Twitter Gender Classification

Twitter Gender Classification

Machine learning techniques to predict the user's gender on Twitter text data

This project was produced by MSITM 2021-2022 Team 6:
Shreya Bakshi, Jonathan Cope, Preeti Gupta, Gowtami Khambhampati, Preety Pinghal, Sneha Reddy, Danqing Wang

Introduction:

The user profile is a persona for a product or service. Gender profiling of unstructured data has several applications in areas such as marketing, advertising, recommendation systems, etc. We can segment the data and understand what drives users, how to attract more users, and how users interact with the service. Now we are going to use the profile information of users on Twitter to predict their gender.

The Problem Statement:

Given this dataset of Twitter conversations, how well can our proposed model predict the gender of the user based on linguistic cues from textual Twitter data.

Environment:

For this project, we Installed classifiers, necessary packages and imported all the libraries required for visualizations, modeling, automation, vectorization, etc.
We also pip installed pycaret and lgbm.

```
! pip install https://github.com/pandas-profiling/pandas-profiling/archive/master.zip
! pip install pycaret
! pip install lightgbm
```

```
import pandas as pd
import numpy as np
from IPython.display import display
from google.colab import files
!pwd
import os
import re
import regex
import nltk
from nltk.stem import PorterStemmer #Textual data cleaning
nltk.download('stopwords')
from nltk.corpus import stopwords
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams.update({'figure.max_open_warning': 0})
import pylab
from mpl_toolkits.mplot3d import Axes3D # 3D visualization
from matplotlib import pyplot
from mpl_toolkits.axes_grid1 import make_axes_locatable
from scipy import ndimage
from sklearn.feature_extraction import text
from sklearn import model_selection
from sklearn.naive_bayes import MultinomialNB # Naive Bayes model
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

import seaborn as sns
from pandas_profiling import ProfileReport # This library is great for seeing what is missing and what needs to be cleaned up
from bokeh.plotting import output_notebook, figure, show #graph
from bokeh.layouts import gridplot
from bokeh.models import ColumnDataSource

from collections import Counter
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from lightgbm import LGBMClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import ComplementNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.pipeline import Pipeline
from sklearn.neighbors import KNeighborsClassifier
```

Data Source:

The data has been extracted from Kaggle from the link below.

https://www.kaggle.com/crowdflower/twitter-user-gender-classification

Read the CSV file:

```
[3] twitter_data = files.upload()
    data = pd.read_csv("gender-classifier-DFE-791531.csv", encoding='latin-1')
```

```
Browse... gender-classifier-DFE-791531.csv
gender-classifier-DFE-791531.csv(text/csv) - 8176739 bytes, last modified: n/a - 100% done
Saving gender-classifier-DFE-791531.csv to gender-classifier-DFE-791531 (1).csv
```

Shape of the dataset:

```
data.shape

(20050, 26)
```

Dataset consists of 20050 rows and 26 columns. Out of 26 columns, we use 25 predictor variables and 1 target variable which is 'gender'.

Columns and their associated data types:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20050 entries, 0 to 20049
Data columns (total 26 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   _unit_id               20050 non-null  int64
 1   _golden                20050 non-null  bool
 2   _unit_state            20050 non-null  object
 3   _trusted_judgments     20050 non-null  int64
 4   _last_judgment_at      20000 non-null  object
 5   gender                 19953 non-null  object
 6   gender:confidence      20024 non-null  float64
 7   profile_yn             20050 non-null  object
 8   profile_yn:confidence  20050 non-null  float64
 9   created                20050 non-null  object
 10  description            16306 non-null  object
 11  fav_number             20050 non-null  int64
 12  gender_gold            50 non-null     object
 13  link_color             20050 non-null  object
 14  name                   20050 non-null  object
 15  profile_yn_gold        50 non-null     object
 16  profileimage           20050 non-null  object
 17  retweet_count          20050 non-null  int64
 18  sidebar_color          20050 non-null  object
 19  text                   20050 non-null  object
 20  tweet_coord            159 non-null    object
 21  tweet_count            20050 non-null  int64
 22  tweet_created          20050 non-null  object
 23  tweet_id               20050 non-null  float64
 24  tweet_location         12566 non-null  object
 25  user_timezone          12252 non-null  object
dtypes: bool(1), float64(3), int64(5), object(17)
memory usage: 3.8+ MB
```

Analyzing Data source:

Features of dataset:

```
[ ] data.keys()
```

```
Index(['_unit_id', '_golden', '_unit_state', '_trusted_judgments',
       '_last_judgment_at', 'gender', 'gender:confidence', 'profile_yn',
       'profile_yn:confidence', 'created', 'description', 'fav_number',
       'gender_gold', 'link_color', 'name', 'profile_yn_gold', 'profileimage',
       'retweet_count', 'sidebar_color', 'text', 'tweet_coord', 'tweet_count',
       'tweet_created', 'tweet_id', 'tweet_location', 'user_timezone'],
      dtype='object')
```

Exploring Target variable - 'Gender':

```
data['gender'].value_counts()
```

```
female     6700
male       6194
brand      5942
unknown    1117
Name: gender, dtype: int64
```

Null values in the dataset:

```
data.isnull().sum()
```

```
_unit_id                     0
_golden                      0
_unit_state                  0
_trusted_judgments           0
_last_judgment_at           50
gender                      97
gender:confidence           26
profile_yn                   0
profile_yn:confidence        0
created                      0
description               3744
fav_number                   0
gender_gold              20000
link_color                   0
name                         0
profile_yn_gold          20000
profileimage                 0
retweet_count                0
sidebar_color                0
text                         0
tweet_coord              19891
tweet_count                  0
tweet_created                0
tweet_id                     0
tweet_location            7484
user_timezone             7798
dtype: int64
```

Data Cleaning:

Data cleansing is a very crucial step in the overall data preparation process and it is the process of analyzing, identifying, and correcting messy, raw data.

We started our data cleaning by dropping unnecessary features

Drop unnecessary columns/features

```
data.drop (columns = ['_unit_id',
                      '_last_judgment_at',
                      'user_timezone',
                      'tweet_coord',
                      'tweet_created',
                      'tweet_id',
                      'tweet_location',
                      'profileimage',
                      'created',
                      'name'], inplace = True)

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20050 entries, 0 to 20049
Data columns (total 16 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   _golden               20050 non-null  bool
 1   _unit_state           20050 non-null  object
 2   _trusted_judgments    20050 non-null  int64
 3   gender                19953 non-null  object
 4   gender:confidence     20024 non-null  float64
 5   profile_yn            20050 non-null  object
 6   profile_yn:confidence 20050 non-null  float64
 7   description           16306 non-null  object
 8   fav_number            20050 non-null  int64
 9   gender_gold           50 non-null     object
 10  link_color            20050 non-null  object
 11  profile_yn_gold       50 non-null     object
 12  retweet_count         20050 non-null  int64
 13  sidebar_color         20050 non-null  object
 14  text                  20050 non-null  object
 15  tweet_count           20050 non-null  int64
dtypes: bool(1), float64(2), int64(4), object(9)
memory usage: 2.3+ MB
```

Getting rid of gender type - “unknown”:

```
unknown_items_idx = data[data['gender'] == 'unknown'].index
data.drop (index = unknown_items_idx, inplace = True)
data['gender'].value_counts()
```

```
female    6700
male      6194
brand     5942
Name: gender, dtype: int64
```

Getting rid of rows where column “profile_yn” is no:

```
[20] drop_items_idx = data[data['profile_yn'] == 'no'].index
     data.drop (index = drop_items_idx, inplace = True)
     data.drop (columns = ['profile_yn','profile_yn:confidence','profile_yn_gold'], inplace = True)
```

Removing Stop words and cleaning the Text:

Here we are using functions **preprocessor(), remove_dup_whitespace(), tokenizer_porter(), clean_tweet, has_nan** to clean, stem and tokenize the text

