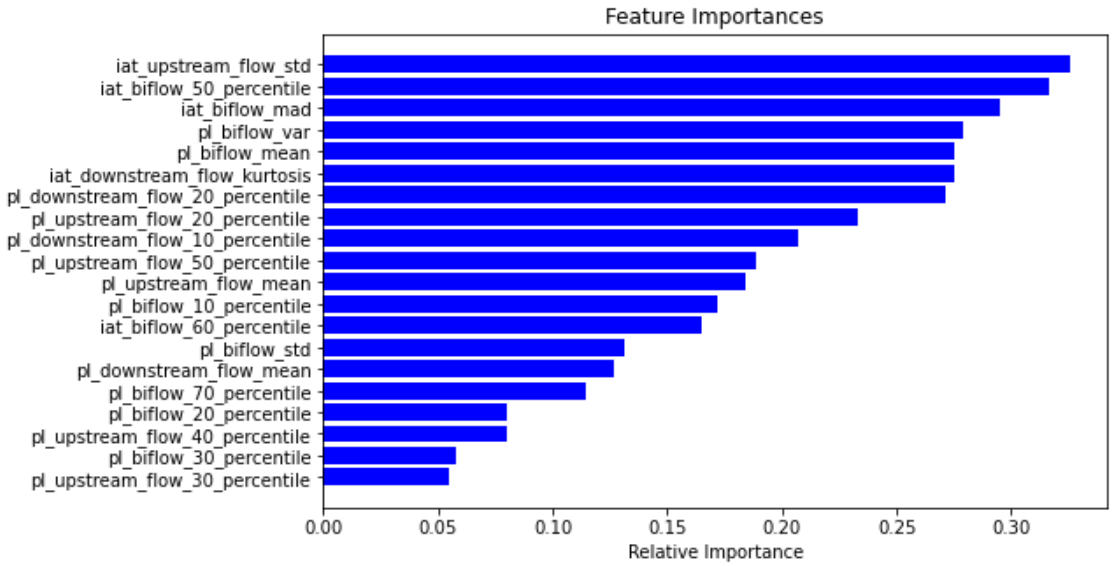

Figure 5.4: Catboost Feature Importance

5.2.3.1 CATBOOST FEATURE OVERVIEW

The top 3 features used by the CatBoost algorithm are the iat upstream flow standard deviation, iat biflow 50 percentile and the iat biflow mad. Table 5.4 depicts the ranges of these values for the 6 app categories. The table shows that the mean values for bi-flow packet length and bi-flow minimum inter-arrival time are very close for these two apps.

Table 5.4

Catboost Feature Overview

[App]_featureID	MIN	MAX	MEAN	STDEV
E_iat_upstream_flow_std	0	175.6	8.75	16.1
E_iat_biflow_50_percentile	0.000010	240.02	0.74	5.9
E_iat_biflow_mad	0	90.02	0.32	2.96
S_iat_upstream_flow_std	0	327.4	13.6	21.7
S_iat_biflow_50_percentile	0.000013	173.36	1.56	8.26
S_iat_biflow_mad	0	114.99	0.73	4.55
U_iat_upstream_flow_std	0	408.7	7.37	16.43
U_iat_biflow_50_percentile	0.000502	240.007	0.78	5.92

U_iat_biflow_mad	0	215.54	0.43	3.508
L_iat_upstream_flow_std	0	176.79	14.81	20.38
L_iat_biflow_50_percentile	0.000505	240.006	1.92	9.67
L_iat_biflow_mad	0	55.002	0.94	5.05
P_iat_upstream_flow_std	0	339.65	15.83	23.64
P_iat_biflow_50_percentile	0.000011	115.033	2.41	10.78
P_iat_biflow_mad	0.000000	114.99	1.09	5.94
G_iat_upstream_flow_std	0	109.34	13.49	22.49
G_iat_biflow_50_percentile	0.000503	115.03	1.72	8.09
G_iat_biflow_mad	0	114.99	0.98	5.21

5.3 LIGHTGBM

As mentioned in the Background section, num_leaves was set to 100, min child samples was set to 15, max_depth was set to 20, learning_rate was set to 0.2 and reg_alpha was set to 0.03. On parameter selection process, GridsearchCV was used and parameters that negatively affect the success rate when any changes are made are the default value. In addition to this within GridsearchCV cross validation is set to 2 prevent overfitting.

