TikTok commercial video advertisements detect! Who can make the accuracy 0.99?

Capstone Project 2: Milestone Report

Introduction

I am Trying to predict the label and accuracy 0.99

Why is it a useful question to answer and details about the client?TikTok is the leading destination for short-form mobile video. I am interested in learning about TikTok ads and how they work . This project enables me to use different models to make the prediction and find the accuracy of 0.99.

Descriptive Data Analysis

The dataset for this project is downloaded as a csv file from the Kaggle website.
Kaggle-https://www.kaggle.com/lilhxr/commercial-vedio-data
This dataset contains 29685 rows × 232 columns
The data provides information for 129685 videos,which are stored in the commercial*vedio*data.csv file, where labels are labels.

The columns are as follows:-

**<bound method DataFrame.info of Unnamed: 0 1 2 3 4 5 6 **

0 0 123 1.316440 1.516003 5.605905 5.346760 0.013233

1 1 124 0.966079 0.546420 4.046537 3.190973 0.008338

2 2 109 2.035407 0.571643 9.551406 5.803685 0.015189

3 3 86 3.206008 0.786326 10.092709 2.693058 0.013962

4 4 76 3.135861 0.896346 10.348035 2.651010 0.020914

... ...

129680 129680 41 0.558875 0.247396 2.279039 0.798749 0.023244

129681 129681 105 6.594097 3.418464 17.372873 8.571255 0.020726

129682 129682 100 1.299783 0.543502 5.095611 1.527174 0.018389

129683 129683 60 0.571542 0.487061 2.218786 2.174240 0.018031

129684 129684 25 1.949249 3.074548 11.921268 14.297340 0.017515

**7 8 9 ... 959 1002 1016 **

0 0.010729 0.091743 0.050768 ... 0.036017 0.006356 0.008475

1 0.011490 0.075504 0.065841 ... 0.117647 0.006303 0.055489

2 0.014294 0.094209 0.044991 ... 0.062500 0.004808 0.055489

3 0.011039 0.092042 0.043756 ... 0.046296 0.012346 0.055489

4 0.012061 0.108018 0.052617 ... 0.044993 0.003521 0.055489

... ...

129680 0.011390 0.057546 0.035940 ... 0.044993 0.040969 0.055489

129681 0.010193 0.085179 0.051084 ... 0.044993 0.040969 0.055489

129682 0.008418 0.104063 0.045564 ... 0.044993 0.023684 0.055489

129683 0.007570 0.122708 0.050065 ... 0.044993 0.022727 0.055489

129684 0.009876 0.090000 0.051796 ... 0.044993 0.040969 0.055489

1028 1048 1112 1119 4124 4125 labels

0 0.003688 0.002119 0.006209 0.036149 0.422334 0.663918 1

1 0.003688 0.008403 0.006209 0.036149 0.332664 0.766184 1

2 0.003688 0.009615 0.006209 0.036149 0.346674 0.225022 1

3 0.003688 0.012346 0.003086 0.036149 0.993323 0.840083 1

4 0.003688 0.045775 0.007042 0.036149 0.341520 0.710470 1

... ...