```python
stop = stopwords.words('english')
porter = PorterStemmer()

def preprocessor(text):             #Return a cleaned version of text, but keeping the emoticons
    text = re.sub('<[^>]*>', '', text) # Remove HTML markup
    text = re.sub('http.*', ' ', text) # Remove url tokens
    return text

def remove_dup_whitespace(text):    #This function removes duplicated whitespaces of a string
    return re.sub('\s{2,}', ' ',text)  #return text

def tokenizer_porter(text):       # This function tokenize and also perform stemming
    return [porter.stem(word) for word in text.lower().split()]

def clean_tweet(text):            #This function tokenizes whole tweet into tokens,clean it, remove stopwords and combine back as a tweet, this function com
    clean = ""
    tokens = tokenizer_porter(text)
    for token in tokens:
        if len(token)> 1:
            if token not in stop:
                clean += preprocessor(token) + " "
    #print(data)
    return clean
    return remove_dup_whitespace(clean)
def has_nan(X):
    '''
    Input: Dataframe
    This func check if the features of a DataFrame has missing values or not
    '''
    X_ = X.isnull()
    X_ = X_.add_suffix('_has_nan')
    return X_

has_nan_df = has_nan(data[['description']])
data = pd.concat([data, has_nan_df], axis=1)

data['description'].fillna("", inplace=True) # Fill NaN with empty string
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/series.py:4536: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  downcast=downcast,
```

```python
data[['text','description']] = data[['text','description']].applymap(clean_tweet)
```

Changing the text to lower characters:

```python
def clean1(review1):

    descrip = re.sub('[^a-zA-Z]', ' ', review1)

    review1 = review1.lower()

    return review1

data['text_Cleaned'] = pd.DataFrame(data['text'].apply(lambda y: clean1(y)))

data.head(5)
```

	_golden	_unit_state	_trusted_judgments	gender	gender:confidence	description	fav_number	gender_gold	link_color	retweet_count	sidebar_color	text	tweet_count	description_has_nan	text_Cleaned
0	False	finalized	3	male	1.0000	sing rhythm.	0	NaN	08C2C2	0	FFFFFF	robbi respond critic win eddi edward #worldtit...	110964	False	robbi respond critic win eddi edward #worldtit...
1	False	finalized	3	male	1.0000	i'm author novel fill famili drama romance.	68	NaN	0084B4	0	C0DEED	ûïit felt like friend wa live stori themû ...	7471	False	ûïit felt like friend wa live stori themû ...
2	False	finalized	3	male	0.6625	loui whine squeal	7696	NaN	ABB8C2	1	C0DEED	absolut ador loui start song hit hard feel good	5617	False	absolut ador loui start song hit hard feel good
3	False	finalized	3	male	1.0000	mobil guy. 49ers, shazam, google, kleiner perk...	202	NaN	0084B4	0	C0DEED	hi @jordanspieth look url use @ifttt?! typic s...	1693	False	hi @jordanspieth look url use @ifttt?! typic s...
4	False	finalized	3	female	1.0000	ricki wilson best frontman/kais chief best ban...	37318	NaN	3B94D9	0	0	watch neighbour sky+ catch neighbs!! xxx _ù÷ã_...	31462	False	watch neighbour sky+ catch neighbs!! xxx _ù÷ã_...

```python
data['text_Cleaned'] = data['text'].str.replace('[^A-Za-z0-9 ]+', '')
data['text_Cleaned']= data['text_Cleaned'].str.lower()
data.head()
```

	gender	gender:confidence	description	fav_number	link_color	retweet_count	sidebar_color	text	tweet_count	description_has_nan	text_Cleaned
0	male	1.0000	sing rhythm.	0	08C2C2	0	FFFFFF	robbi respond critic win eddi edward #worldtit...	110964	False	robbi respond critic win eddi edward worldtitl...
1	male	1.0000	i'm author novel fill famili drama romance.	68	0084B4	0	C0DEED	ûïit felt like friend wa live stori themû ...	7471	False	it felt like friend wa live stori them retir...
2	male	0.6625	loui whine squeal	7696	ABB8C2	1	C0DEED	absolut ador loui start song hit hard feel good	5617	False	absolut ador loui start song hit hard feel good
3	male	1.0000	mobil guy. 49ers, shazam, google, kleiner perk...	202	0084B4	0	C0DEED	hi @jordanspieth look url use @ifttt?! typic s...	1693	False	hi jordanspieth look url use ifttt typic see a...
4	female	1.0000	ricki wilson best frontman/kais chief best ban...	37318	3B94D9	0	0	watch neighbour sky+ catch neighbs!! xxx _ù÷ã_...	31462	False	watch neighbour sky catch neighbs xxx xxx

We have removed the column description_has_nan as it has no significance in predicting the gender which is our goal here.

```
data.drop(['text', 'description_has_nan'],axis=1,inplace=True)
data.head()
```

	gender	gender:confidence	description	fav_number	link_color	retweet_count	sidebar_color	tweet_count	text_Cleaned
0	male	1.0000	sing rhythm.	0	08C2C2	0	FFFFFF	110964	robbi respond critic win eddi edward worldtitl...
1	male	1.0000	i'm author novel fill famili drama romance.	68	0084B4	0	C0DEED	7471	it felt like friend wa live stori them retir...
2	male	0.6625	loui whine squeal	7696	ABB8C2	1	C0DEED	5617	absolut ador loui start song hit hard feel good
3	male	1.0000	mobil guy. 49ers, shazam, google, kleiner perk...	202	0084B4	0	C0DEED	1693	hi jordanspieth look url use ifttt typic see a...
4	female	1.0000	ricki wilson best frontman/kais chief best ban...	37318	3B94D9	0	0	31462	watch neighbour sky catch neighbs xxx xxx

PyCaret Machine Learning

PyCaret is an automated tool in Python that allows (with time) to create insights on what kind of data is being collected (categorical, numerical, boolean, etc.), and give the user a chance to make adjustments and tweaks to the auto-generated assumptions. The tool will then compile a variety of models using default hyperparameters to provide results such as accuracy and AUC. The tool can be further used to boost/ensemble/predict.

On the cleaned data without much feature engineering, we were able to find that out of models (Light Gradient Boosting, SVM, Random Forest, Logistic Regression, Ridge, Naive Bayes, Linear Discriminant Analysis, Extra Trees, Gradient Boosting, Quadratic Discriminant Analysis, Ada Boost, Decision Tree, K Neighbors, Dummy) and it tells us here the accuracy was 58%.

```
compare_models(sort = 'Accuracy', fold = 5)
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
lightgbm	Light Gradient Boosting Machine	0.5808	0.7675	0.5788	0.5929	0.5816	0.3674	0.3710	1.956
svm	SVM - Linear Kernel	0.5701	0.0000	0.5685	0.5769	0.5691	0.3517	0.3548	3.112
rf	Random Forest Classifier	0.5675	0.7510	0.5680	0.5673	0.5665	0.3495	0.3502	15.542
lr	Logistic Regression	0.5661	0.7513	0.5655	0.5714	0.5669	0.3465	0.3477	9.128
ridge	Ridge Classifier	0.5659	0.0000	0.5647	0.5756	0.5675	0.3457	0.3477	0.622
nb	Naive Bayes	0.5655	0.7326	0.5628	0.5765	0.5647	0.3440	0.3481	0.256
lda	Linear Discriminant Analysis	0.5625	0.7463	0.5610	0.5814	0.5663	0.3401	0.3429	5.518
et	Extra Trees Classifier	0.5612	0.7424	0.5620	0.5593	0.5594	0.3405	0.3411	20.846
gbc	Gradient Boosting Classifier	0.5481	0.7451	0.5392	0.6063	0.5360	0.3119	0.3409	37.750
qda	Quadratic Discriminant Analysis	0.5295	0.6996	0.5257	0.5821	0.5201	0.2887	0.3117	5.498
ada	Ada Boost Classifier	0.5160	0.6957	0.5041	0.5993	0.4667	0.2596	0.3219	3.946
dt	Decision Tree Classifier	0.5118	0.6424	0.5125	0.5097	0.5101	0.2661	0.2665	4.990
knn	K Neighbors Classifier	0.4630	0.6423	0.4599	0.4796	0.4635	0.1888	0.1917	78.654
dummy	Dummy Classifier	0.3562	0.5000	0.3333	0.1269	0.1871	0.0000	0.0000	0.064

```
LGBMClassifier(boosting_type='gbdt', class_weight=None, colsample_bytree=1.0,
               importance_type='split', learning_rate=0.1, max_depth=-1,
               min_child_samples=20, min_child_weight=0.001, min_split_gain=0.0,
               n_estimators=100, n_jobs=-1, num_leaves=31, objective=None,
               random_state=101, reg_alpha=0.0, reg_lambda=0.0, silent='warn',
               subsample=1.0, subsample_for_bin=200000, subsample_freq=0)
```