5.3.1 LIGHTGBM ACCURACY RESULT

A total of ten runs were completed. The overall and class-wise success rates were computed. Finally, the average of each success rate was calculated. LightGBM results is shown in Table 5.5.

Table 5.5

LightGBM Result

Run Number	Overall Accuracy	Entertainment	Social	Utility	Lifestyle	Productivity	Game
1	0,792	0,879	0,787	0,829	0,395	0,372	0,651
2	0,789	0,881	0,786	0,823	0,379	0,378	0,631

3	0,794	0,876	0,796	0,827	0,403	0,393	0,635
4	0,793	0,883	0,792	0,824	0,394	0,393	0,626
5	0,792	0,885	0,784	0,814	0,394	0,391	0,66
6	0,786	0,874	0,772	0,823	0,377	0,41	0,652
7	0,791	0,878	0,781	0,823	0,41	0,375	0,629
8	0,792	0,881	0,788	0,818	0,384	0,417	0,621
9	0,791	0,883	0,788	0,822	0,369	0,391	0,613
10	0,792	0,875	0,79	0,831	0,372	0,404	0,63
Average	0,7912	0,8795	0,7864	0,8234	0,3877	0,3924	0,6348

With the LightGBM algorithm, the highest and lowest overall accuracies achieved were 0,794 and 0,789, yielding a difference of 0.8% among ten independent runs of the algorithm.

Among the individual categories, the highest accuracy was consistently achieved for Entertainment category, followed by Utility, Social and Game in each run. The highest accuracy achieved for Entertainment is 97.7%. The Lifestyle and Productivity categories yield quite low accuracies, below 40%. The highest variation in accuracy (4.7%) was observed for the Game category, while highly accurate Entertainment category classification yielded the lowest variations of 1.1%.

5.3.2 LIGHTGBM RESULT CONFUSION MATRIX

The sklearn library was used to calculate the confusion matrix, and the seaborn library was used to visualize the results. LightGBM Confusion matrix shown in Figure 5.5.

