Exploring the Lord of the Rings Dataset

Litzy Yulissa Nevarez García

In [1]:
```
#Importamos la librerias necesarias
import numpy as np
import pandas as pd
import math
import seaborn as sns
import re
import missingno as msno
import os
from pandas import read_csv
import matplotlib.pyplot as plt
import matplotlib as mpl
import itertools
import graphviz
import json
import time
import gc
import nltk
from os import path
from PIL import Image
import eli5
import plotly.graph_objs as go
from plotly.offline import init_notebook_mode, iplot
from collections import Counter
from sklearn import model_selection
#from sklearn.preprocessing import Imputer
from sklearn.model_selection import train_test_split, cross_val_score, KFold, learning
from sklearn.metrics import confusion_matrix, make_scorer, accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC, LinearSVC
from sklearn.model_selection import cross_val_score
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import CountVectorizer
from wordcloud import WordCloud, STOPWORDS
from collections import defaultdict
import keras
import keras.backend as K
from keras.layers import Dense, GlobalAveragePooling1D, Embedding
from keras.callbacks import EarlyStopping
from keras.models import Sequential
#from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
from keras.utils import to_categorical
import networkx as nx
#import plotly.plotly as py
from chart_studio import plotly as py
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
from plotly import tools
#import plotly.plotly as py
from collections import Counter
```

```
from nltk.util import ngrams
from nltk.corpus import stopwords
from IPython.core import display as ICD
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
from nltk.sentiment import SentimentAnalyzer
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from sklearn.utils import shuffle
from sklearn.metrics import accuracy_score
from sklearn.utils import class_weight
from sklearn.model_selection import GridSearchCV, learning_curve
from palettable.colorbrewer.qualitative import Pastel1_7
pd.set_option('display.max_colwidth', -1)
pd.set_option('display.max_rows', None)
%matplotlib inline
init_notebook_mode(connected=True) # plotly
import warnings
warnings.filterwarnings("ignore")
```

```
C:\Users\yulis\AppData\Local\Temp\ipykernel_7948\1667921098.py:63: FutureWarning:

Passing a negative integer is deprecated in version 1.0 and will not be supported in
future version. Instead, use None to not limit the column width.
```

Cargamos y procesamos los Datos

In [2]:

```
def cleanData(character):
    char = script1[script1["char"]==character]["dialog"].map(lambda x : x.replace(" ,'
    return char

def countTotal(df,groupby,toCount):
    total = df.groupby([groupby])[toCount].count()
    total = pd.DataFrame(total)
    total = total.reset_index().sort_values(toCount,ascending=0)
    total.reset_index(drop = True)
    total.columns = [groupby, 'Count']
    return total

def countMarried(df, groupby,toCount):
    married = df.groupby(groupby).count()[toCount]
    married = pd.DataFrame(married)
    married = married.reset_index().sort_values(toCount,ascending=0)
    married.reset_index(drop = True)
    married.columns = [groupby,'Count']
    return married

def countCharacters(beginSlice,endSlice):
    counted = int(race2[beginSlice:endSlice]['name'].values)
    return counted

def calcMarriage(beginSlice,endSlice):
    total = int(countTotal(otherData,'race','spouse')[beginSlice:endSlice]['Count'].va
    married = int(countMarried(married,'race','spouse')[beginSlice:endSlice]['Count'].
    unmarried = total - married
    return married, unmarried, total

def grabValue(df,beginSlice,endSlice):
```