129680 0.003688 0.035440 0.006209 0.036149 0.190005 0.946195 1

129681 0.003688 0.035440 0.006209 0.036149 0.166974 0.728715 1

129682 0.003688 0.005263 0.006209 0.036149 0.928798 0.437874 1

129683 0.003688 0.035440 0.006209 0.036149 0.745411 0.260724 1

129684 0.003688 0.035440 0.006209 0.036149 0.630468 0.345910 1

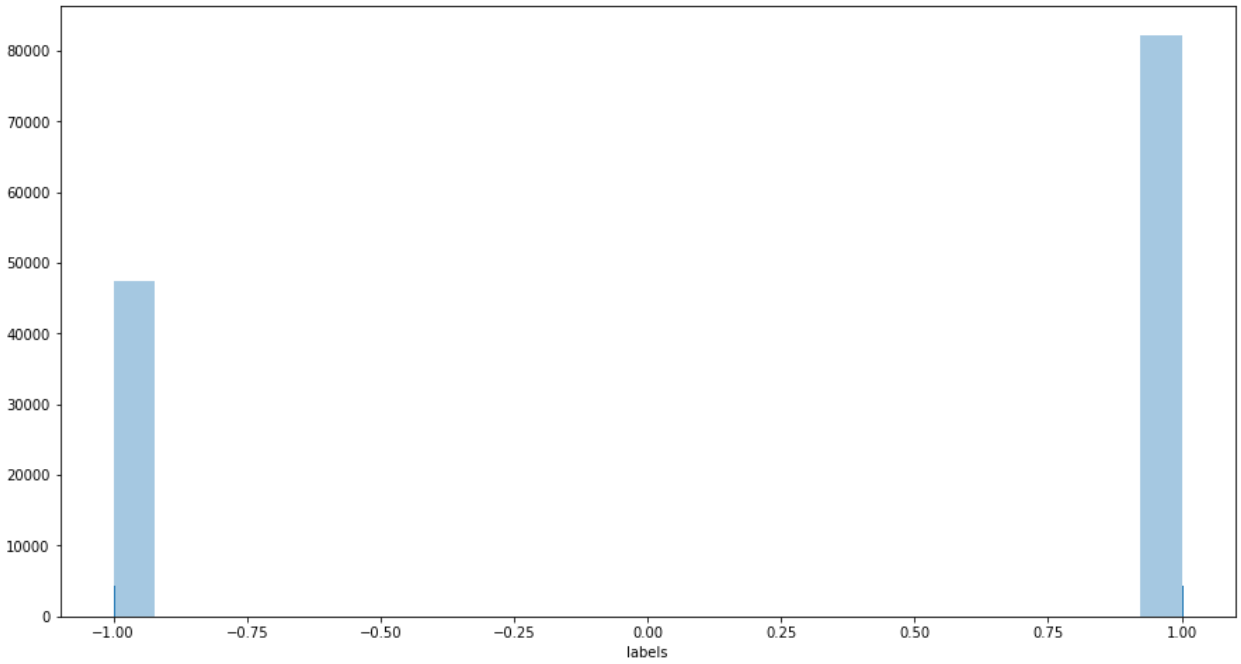
[129685 rows x 232 columns]>

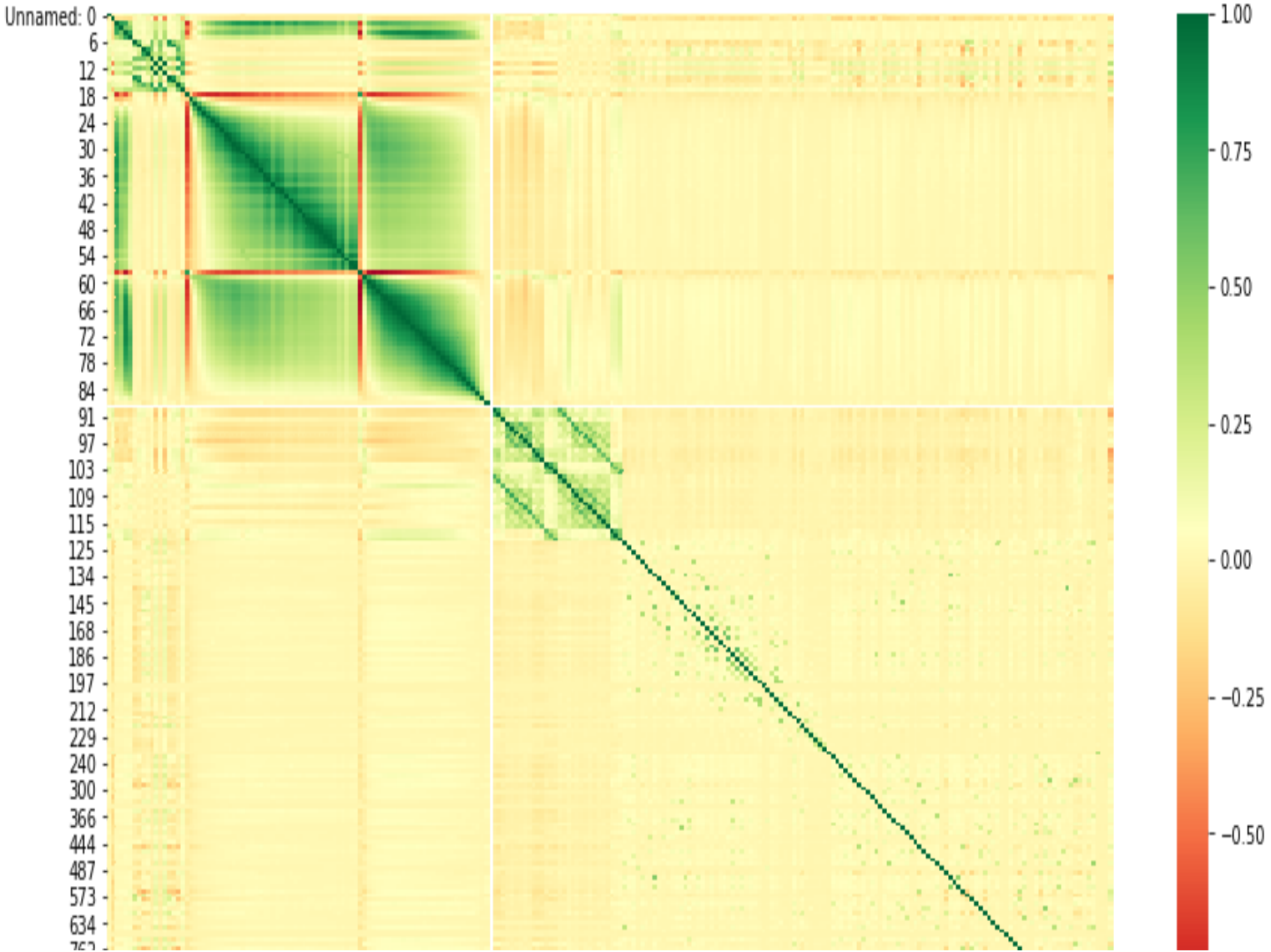
Data Wrangling and Cleaning

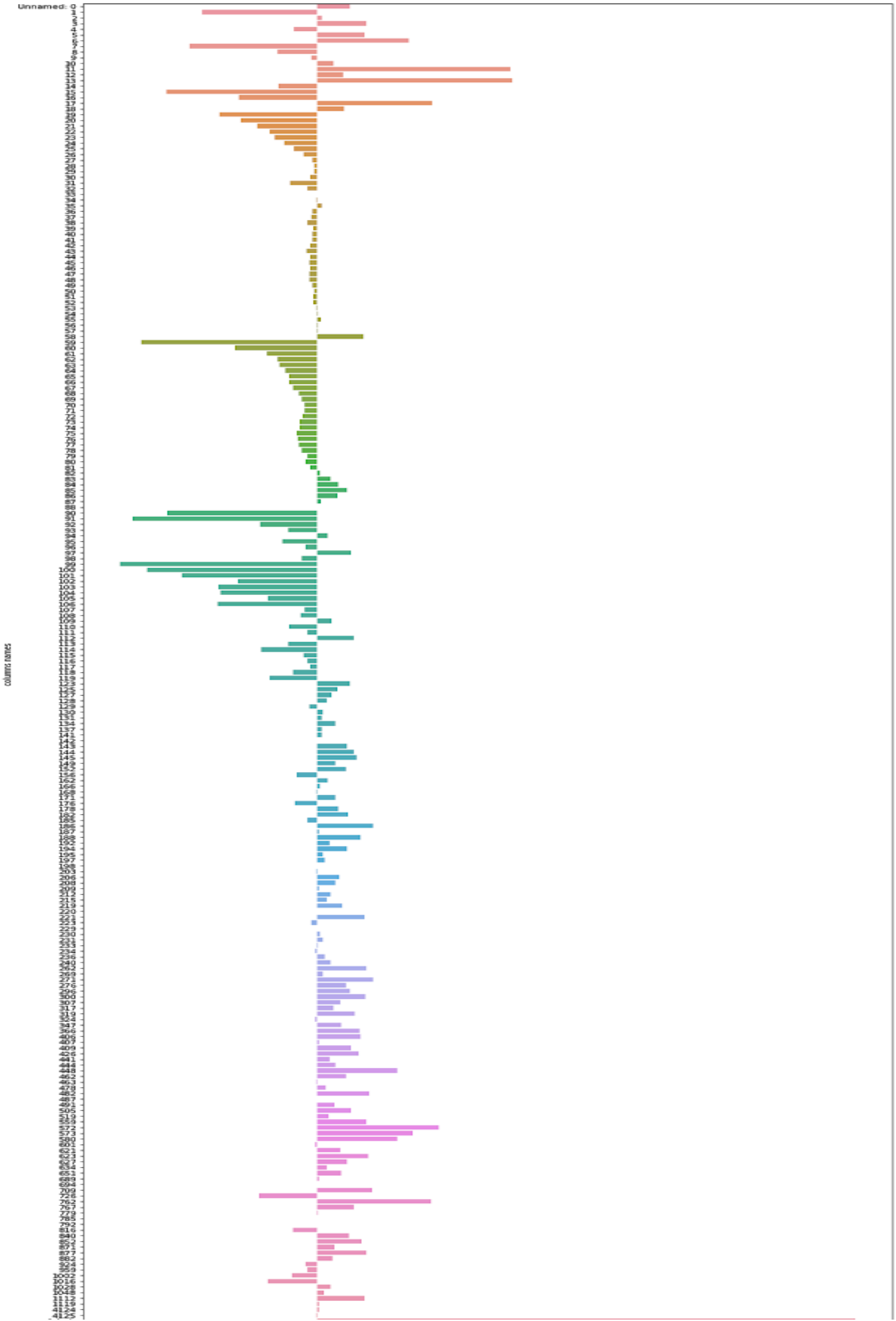
The goal of the wrangling step is to transform the data from csv format into a pandas dataframe. When I took a closer look at the data, I noticed that several features have Null values. I performed **data wrangling** using df.isnull() a method that returns True if there is any NaN value in the DataFrame and returns False if there is not even a single NaN entry in the DataFrame. I noticed that there was a mixture of True or False in the data then I used df = df.fillna(df.mean()) to fill all the null values with mean to make sure all values that are NaN are True.