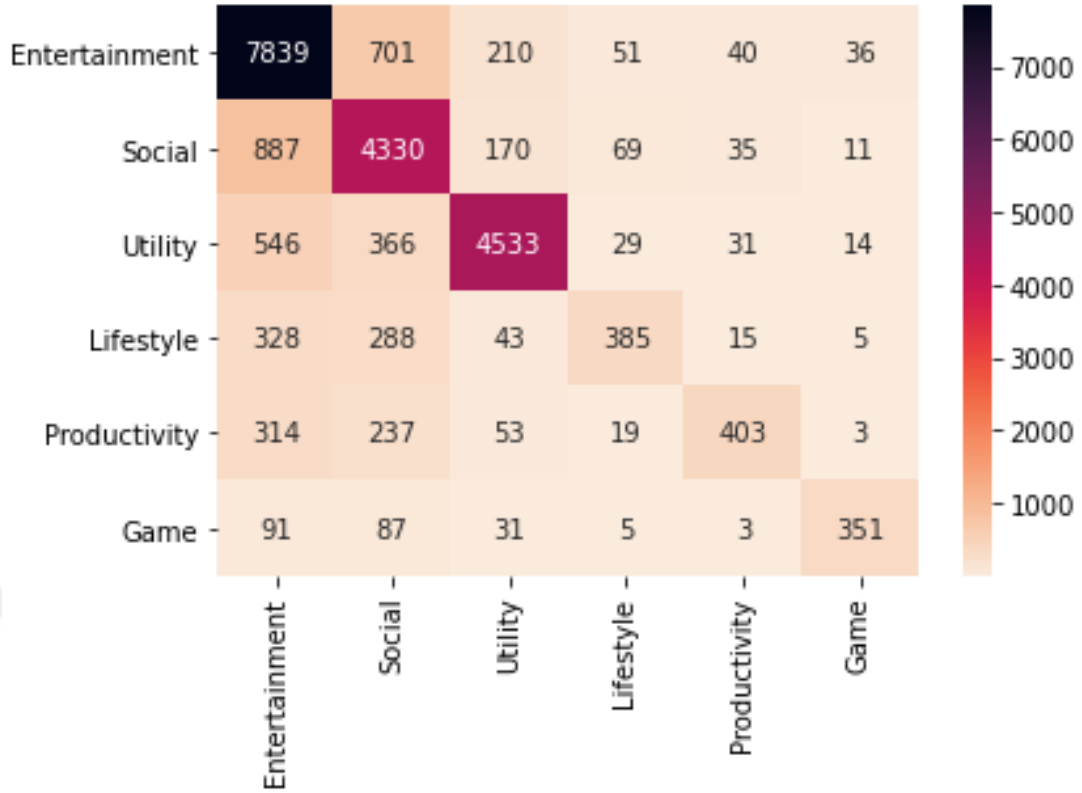

Figure 5.5: LightGBM Confusion Matrix

The Entertainment feature was frequently mistaken for the Social feature, and vice versa. Specifically, in 11.3% of the cases, Entertainment app was categorized as a Social app; and in 16.1% of the cases, Social was categorized as Entertainment. To identify the underlying reasons, we observed feature importance.

5.3.3 LIGHTGBM FEATURE IMPORTANCE

In addition, the feature_importances_ function used to calculate 14 of the 20 most important features and the matplotlib.pyplot library was used to visualize the results. LightGBM Feature Importances shown in Figure 5.6.

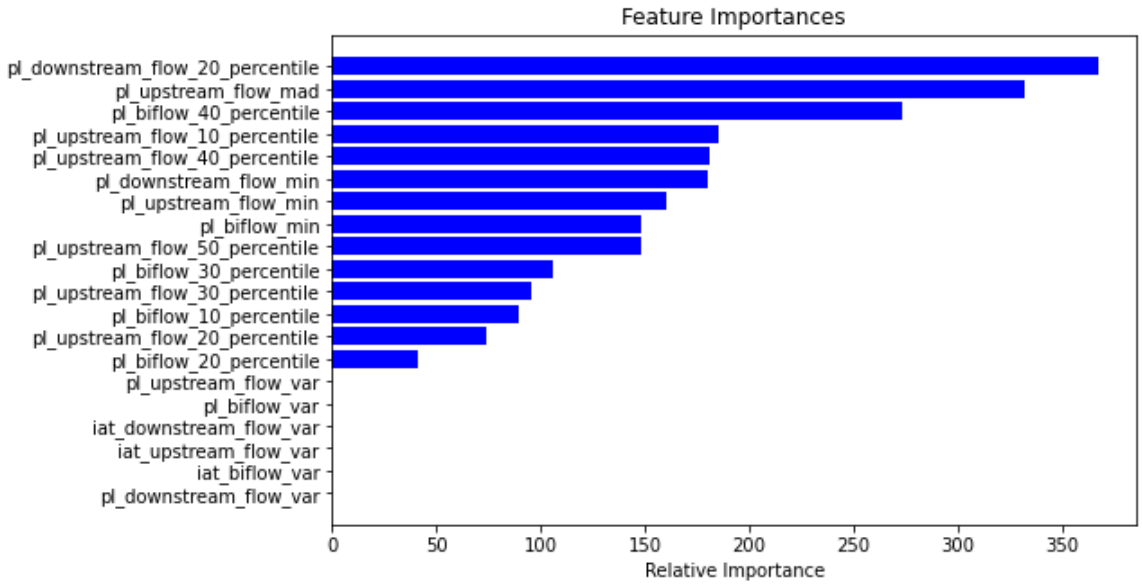

Figure 5.6: LightGBM Feature Importance

5.3.3.1 LIGHTGBM FEATURE OVERVIEW

The top 3 features used by the LightGBM algorithm are the minimum bi-flow packet length, minimum inter-arrival time of the bi-flow and the minimum inter-arrival time of the downstream. Table 5.6 depicts the ranges of these values for the 6 app categories. The table shows that the mean values for bi-flow packet length and bi-flow minimum inter-arrival time are very close for these two apps.