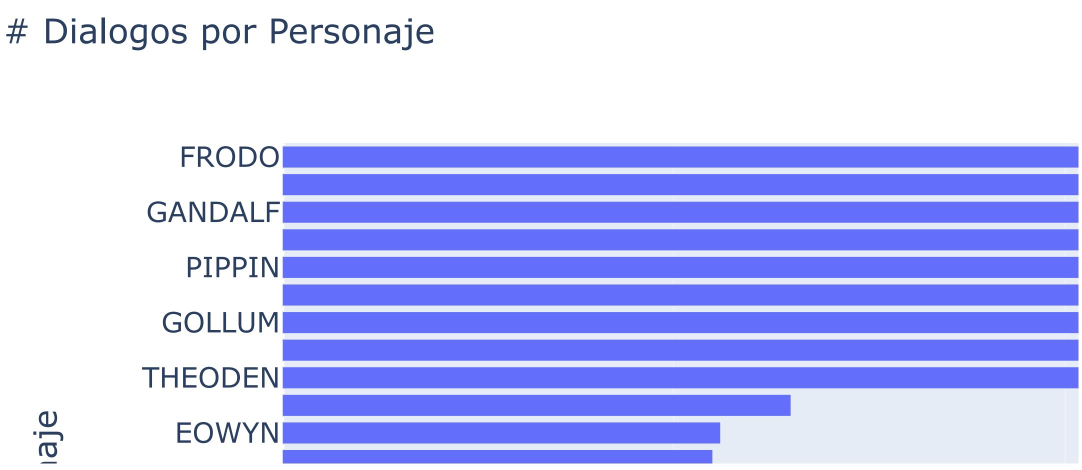
```
    count = int(df.iloc[beginSlice:endSlice]['Count'])
    return count

scriptPath = 'lotr_scripts.csv'
characterPath = 'lotr_characters.csv'
script = pd.read_csv(f"{scriptPath}",encoding='utf-8')
otherData = pd.read_csv(f"{characterPath}",dtype={2:'str'})

married = otherData[~otherData.spouse.isnull()]
married = married[married.spouse != "None"]
married = married.reset_index(drop=True)
otherData = otherData.reset_index(drop=True)

script["count"] = script["char"].map(lambda x: script["char"].tolist().count(x) )
script = script.sort_values("count",ascending = False)
script1 = script[script["count"]>=22]
order = script1["char"].unique()
char = script1["char"]
lineCounts = char.value_counts()
lineCounts = lineCounts.sort_values(ascending = False)[0:50]
```

Visualizamos los Datos

Primero comenzamos con una gráfica que muestra el total de número de lineas de cada personaje.

In [3]:
```
# Vertical Plot
result1 = lineCounts
trace1 = go.Bar(
                x = result1.index,
                y = result1,
                name = "citations",
                marker = dict(color = 'rgba(6, 133, 23, 0.8)',
                             line=dict(color='rgb(0,0,0)',width=1.5)),
                text = result1.index)
data = [trace1]
layout = go.Layout(barmode = "group",title='Total de Dialogos por Personaje', yaxis= d
fig = go.Figure(data = data, layout = layout)
iplot(fig)

# Horizontal Plot
temp = script1['char'].value_counts()
trace = go.Bar(y=temp.index[::-1],x=(temp)[::-1],orientation = 'h')
layout = go.Layout(title = "# Dialogos por Personaje",xaxis=dict(title='# Dialogos por
                   yaxis=dict(title='Personaje',titlefont=dict(size=16),tickfont=dict(
data = [trace]
fig = go.Figure(data=data, layout=layout)
iplot(fig,filename='basic-bar')
```