Feature Engineering:

To get the most out of our PyCaret Auto Modeling, we had to perform a few tasks of Feature Engineering. For example, we began by reindexing the data (since it would skip numbers, which would affect adding back the Y label after one-hot-encoding). Then, we pursued one hot encoding, starting with 500 columns for the “description” column and 500 for the “text_Cleaned” column. We would rename the second set of columns “500”-”999” so that they wouldn't clash with the 0-499 set.

```
data_copy = data.copy().reset_index().drop('index', axis=1)

cv = CountVectorizer(max_features = 500)
pyc = cv.fit_transform(data_copy['description']).toarray()
pyc1=cv.fit_transform(data_copy['text_Cleaned']).toarray()
```

```
P1=pd.DataFrame(pyc)
P2=pd.DataFrame(pyc1)
# We have to rename the second set of columns because two sets of the same names will crash
P2.columns = [x for x in range(500, 1000)]
```

Since PyCaret would not mix the string ‘brand’ in well, with the gender integers (0, 1), we translated ‘brand’ as a ‘2.’ We also converted the datatypes for the table as “int8” so that the computation would run faster than the default int64.

```
[ ] # we shouldn't mix ints and string labels together so lets fix that
    for gen in data_copy['gender']:
      if gen =='brand':
         data_copy['gender'].replace({'brand':'2'},inplace=True)

    data_copy['gender']
```

```
0        1
1        1
2        1
3        1
4        0
        ..
18831    0
18832    1
18833    1
18834    0
18835    0
Name: gender, Length: 18836, dtype: object
```

```
[ ] train_for_pycaret=pd.concat([P2,P1],join='outer',axis=1)
    train_for_pycaret['gender'] = data_copy['gender']
    train_for_pycaret=train_for_pycaret.fillna(0)
    train_for_pycaret = train_for_pycaret.astype('int8')
    train_for_pycaret.columns = [str(x) for x in train_for_pycaret.columns]
    train_for_pycaret
```

We then noticed that if we improve our one-hot-encoding vectorizer from 500 to 1500, the accuracy improves from 56 to 63.14% with Extra Trees Classifier

```
cv = CountVectorizer(max_features = 1500)
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
et	Extra Trees Classifier	0.6314	0.7973	0.6371	0.6286	0.6282	0.4405	0.4420	33.950
rf	Random Forest Classifier	0.6304	0.7977	0.6325	0.6328	0.6275	0.4361	0.4396	20.202
lr	Logistic Regression	0.6303	0.7949	0.6339	0.6343	0.6310	0.4372	0.4381	21.052
lightgbm	Light Gradient Boosting Machine	0.6247	0.7998	0.6247	0.6333	0.6243	0.4262	0.4297	2.486

After this, we improved our accuracy score further by including the Link Color (categorical) data, along with the Favorite Number (numerical) data.

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
lightgbm	Light Gradient Boosting Machine	0.6562	0.8289	0.6617	0.6558	0.6553	0.4782	0.4788	2.640
rf	Random Forest Classifier	0.6521	0.8225	0.6537	0.6555	0.6497	0.4693	0.4729	18.362
et	Extra Trees Classifier	0.6454	0.8129	0.6480	0.6479	0.6436	0.4596	0.4622	32.608
gbc	Gradient Boosting Classifier	0.6273	0.8068	0.6291	0.6268	0.6243	0.4327	0.4349	94.150

Finally, we decided to go a step further with our Link Color data (which was previously in HEX format (ex. #AAFF00). We converted this data into RGB Values (Red, Green, Blue values of 0-255 inclusive). Once we had these values in a usable format, we converted them into Color names using PIL's ImageColor library. If the name included "light" or "dark" substrings, we would remove that portion of the name. That way we would have multiple values using the standard labels "red," "blue," "gold," etc. regardless of whether they were light reds, dark blues, etc. This also made the data more categorical, and we also included the favorite color column as categorical data. **This brought our accuracy score to 65.99%**.

Based off of fiatjaf's comment and modified: https://stackoverflow.com/questions/9694165/convert-rgb-color-to-english-color-name-like-green-with-python

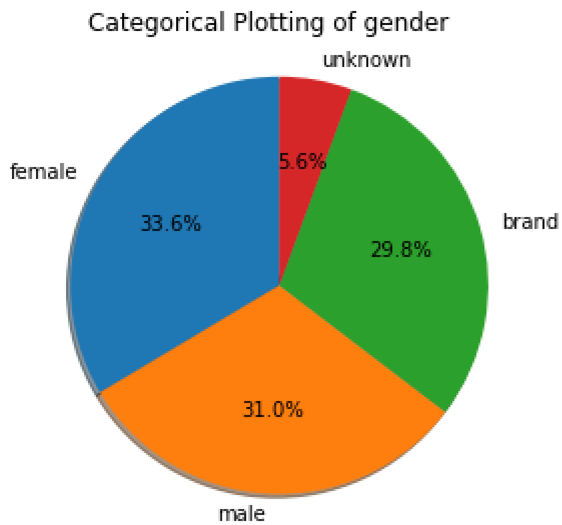
```
[ ] import matplotlib.colors as mc
    mycss4list = mc.CSS4_COLORS

    # For HEX to RGB
    from PIL import ImageColor

    def getColorName(hex_input):
      min_colors = {}
      for name, hex in mycss4list.items():
        r, g, b = ImageColor.getcolor(hex, "RGB")
        r_input, g_input, b_input = ImageColor.getcolor(hex_input, "RGB")
        rd = (r - r_input) ** 2
        gd = (g - g_input) ** 2
        bd = (b - b_input) ** 2
        min_colors[(rd + gd + bd)] = name
      #print(min_colors)
      colorName = min_colors[min(min_colors.keys())]
      colorName = colorName.replace("dark", "")
      colorName = colorName.replace("light", "")
      return colorName
```

```
[ ] print(getColorName("#990003"))