Table 5.6

LightGBM Feature Overview

[App]_featureID	MIN	MAX	MEAN	STDEV
E_pl_downstream_flow_20_percentile	40	1500	292.008	512.5
E_pl_upstream_flow_mad	0	714	13.504	57.62
E_pl_biflow_40_percentile	40	1500	107.05	258.56
S_pl_downstream_flow_20_percentile	40	1500	266.44	479.77
S_pl_upstream_flow_mad	0	718	13.15	57.28
S_pl_biflow_40_percentile	40	1500	132.207	302.28
U_pl_downstream_flow_20_percentile	40	1500	230.13	439.3
U_pl_upstream_flow_mad	0	722	9.064	44.63
U_pl_biflow_40_percentile	40	1500	113.28	273.01
L_pl_downstream_flow_20_percentile	40	1500	232.001	443.802

L_pl_upstream_flow_mad	0	645	12.35	51.707
L_pl_biflow_40_percentile	40	1500	110.204	258.61
P_pl_downstream_flow_20_percentile	40	1500	163.83	360.63
P_pl_upstream_flow_mad	0	677	16.51	65.75
P_pl_biflow_40_percentile	40	1500	98.19	232.22
G_pl_downstream_flow_20_percentile	40	1500	144.301	326.28
G_pl_upstream_flow_mad	00	672	5.86	34.309
G_pl_biflow_40_percentile	40	1500	103.84	258.98

20 percentile of packet length in downstream flow, mean absolute deviation of packet length in upstream flow and 40 percentile of packet length in biflow are very close for Entertainment and Social.

5.4 TABNET

As mentioned in the Background section 4 layer and total of 1005 neurons 103, 512, 256, 128 and 6, respectively, were used. Epochs was set to 300, batch_size was set to 32 and learning_rate was set to 0.001. On parameter selection process, GridsearchCV was used and parameters that negatively affect the success rate when any changes are made are the default value. In addition to this within GridsearchCV cross validation is set to 2 prevent overfitting.

5.4.1 TABNET ACCURACY RESULT

A total of ten runs were completed. The overall and class-wise success rates were computed. Finally, the average of each success rate was calculated. Tabnet results is shown in Table 5.7.

Table 5.7

Tabnet Result

Run Number	Overall Accuracy	Entertainment	Social	Utility	Lifestyle	Productivity	Game
1	0,72	0,84	0,68	0,77	0,31	0,24	0,43
2	0,72	0,83	0,69	0,78	0,31	0,32	0,29
3	0,72	0,85	0,69	0,76	0,30	0,25	0,30
4	0,72	0,85	0,64	0,80	0,32	0,27	0,38
5	0,73	0,84	0,67	0,80	0,32	0,28	0,33
6	0,72	0,84	0,66	0,81	0,29	0,25	0,37
7	0,72	0,84	0,67	0,78	0,29	0,26	0,43
8	0,73	0,84	0,67	0,80	0,28	0,31	0,37

9	0,72	0,84	0,65	0,80	0,31	0,30	0,35
10	0,72	0,84	0,66	0,78	0,32	0,31	0,30
Average	72,20	84,10	66,80	78,80	30,50	27,90	35,50

With the Tabnet algorithm, the highest and lowest overall accuracies achieved were 0,73 and 0,72, yielding a difference of 1% among ten independent runs of the algorithm.

Among the individual categories, the highest accuracy was consistently achieved for Entertainment category, followed by Utility, Social and Game in each run. The highest accuracy achieved for Entertainment is 97.7%. The Lifestyle and Productivity categories yield quite low accuracies, below almost 30%. The highest variation in accuracy (6.9%) was observed for the Game category, while highly accurate Entertainment category classification yielded the lowest variations of 1.1%.

5.4.2 TABNET CONFUSION MATRIX

The sklearn library was used to calculate the confusion matrix, and the seaborn library was used to visualize the results. Tabnet Confusion matrix shown in Figure 5.7.