Total de Dialogos por Personaje
200
150
ablados
FRODO
SAM
GANDALF
ARAGORN
PIPPIN

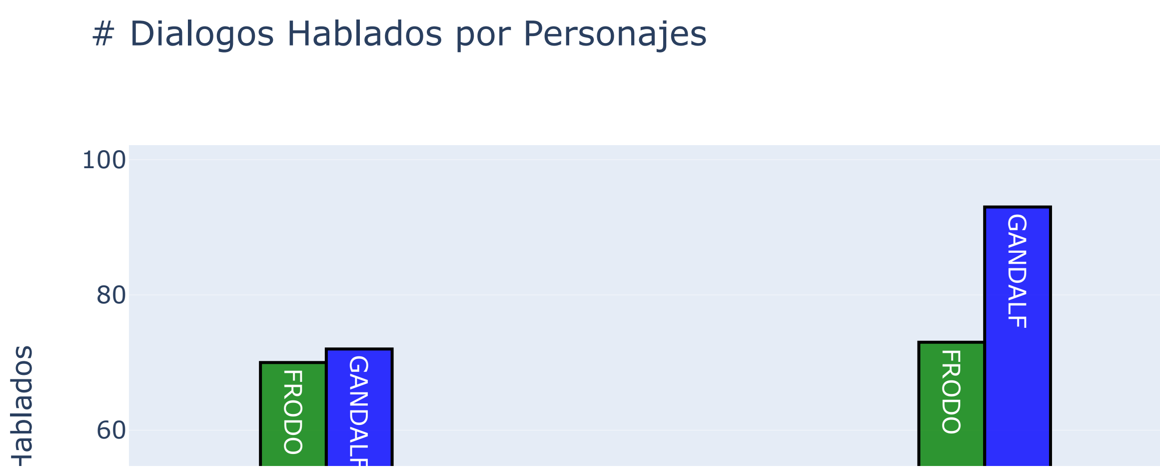


Si se divide la trilogía en tres partes, algunos de los personajes son mucho más importantes durante el comienzo de la trilogía (por ejemplo, Gandalf, que muere), mientras que otros son mucho más importantes hacia el final de la trilogía (por ejemplo, Gollum, que se une al equipo).

In [4]:
```
#gourpby movies and characters
grouped = script1.groupby(['char',"movie"]).count()
grouped.columns = ["_".join(x) for x in grouped.columns.ravel()]
grouped = grouped.reset_index()
grouped = grouped.iloc[:,:3]
grouped.columns = ["char","movie","count"]
grouped.head()

# grouped['char'].unique()
ARAGORN = grouped[0:3] #
ARWEN = grouped[3:6]
BILBO = grouped[6:8] #
BOROMIR = grouped[8:10]
DENETHOR = grouped[10:12]
ELROND = grouped[12:15]
EOMER = grouped[15:17]
EOWYN = grouped[17:19]
FARAMIR = grouped[19:21]
FRODO = grouped[21:24] #
GANDALF = grouped[24:27] #
GIMLI = grouped[27:30]
```

```
GOLLUM = grouped[30:33] #
GRIMA = grouped[33:35]
LEGOLAS = grouped[35:38]
MERRY = grouped[38:41] #
ORC = grouped[41:44]
PIPPIN = grouped[44:47] #
SAM = grouped[47:50] #
SARUMAN = grouped[50:53]
SMEAGOL = grouped[53:55]
SOLDIER = grouped[55:57]
STRIDER = grouped[57:58]
THEODEN = grouped[58:60]
TREEBEARD = grouped[60:62]

trace7 = go.Bar(
                x = ARAGORN.movie,
                y = ARAGORN['count'],
                name = "ARAGORN",
                marker = dict(color = 'rgba(32, 64, 32, 0.8)',
                             line=dict(color='rgb(0,0,0)',width=1.5)),
                text = ARAGORN.char)
trace6 = go.Bar(
                x = SAM.movie,
                y = SAM['count'],
                name = "SAM",
                marker = dict(color = 'rgba(32, 32, 32, 0.8)',
                             line=dict(color='rgb(0,0,0)',width=1.5)),
                text = SAM.char)
trace5 = go.Bar(
                x = PIPPIN.movie,
                y = PIPPIN['count'],
                name = "PIPPIN",
                marker = dict(color = 'rgba(128, 128, 128, 0.8)',
                             line=dict(color='rgb(0,0,0)',width=1.5)),
                text = PIPPIN.char)
trace4 = go.Bar(
                x = MERRY.movie,
                y = MERRY['count'],
                name = "MERRY",
                marker = dict(color = 'rgba(255, 255, 0, 0.8)',
                             line=dict(color='rgb(0,0,0)',width=1.5)),
                text = MERRY.char)
trace3 = go.Bar(
                x = GOLLUM.movie,
                y = GOLLUM['count'],
                name = "GOLLUM",
                marker = dict(color = 'rgba(0, 255, 255, 0.8)',
                             line=dict(color='rgb(0,0,0)',width=1.5)),
                text = GOLLUM.char)
trace2 = go.Bar(
                x = GANDALF.movie,
                y = GANDALF['count'],
                name = "GANDALF",
                marker = dict(color = 'rgba(0, 0, 255, 0.8)',
                             line=dict(color='rgb(0,0,0)',width=1.5)),
                text = GANDALF.char)
trace1 = go.Bar(
                x = FRODO.movie,
                y = FRODO['count'],
                name = "FRODO",
```

```
                marker = dict(color = 'rgba(0, 128, 0, 0.8)',
                             line=dict(color='rgb(0,0,0)',width=1.5)),
                text = FRODO.char)
trace0 = go.Bar(
                x = BILBO.movie,
                y = BILBO['count'],
                name = "BILBO",
                marker = dict(color = 'rgba(0, 128, 128, 0.8)',
                             line=dict(color='rgb(0,0,0)',width=1.5)),
                text = BILBO.char)
data = [trace0,trace1,trace2,trace3,trace4,trace5,trace6,trace7]
layout = go.Layout(barmode = "group",title='# Dialogos Hablados por Personajes',yaxis=

fig = go.Figure(data = data, layout = layout)
iplot(fig)
```