    red
```

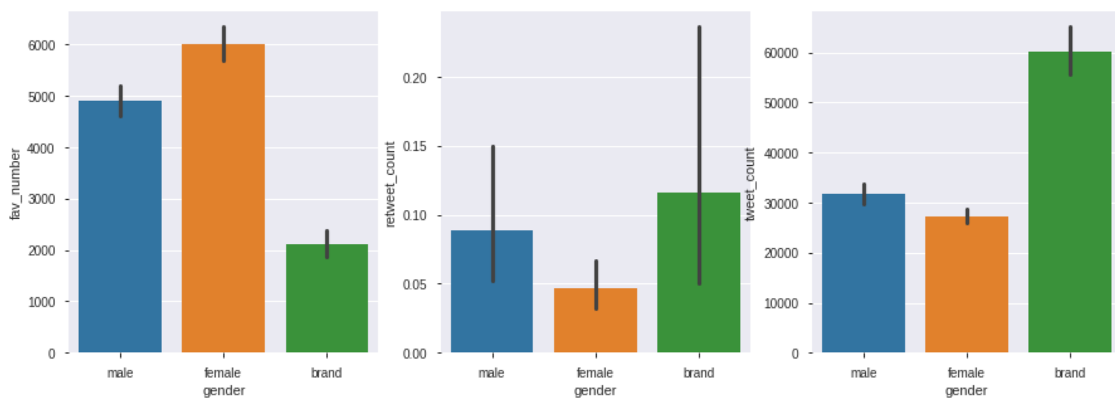
	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
lr	Logistic Regression	0.6599	0.8211	0.6638	0.6636	0.6610	0.4829	0.4834	49.698
svm	SVM - Linear Kernel	0.6369	0.0000	0.6413	0.6420	0.6386	0.4484	0.4490	10.040

Visualizing the Data:

Gender distribution

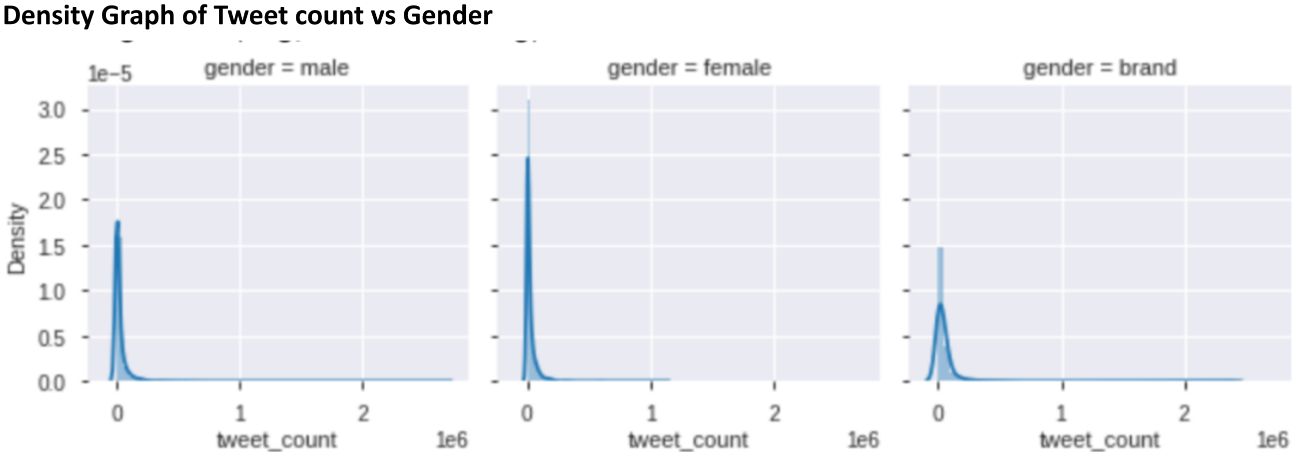


Subplots for fav_number, retweet_count, tweet_count for all ‘gender’ types:



From the figure above we notice that the retweet_count and tweet_count for brands are higher when compared to others.

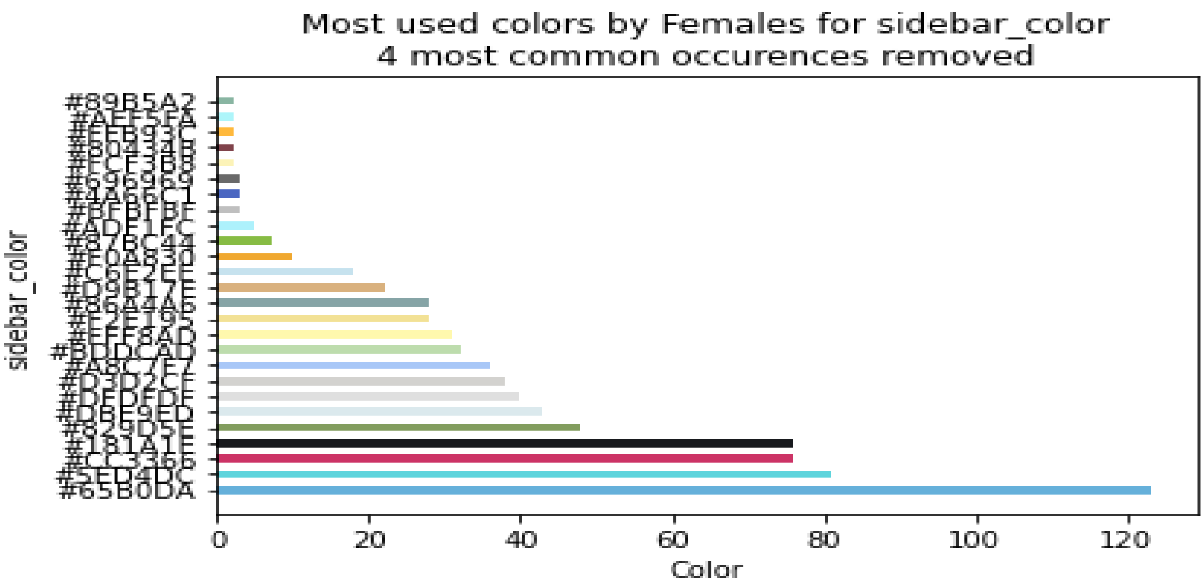
Density Graph of Tweet count vs Gender

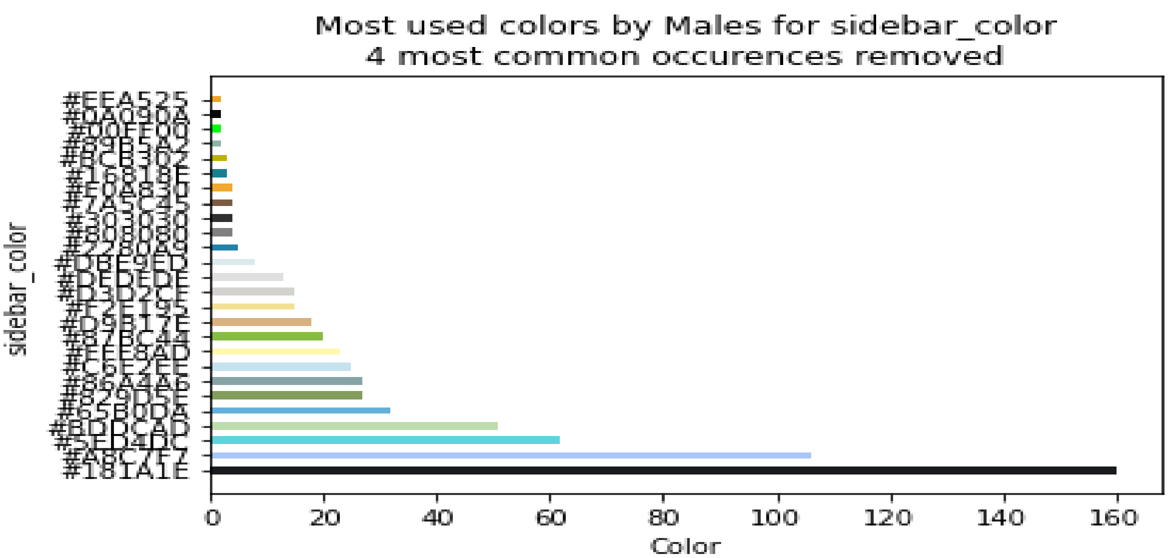


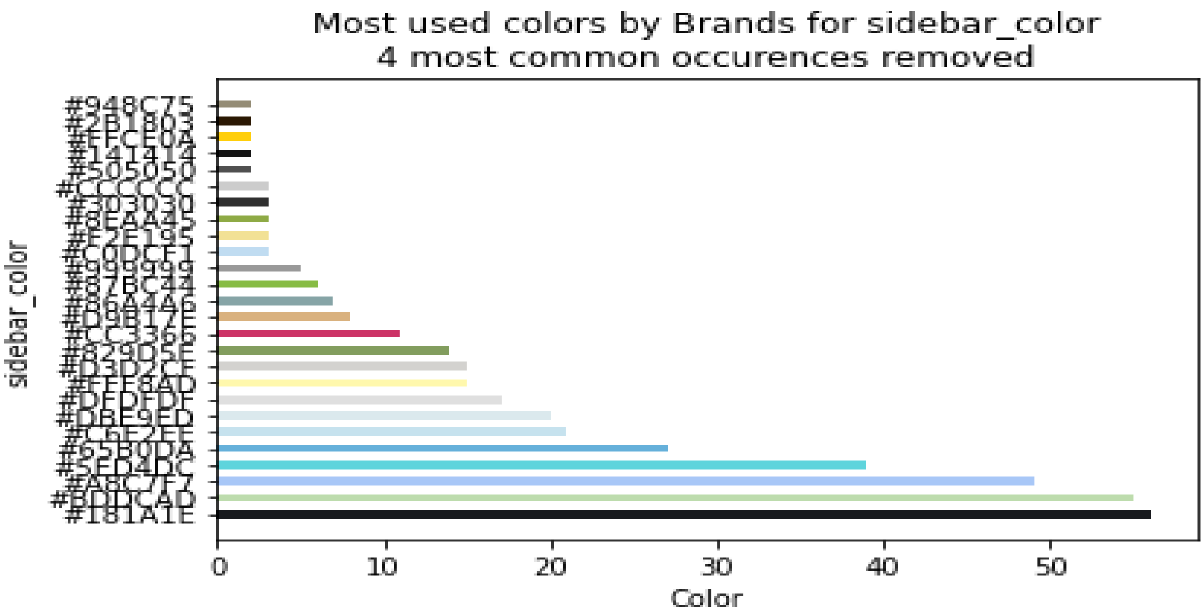