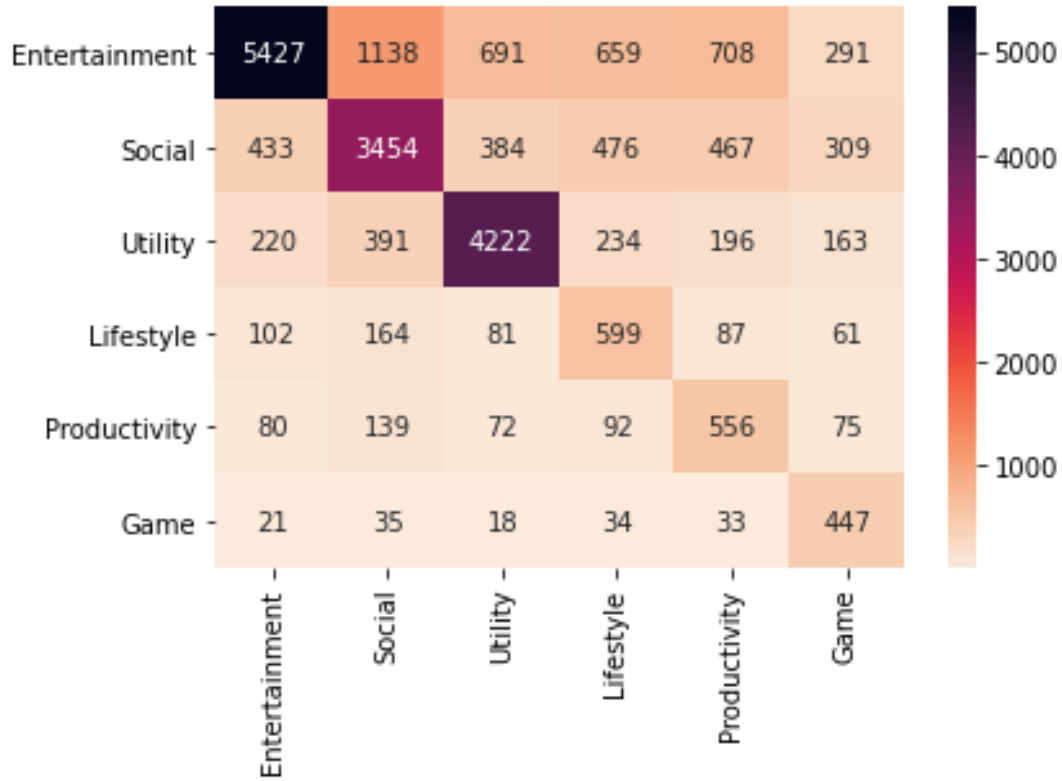

Figure 5.7: Tabnet Confusion Matrix

The Social feature was frequently mistaken for the Entertainment feature, and vice versa. Specifically, in 32.9% of the cases, Social app was categorized as a

Entertainment app; and in 16.3% of the cases, Utility was categorized as Entertainment. To identify the underlying reasons, we observed feature importance.

5.4.3 TABNET FEATURE IMPORTANCE

In addition, the feature_importances_ function used to calculate 20 most important features and the matplotlib.pyplot library was used to visualize the results. Tabnet feature importances shown in Figure 5.8.