Exploratory data analysis summary (visualization and inferential statistics)

I used the describe() method to get a basic statistical summary about the data such as: Percentile, Mean /Average, Median, Variance, Std.The Data visualization is presented by a bar plot and correlation plot. The bar plot makes it easy to compare the data between the labels and the other variables in the data. From the data When the label is 1.0 there are 88000 zeroes. When the label is -1 there are 45000. We used the Correlation to assess a possible linear association between the labeled and the other variables in the data.


80000
70000
60000
50000
40000
30000
20000
10000
0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
labels


Unnamed: 0
6
12
18
24
30
36
42
48
54
60
66
72
78
84
91
97
103
109
115
125
134
145
168
186
197
212
229
240
300
366
444
487
573
634
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50


Unnamed: 0
columns names

Summary of Findings

For the prediction and to compare the model for 0.99 accuracies, I used Random Forest, Neural Network, and Keras.

Random Forests

1)I started by importing the datasets library from scikit-learn
2)Second, I separated the columns into dependent and independent variables (or features and labels). Then I split those variables into a training and test set.
3)After splitting, I train the model on the training set and perform predictions on the test set.
4)After training, check the accuracy using actual and predicted values

- clf=RandomForestClassifier()
- clf.fit(X_train, y_train)
- pred = clf.predict(X_test)
- accuracy_tst = accuracy_score(y_test, pred)
- print("testing accuracy", accuracy_tst)

Testing accuracy 0.9571268843736747

Pros:

- Used for regression and classification problems, making it a diverse model.
- Prevents overfitting of data.
- Fast to train with test data.

Cons:

- Slow in creating predictions once a model is made.
- Must beware of outliers and holes in the data.

Data Normalization/preprocessing

To standardize the digits data, I used the scale() method

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
```

```
scaler.fit(X_train)
X_train=scaler.transform(X_train)
X_test=scaler.transform(X_test)
```

By scaling the data, I shifted the distribution of each attribute to have a mean of zero and a standard deviation of one (unit variance).

Splitting The Data Into Training And Test Sets

To assess the model's performance, I divided the data set into two parts: a training set and a test set. The first is used to train the system, while the second is used to evaluate the learned or trained system.

```
# Import `train_test_split` from `sklearn.model_selection`
from sklearn.model_selection import train_test_split

# Specify the data
X=df.drop("labels",axis=1)

# Specify the target labels and flatten the array
y=df["labels"]

# Split the data up in train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

The next step is to predict the labels of the test set:

```
    # Specify the data
    X=df.drop("labels",axis=1)

    # Specify the target labels and flatten the array
    y=df["labels"]

    # Split the data up in train and test sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,
    random_state=42
```

Neural Network

I used Class **MLPClassifier** Which implements a multi-layer perceptron (MLP) algorithm that trains using Backpropagation.

```
    clf = MLPClassifier(alpha=1e-3,max_iter=500, hidden_layer_sizes =
    (16,32,64,32,16))
    clf.fit(X_train, y_train)
```

```
pred = clf.predict(X_test)
accuracy_tst = accuracy_score(y_test, pred)
print("testing accuracy", accuracy_tst)

testing accuracy 0.9220418706866639
```

Keras Sequential

Besides using the neural_network/MLPClassifier to build the multi-layer perceptron, I have also used the Keras Sequential model. In the first layer, the activation argument takes the value relu. The input_shape has been defined as follow**:**

```
the model takes as input arrays of shape  (16, input_dim=231,
activation='relu',kernel_initializer='normal'))

model.add(Dense(32, activation='relu',kernel_initializer='normal'))

model.add(Dense(64, activation='relu',kernel_initializer='normal'))

model.add(Dense(32, activation='relu',kernel_initializer='normal'))

model.add(Dense(16, activation='relu',kernel_initializer='normal'))

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

opt = keras.optimizers.Adam(learning_rate=0.0001)

model.compile(loss='binary_crossentropy', optimizer=opt,metrics=['accuracy'])

model.summary()

model.fit(x=X_train, y=y_train, validation_data=(X_test, y_test), epochs=500,
batch_size=128)
```

This model has different accuracies. The highest accuracy is 0.9514. In comparison to the other models, Random Forest provides the best accuracy to my prediction.