The above image shows the density for tweet count for genders male, female, and brand. The female gender has a high tweet count density in the dataset.

Visualizing color features

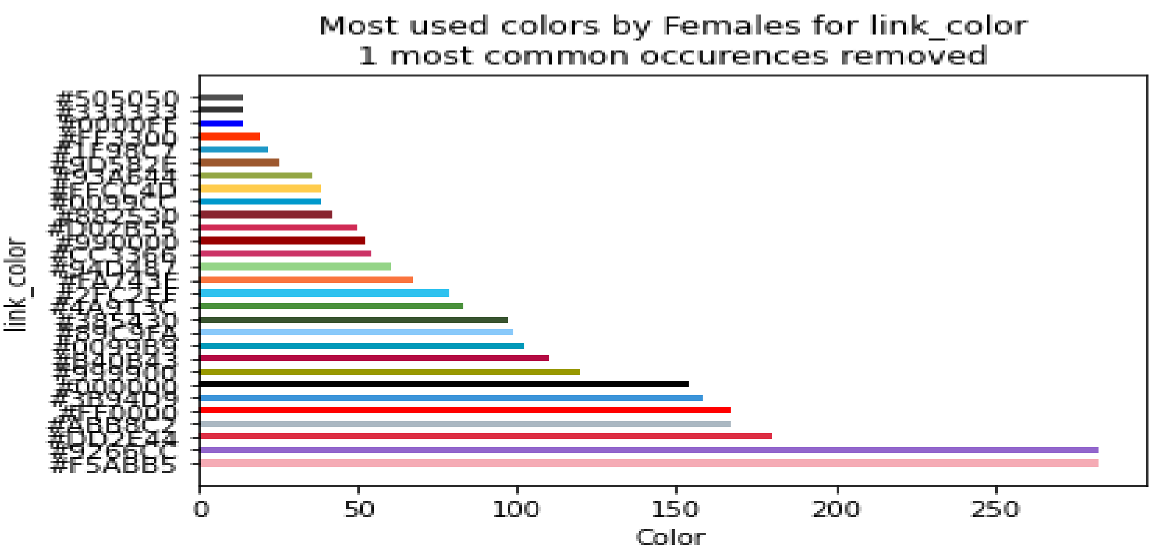
link_color and sidebar_color are two features that we are interested in to see if they can be used to predict genders . Like what kind of color women are preferring compared to men?



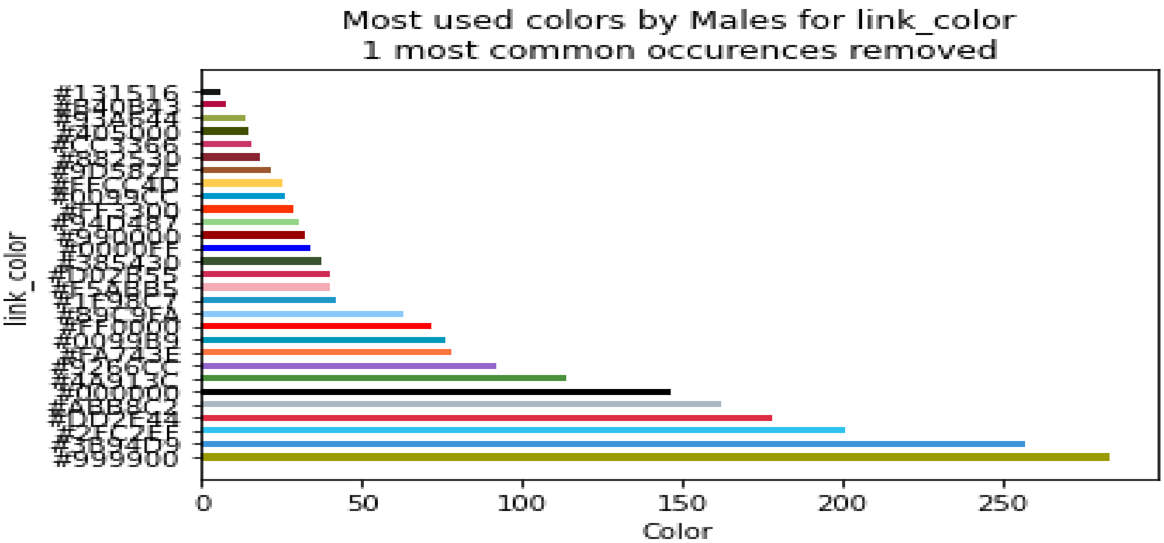


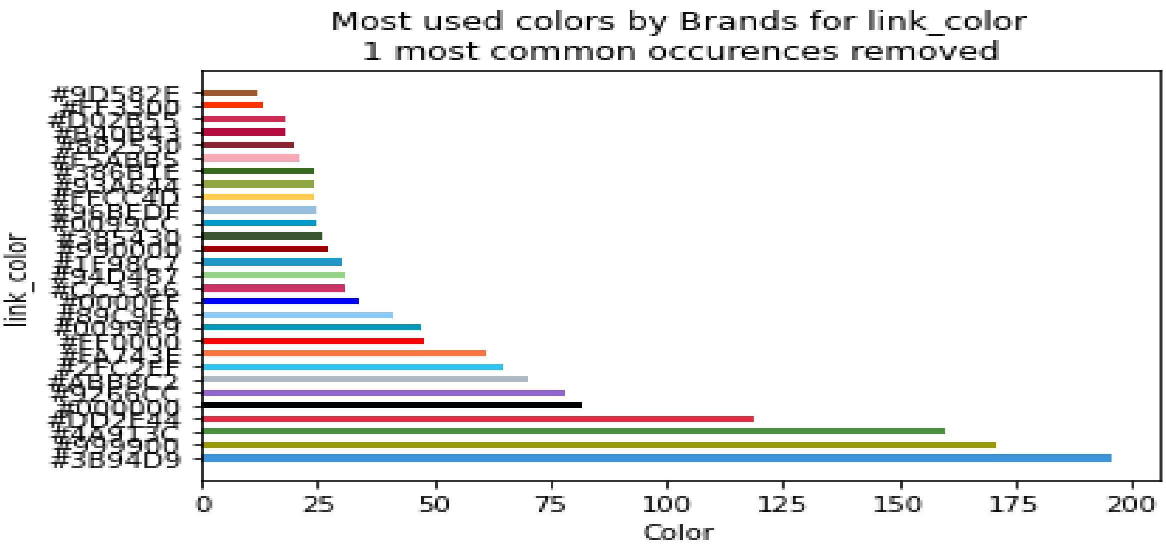


Most used colors by Brands for sidebar_color
4 most common occurences removed
sidebar_color
0
10
20
30
40
50
Color



Most used colors by Females for link_color
1 most common occurences removed
link_color
0
50
100
150
200
250
Color



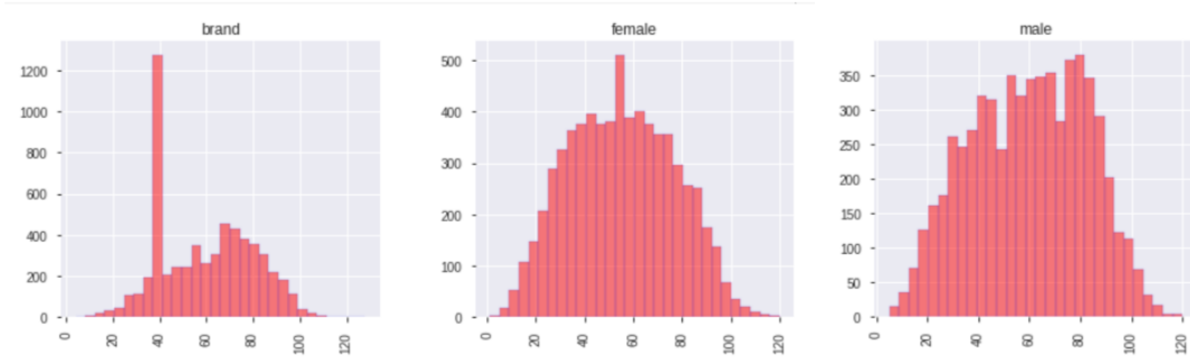