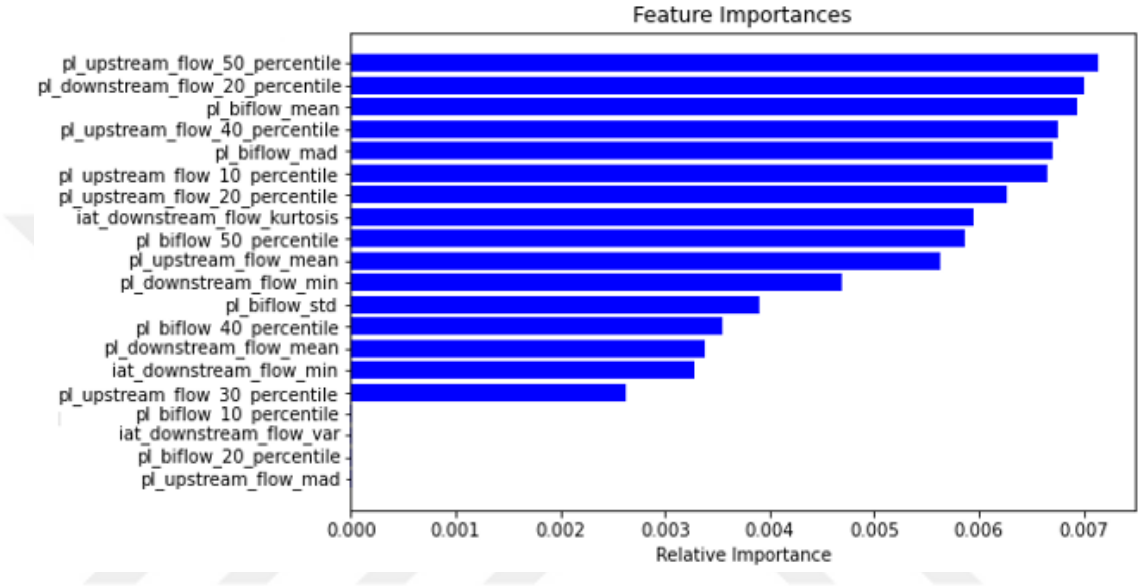

Figure 5.8: Tabnet Feature Importance

5.4.3.1 TABNET FEATURE OVERVIEW

The top 3 features used by the Tabnet algorithm are the minimum bi-flow packet length, minimum inter-arrival time of the bi-flow and the minimum inter-arrival time of the downstream. Table 5.8 depicts the ranges of these values for the 6 app categories. The table shows that the mean values for bi-flow packet length and bi-flow minimum inter-arrival time are very close for these two apps.

Table 5.8

Tabnet Feature Overview

[App]_featureID	MIN	MAX	MEAN	STDEV
E_pl_downstream_flow_20_percentile	40	1500	292.008	512.5
E_pl_upstream_flow_mad	0	714	13.504	57.62
E_pl_biflow_40_percentile	40	1500	107.05	258.56
S_pl_downstream_flow_20_percentile	40	1500	266.44	479.77
S_pl_upstream_flow_mad	0	718	13.15	57.28
S_pl_biflow_40_percentile	40	1500	132.207	302.28
U_pl_downstream_flow_20_percentile	40	1500	230.13	439.3
U_pl_upstream_flow_mad	0	722	9.064	44.63
U_pl_biflow_40_percentile	40	1500	113.28	273.01
L_pl_downstream_flow_20_percentile	40	1500	232.001	443.802
L_pl_upstream_flow_mad	0	645	12.35	51.707
L_pl_biflow_40_percentile	40	1500	110.204	258.61
P_pl_downstream_flow_20_percentile	40	1500	163.83	360.63
P_pl_upstream_flow_mad	0	677	16.51	65.75
P_pl_biflow_40_percentile	40	1500	98.19	232.22
G_pl_downstream_flow_20_percentile	40	1500	144.301	326.28
G_pl_upstream_flow_mad	00	672	5.86	34.309
G_pl_biflow_40_percentile	40	1500	103.84	258.98

Upstream flow 50 percentile and biflow mean vlaues are especially cloes for Lifestyle and Productivity, Entertainment and Social.

6. DISCUSSION

This thesis focused on analyzing the application category from the relevant features using classification techniques. For this section examines the comparison of machine learning algorithms which are used.

In order to better evaluate the performance of the algorithms, the results are shown in Table 6.1.

Table 6.1

Comparison of Accuracies of Algorithms

Algotihm Name	Overall Accuracy	Entertainment	Social	Utility	Lifestyle	Productivity	Game
RandomForest	0,7935	0,898	0,7815	0,8185	0,364	0,3768	0,6356
LightGBM	0,7912	0,8795	0,7864	0,8234	0,3877	0,3924	0,6348
Catboost	0,7611	0,8702	0,7571	0,784	0,3619	0,3706	0,6141
Tabnet	0,72	0,84	0,668	0,788	0,3	0,279	0,355

In addition, the performance rates of the algorithms are shown in the Figure 6.1.