There are more variations in link_color than sidebar_color so people are changing link_color more than sidebar color.

For females pink & purple seems to be the most popular color for link_color whereas males prefer blue , shades of blue and green.

Choice of link_color overlaps more between brands and males than between brands and females.

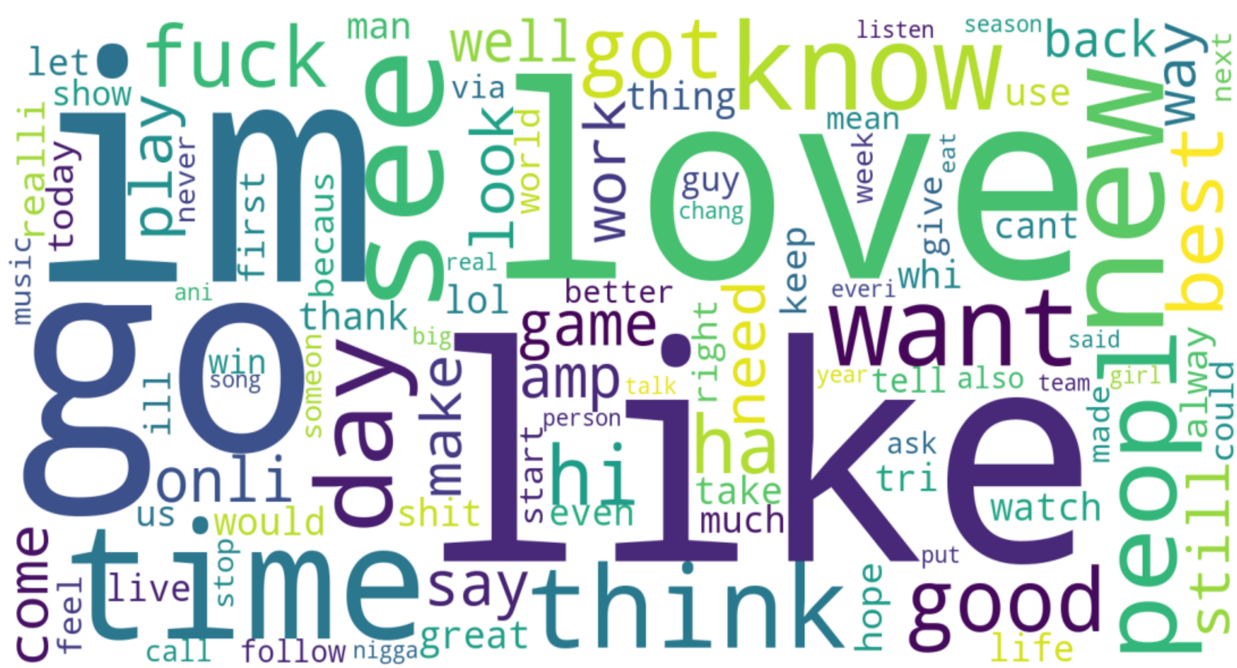
Text length vs Gender



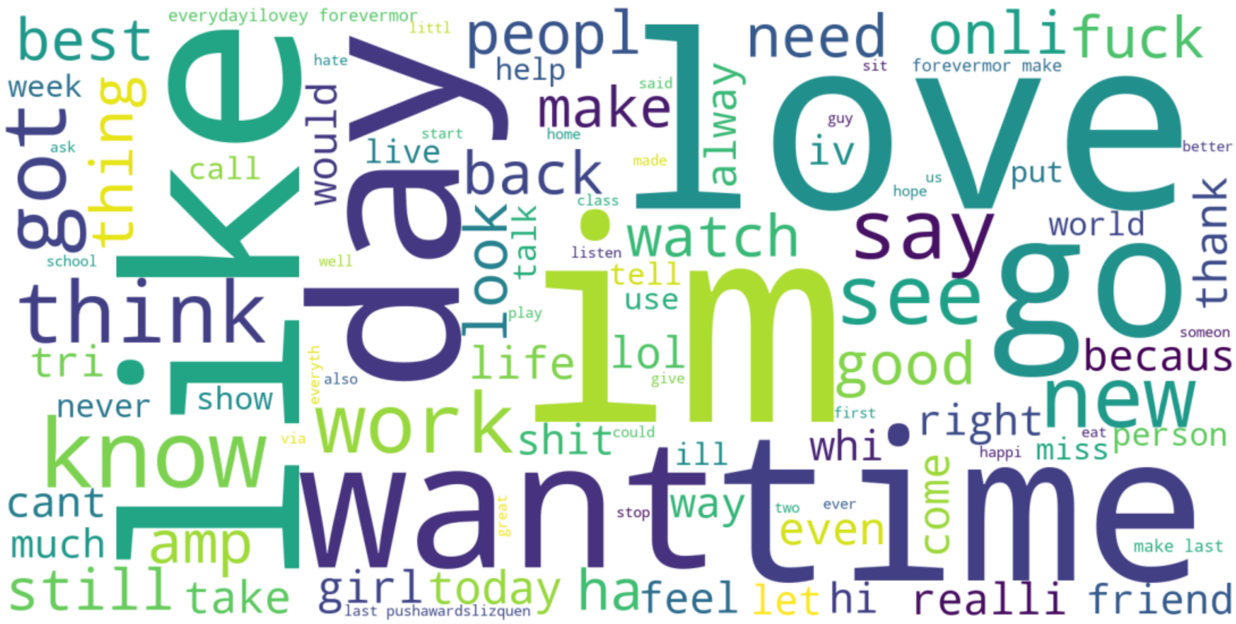
The above image shows a comparison of text lengths and how they differ among the gender and brand. We can see data follows a normal distribution trend here.

<u>Word cloud of “text” column based on gender:</u>

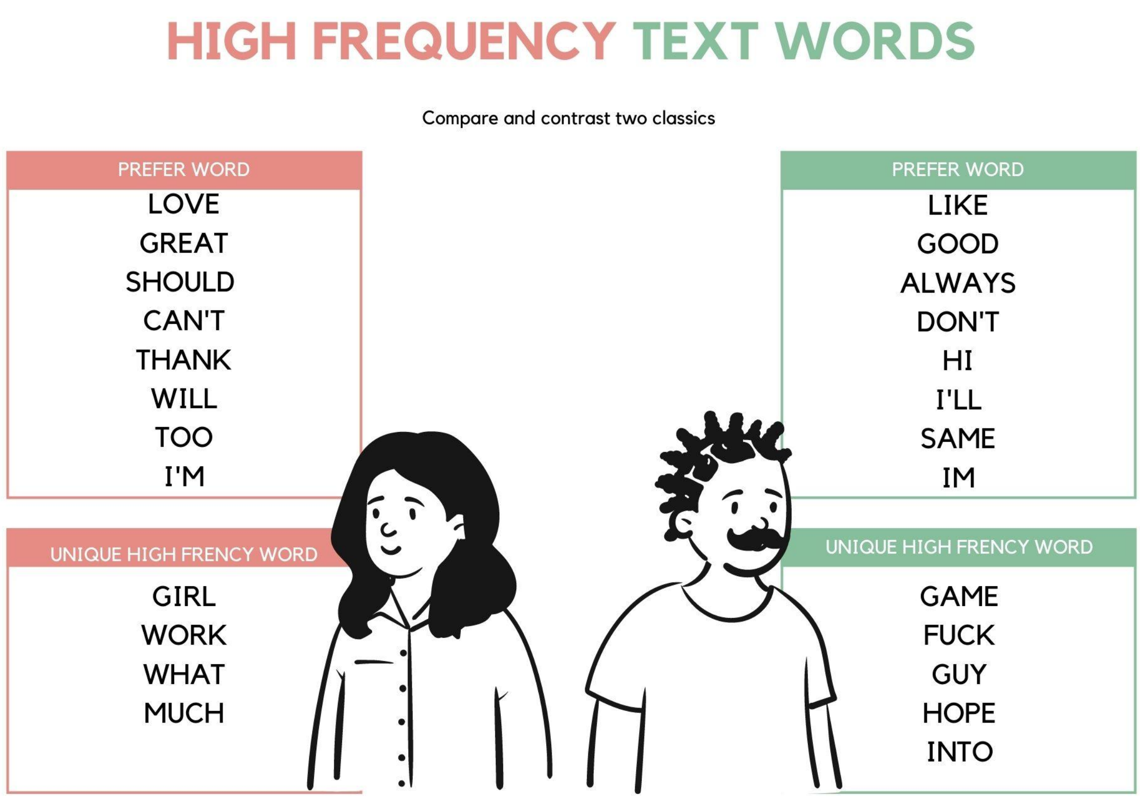
Word Cloud of Gender: Male



Word Cloud of Gender: Female



High-frequency words:



Label Encoding:

```python
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
y = encoder.fit_transform(data['gender'])

# split the dataset in train and test
X = data['text_Cleaned']
# Stratify will create a train set with the same class balance than the original set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0, stratify=y)
```

Splitting train and test data:

The train and test data is split into 70% and 30%

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0, stratify=y)
```

Training data on different ML models:

We have tried to implement the different prediction models just by using text and non -text and text columns without involving any sentiment analysis with the textual data.

Features used for the predictions:

Link Color: It has non-text data and indicates the link color on the profile
Description: The user's profile description
Text: Text of a random one of the user's tweets
We found these attributes to provide useful information regarding gender classification.

Label Encoder: Before the prediction, we have encoded the target column to 0-1

Predictions using non-text column- Link Color

Twitter allows customizing and personalizing the account by changing the colors of the links or the sidebars, and we expect people from different genders to have different behaviors in how they personalize their page.

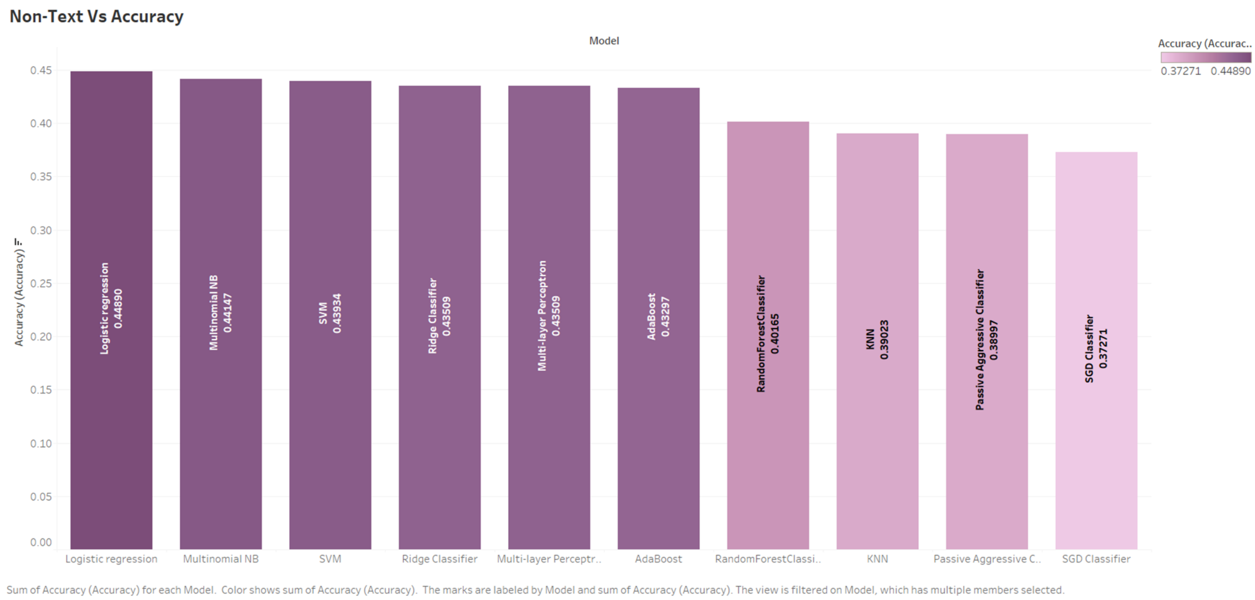
Hence, we have used link color as the feature for this prediction.

```python
def model_test(model,X_train,y_train,X_test,y_test, full_voc, displayResults = True, displayColors = False, featureIntent = 'text'):
    switcher = {

        'link_color' : "theme color",
    }
    featureText =  switcher.get(featureIntent, '')

    model.fit(X_train,y_train)
    y_pred = model.predict(X_test)

    # compute MSE
    mse = metrics.mean_squared_error(y_test,y_pred)
    print('mse: {:.4f}'.format(mse))

    # Prints the accuracy of the gender prediction
    acc = model.score(X_test,y_test)
    print('score: ', acc)

    import matplotlib.pyplot as plt
    import sklearn
    conf = sklearn.metrics.confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(5,5))
    sns.heatmap(conf, annot=True, fmt=".3f", linewidths=.5, square = True, cmap = 'Blues');
    plt.ylabel('Actual label');
    plt.xlabel('Predicted label');
    plt.imshow(conf, cmap='binary', interpolation='None')
    plt.show()


    return model, acc
```

```python
def compute_bag_of_words(text):
    vectorizer = CountVectorizer()
    vectors = vectorizer.fit_transform(text)
    vocabulary = vectorizer.get_feature_names()
    return vectors, vocabulary
```

```python
def predictors(df, feature, model, modelname, displayResults = True, displayColors = False):
  print('Testing', modelname, 'model for gender prediction using', feature)
  full_bow, full_voc = compute_bag_of_words(df[feature])
  X = full_bow
  y = LabelEncoder().fit_transform(df['gender'])
#     # Create Training and testing sets.
  n,d = X.shape
  test_size = n // 5
  print('Split: {} testing and {} training samples'.format(test_size, y.size - test_size))
  perm = np.random.permutation(y.size)
  X_test  = X[perm[:test_size]]
  X_train = X[perm[test_size:]]
  y_test  = y[perm[:test_size]]
  y_train = y[perm[test_size:]]
  print('model: ', modelname)
  model, acc = model_test(model,X_train,y_train,X_test,y_test, full_voc, displayResults = displayResults, displayColors = displayColors, featureIntent = feature)

  return model, full_voc, acc
```

Accuracy Scores



Logistic Regression gave the highest accuracy followed by Multinomial NB SGD has the lowest scores, whereas Adaboost also gave a low accuracy score.

Predictions using text column- Text

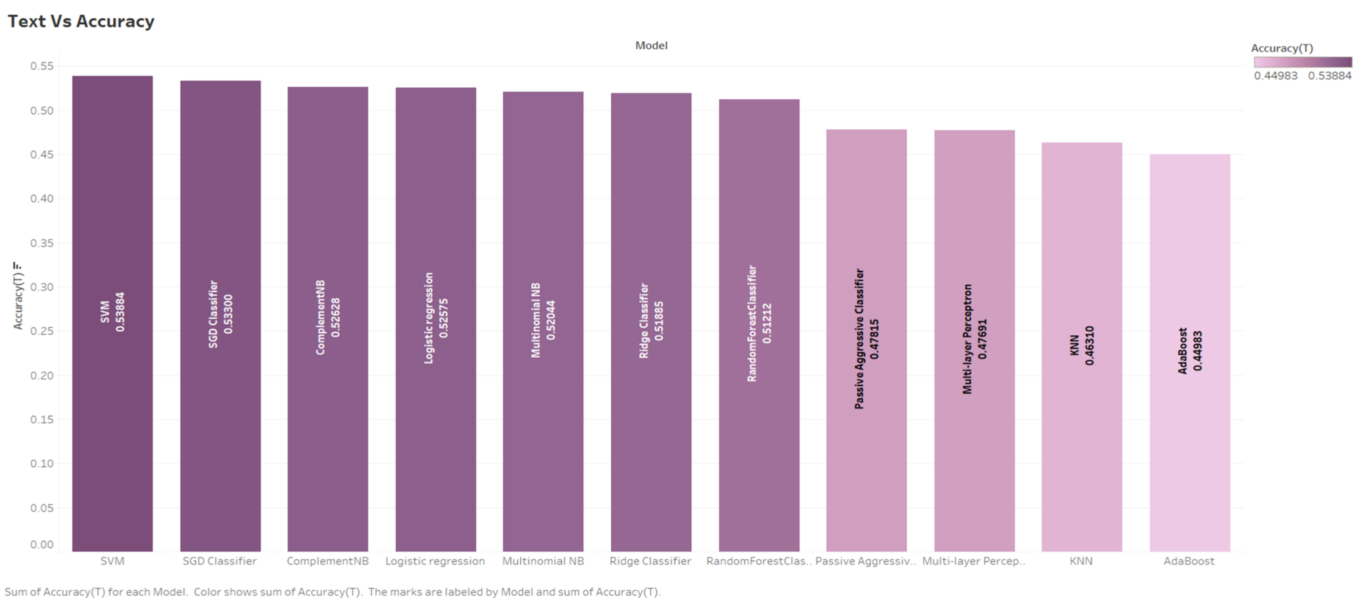
We have cleaned the text column before doing the prediction in order to get rid of noisy data.