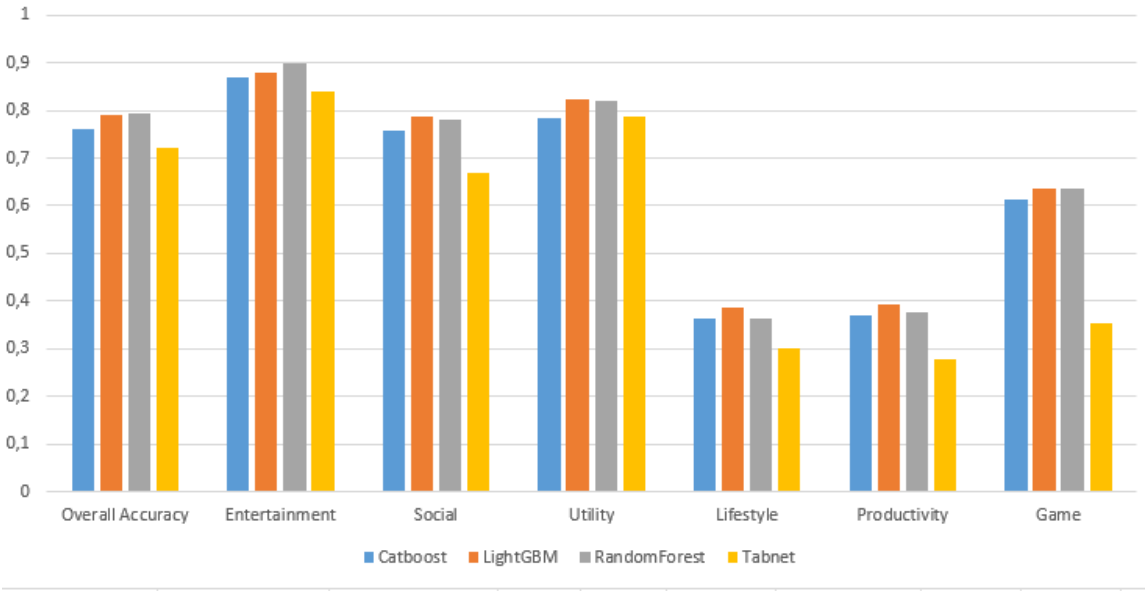

Figure 6.1: Comparison of accuracy rates of algorithms

As can be seen in Figure 6.1 as the number of samples (Figure 4.7.) decreased, the performance of the algorithms decreased as well. However, Tabnet was the most affected by this decrease.
Algorithms were run 10 times in total, and can be seen under the Results section, the best and worst results were produced for each category and these differences were revealed. The the highest difference 0.073 was calculated for the Game category in the Catboost algorithm, while the lowest difference 0.002 was calculated in the Entertainment category in the Tabnet algorithm.

When the Confusion Matrixes are examined for each algorithm, it can be observed that Tabnet was the algorithm that mostly misake the categories with each other. As a result, Random Forest is slightly better than LightGBM, followed by Catboost and Tabnet, respectively.

Without additional metadata indicating the app's functionality, it may not be possible to significantly improve the accuracies obtained in the classification. As an example, the packet length for different apps of the same category may differ. For example, Pinterest and YouTube are both in the Social app category; but while former streams images the latter streams video. Similarly, while both Spotify and Duolingo are placed in the Entertainment category, the upload frequency on both applications may differ: while the user may only stream downlink for minutes (or hours) with Spotify, Duolingo (an interactive language app) involves periodic uplink messages, causing in different inter-arrival durations.

When we classified according to the traffic generated by the packages, we could only predict the labels suggested by google appstore with this accuracy.

It turns out that our classification accuracy is due to inaccuracies in appstore naming, or different features of different applications of the same type. To improve this, application-based predictions can be made. In addition, by clustering, we can observe how different the applications in the same category are from each other.

In our future work, we plan to categorize apps based on functionalities (a more coarse grouping than the 6 groups in this work) or try to categorize directly into the applications.

6. CONCLUSION

Network packet analysis is becoming more and more important day by day as the number of devices joining networks increases. By analyzing the packages produced by smartphone applications, very important results can be achieved in many areas. But, as the number of devices joining the network has increased, the analysis process has become difficult. With machine learning, this analysis has become easier while making predictions for the future. In addition, due to the fact that machine learning is a popular field, network packet analysis and estimation on these packets are constantly developing thanks to new methods emerging almost every day.

However, analysis of network packets with machine learning is a relatively new topic. ince the proliferation of smartphones is a new process, the analysis of packages created by smartphones is an even newer subject. The analysis of these packages with machine learning has become an indispensable field in subjects such as category classification and user behavior analysis, which are important for this field.

Smartphones have become very common nowadays and the analysis of the network packet data they produce is even more important. Therefore, in this thesis, a total of four algorithms Random Forest, Catboost, LightGBM and Tabnet were selected for category estimation by analyzing the packet data of smartphones, and each algorithm was studied and these algorithms were compared with each other.

REFERENCES

Books

Boschetti A.,Massaron L.. (2018), *Python Data science Essentials – Third Edition*

Other Publications

Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, Valerio Persico and Antonio Pescapé (2019). *MIRAGE: Mobile-app Traffic Capture and Ground-truth Creation.* IEEE International Conference on Computing, Communication and Security (ICCCS).

Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri and Antonio Pescapé (2019). *Mobile Encrypted Traffic Classification Using Deep Learning: Experimental Evaluation, Lessons Learned, and Challenges*. IEEE Transactions on Network and Service Management

Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. 2018. "CatBoost: unbiased boosting with categorical features". *In Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18)*, 6639–6649.

Valent´ın Carela-Espanol, Pere Barlet-Ros, Marc Sol´e-Sim´o, Alberto Dainotti, Walter de Donato, and Antonio Pescape (2010). *K-Dimensional Trees for Continuous Traffic Classification. In Traffic Monitoring and Analysis.*

Thuy T.T. Nguyen and Grenville Armitage (2008). *A Survey of Techniques for Internet Traffic Classification using Machine Learning.* IEEE Communications Surveys & Tutorials.

Alberto Dainotti, Francesco Gargiulo, Ludmila I. Kuncheva, Antonio Pescap`e and Carlo Sansone (2010). *Identification of traffic flows hiding behind TCP port 80.* Proceedings of IEEE International Conference on Communications, ICC 2010, Cape Town, South Africa.

A. Al-Subaihin, F. Sarro, S. Black, L. Capra, M. Harman, Y. Jia and Y. Zhang (2016). Clustering Mobile Apps Based on Mined Textual Features. ESEM '16: Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement

Gaofeng He, Bingfeng Xu and Haiting Zhu (2017). *Identifying Mobile Applications for Encrypted Network Traffic.* Fifth International Conference on Advanced Cloud and Big Data

Yang Liu, Shuzhuang Zhangz, Bo Din, Xiaoqing Li and Yipeng Wang (2018). *Cascade Forest Approach to Application Classification of Mobile Traces.* IEEE Wireless Communications and Networking Conference

Vincent F. Taylor, Riccardo Spolaor, Mauro Conti and Ivan Martinovic (2018). *Robust Smartphone App Identification via Encrypted Network Traffic Analysis.* IEEE Transactions on Information Forensics and Security

Y. Zhang, X. Chen, L. Jin, X. Wang and D. Guo, "Network Intrusion Detection: Based on Deep Hierarchical Network and Original Flow Data," in IEEE Access, vol. 7, pp. 37004-37016, 2019.

Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri and Antonio Pescapé (2017). *Traffic Classification of Mobile Apps through Multi-classification.* IEEE Conference and Exhibition on Global Telecommunications.

M. Mellia, A. Pescapé and L. Salgarelli, "Traffic classification and its applications to modern networks", *Computer Networks*, vol. 53, no. 6, pp. 759-760, April 2009.

Pedregosa *et al.* (2011). *Scikit-learn: Machine Learning in Python*, JMLR 12, pp. 2825-2830.

RFC: 791 Internet Protocol (1981). Retrieved from https://datatracker.ietf.org/doc/html/rfc791

Jagandeep Singh (2020). Random Forest: Pros and Cons. Retrieved from https://medium.datadriveninvestor.com/random-forest-pros-and-cons-c1c42fb64f04

Subham Surana (2020) What is Light GBM? Advantages & Disadvantages? Light GBM vs XGBoost? Retrieved from https://www.kaggle.com/general/264327

Shreyanshi Singh (2021). LightGBM (Light Gradient Boosting Machine). Retrieved from

https://www.geeksforgeeks.org/lightgbm-light-gradient-boosting-machine/

LightGBM documentation (2021). Retrieved from
https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.LGBMClassifier.html

Tal Peretz (2018) Mastering The New Generation of Gradient Boosting Retrieved from
https://towardsdatascience.com/https-medium-com-talperetz24-mastering-the-new generation-of-gradient-boosting-db04062a7ea2

Abhishek Mishra (2020) CatBoost , The Spirit of a True Racer Retrieved from
https://datascience.foundation/datatalk/catboost-the-spirit-of-a-true-racer

Adam Shafi (2021) TabNet: The End of Gradient Boosting?. Retrieved from
https://towardsdatascience.com/tabnet-e1b979907694

Tanul Singh (2020) Achieving SOTA Results with Tabnet. Retrieved from
https://www.kaggle.com/tanulsingh077/achieving-sota-results-with-tabnet

42matters.com (2021). IAB Category API for Android Apps. Retrieved from
https://42matters.com/docs/app-market-data/android/apps/category