We have used TfidfVectorizer to calculate the TF-IDF values and understand the importance and weightage of a word in the text.

```python
def classification_modeling(X_train, X_test, y_train, y_test, text_feature=False):
    """
    This function iterates different possible models
    and return corresponding accuracy

    Args:
        text_feature: Whether the model handles text features or not

    Return: The best fitted model
    """
    clf_dict = {'lr': linear_model.LogisticRegression(multi_class='ovr', random_state=0),
                'rf': ensemble.RandomForestClassifier(n_estimators = 50, random_state=0),
            'svm': SVC(kernel = 'rbf', probability=True),
                 'nb': ComplementNB(),
               'ridge': linear_model.RidgeClassifier(),
              'sgd' :linear_model.SGDClassifier(),
             'passaggre':linear_model.PassiveAggressiveClassifier(),
              'NB':naive_bayes.MultinomialNB(),
            'NN':neural_network.MLPClassifier(),
            'Knn': neighbors.KNeighborsClassifier(n_neighbors=5,weights='distance',algorithm='auto'),
            'Adaboost':ensemble.AdaBoostClassifier()
              }
    result_dict = dict.fromkeys(clf_dict, None)
    pred_dict = dict.fromkeys(clf_dict, None)

    modelNamesList = [
                'LogisticRegression',
                'RandomForestClassifier',
                'SVM',
                 'ComplementNB',
                'RidgeClassifier',
             'SGDClassifier',
              'PassiveAggressiveClassifier',
              'MultinomialNB',
            'MLPClassifier',
            'KNN',
            'Adaboost'

                 ]
    acc_color = np.zeros(len(modelListColor))


    acc_val=[]
    for clf_key in clf_dict:
        if text_feature == True:
            tfidf = TfidfVectorizer()
            clf = Pipeline([('vect', tfidf),
                            ('clf', clf_dict[clf_key])])
        else:
            clf = clf_dict[clf_key]
        clf.fit(X_train, y_train)
        predictions = clf.predict(X_test)
```

```
        acc = accuracy_score(y_test,predictions)
        result_dict[clf_key] = acc
        pred_dict[clf_key] = predictions
        print('Fitting ' + clf_key + ' - Acc:', acc)
        acc_val.append(acc);


        import matplotlib.pyplot as plt
        import sklearn
        conf = sklearn.metrics.confusion_matrix(y_test,predictions)
        plt.figure(figsize=(5,5))
        sns.heatmap(conf, annot=True, fmt=".3f", linewidths=.5, square = True, cmap = 'icefire');
        plt.ylabel('Actual label');
        plt.xlabel('Predicted label');
        plt.imshow(conf, cmap='binary', interpolation='None')
        plt.show()
        #print('Confusion matrix:\n',confusion_matrix(y_test,predictions))
        #print('-'*40)
        #print(acc_val)

    win_clf = max(result_dict, key=lambda key: result_dict[key])
    print("Win classifier: ", win_clf, "- Acc: ",result_dict[win_clf])
    fig, ax1 = plt.subplots(figsize=(8,8))
    ax1.set_xlim([0, 1])
    #bar_width = 0.5
#plt.figure(figsize=(8,8))
    model_number = np.arange(len(modelNamesList))+1
    rects1 = plt.barh(model_number, acc_val,color = '#D95319')
    plt.yticks(model_number,modelNamesList)
    plt.xlabel('Accuracy with text')
    plt.ylabel('Model')
    plt.title('Accuracy of the different Classifiers')
    plt.tight_layout()
    plt.show()
    return np.asarray(pred_dict[win_clf])
```

Accuracies:



Support vector machines (SVM) have the highest accuracy score followed by SGD- stochastic gradient descent.

Overall accuracy score increased compared to scores from using just link color feature but still, it is between 47%-53% and has not improved significantly.

Predictions using text + description column

```
data['text_description'] = data['text_Cleaned'].str.cat(data['description'], sep=' ')
```

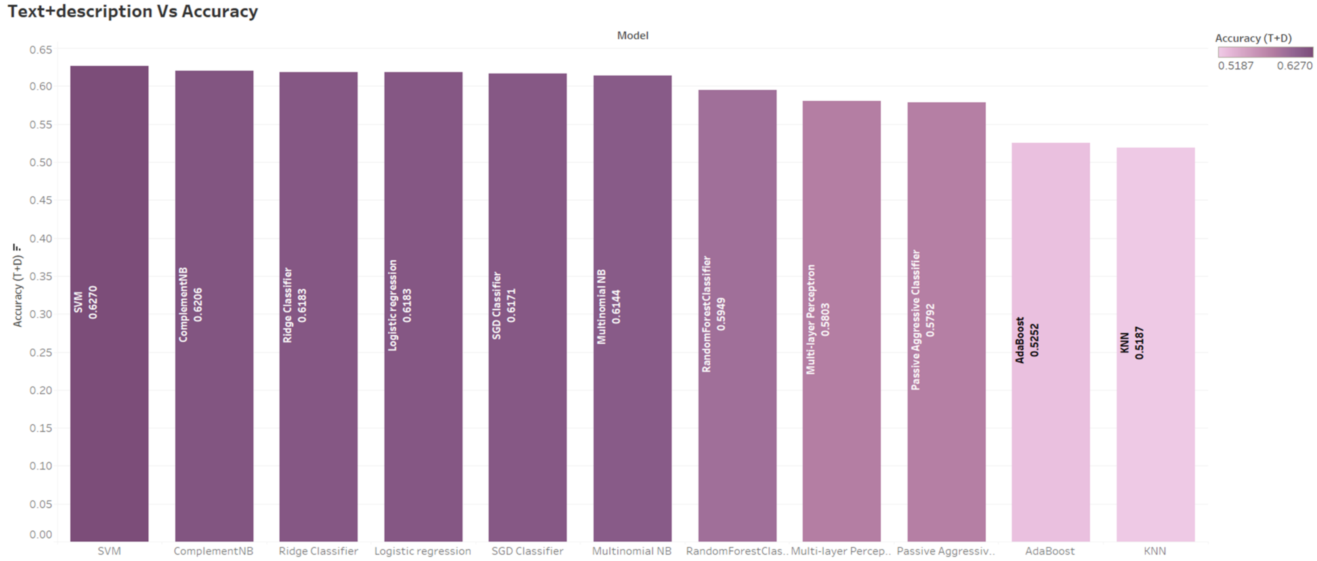
```
X = data['text_description']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0, stratify=y)
#In the code line above, stratify will create a train set with the same class balance than the original set

X_train.head()
X_train.isnull().values.any() # Check if any null values, True if there is at least one.
```

```
False
```

```
best_text_preds = classification_modeling(X_train, X_test, y_train, y_test, text_feature=True)
```

We have concatenated the text and the description column and then have used the same code as described above

Text+description Vs Accuracy



Sum of Accuracy (T+D) for each Model. Color shows sum of Accuracy (T+D). The marks are labeled by Model and sum of Accuracy (T+D).

SVM again has the highest score followed by Complement Naive Bayes and the accuracy scores have increased significantly when we are combining the text and the description column. We have better prediction chances using the user's profile description and the text they are tweeting. There is no single prediction model which performs well in all the cases; however, we see that overall SVM has higher accuracy predicting the gender compared to other models.

Note- Prediction graphs generated using tableau , based on the data from the prediction models

By using boosting classifiers like LGBM Classifier, We are getting an accuracy of 56%

```
lgbmodel = LGBMClassifier(max_depth=5)
lgbmodel.fit(X_train, y_train)
y_pred_lgbm = lgbmodel.predict(X_test)
accuracy_lgbm = accuracy_score(y_test, y_pred_lgbm)
print("Accuracy:%.2f%%" %(accuracy_lgbm * 100.0))
```

```
Accuracy:56.71%
```

By using SVM Classifier, We are getting an accuracy of 62.7%

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
lr	Logistic Regression	0.6599	0.8211	0.6638	0.6636	0.6610	0.4829	0.4834	49.698

With PyCaret, and feature engineering (explained previously), our highest possible accuracy given our computation limits was 65.99%.