NLKT

Es un conjunto de librerías y programas para Python que nos permiten lleva a cabo muchas tareas relacionadas con el Procesamiento del Lenguaje Natural.

Aquí usamos la función NLTK.SentimentIntensityAnalyzer() para analizar el diálogo con más detalle. También podemos ver que Sam es el más negativo de los personajes, siempre está tratando de proteger al Sr. Frodo y es muy cauteloso, mientras que Pippen y Merry son muy positivos (dadas sus circunstancias más cómodas).

In [5]:
```
#Descargamos la libreria
nltk.download()
```

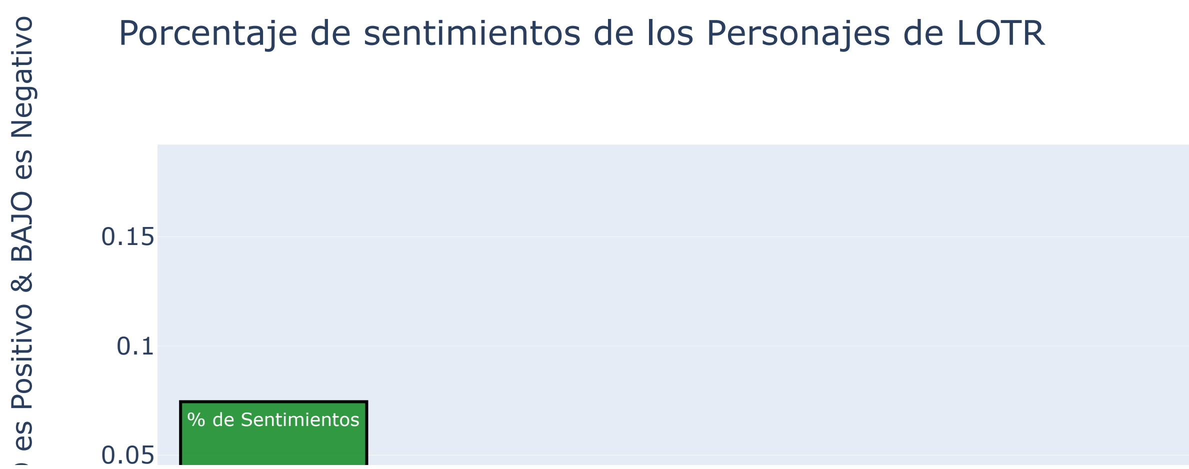
```
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml
```

Out[5]:
```
True
```

Este código realiza el análisis de sentimientos de varios personajes de El Señor de los Anillos (LOTR) y genera un gráfico de barras que muestra el porcentaje de sentimientos positivos y negativos para cada personaje. Se usa función sentiment se define para calcular el sentimiento promedio de una lista de textos. Utiliza la biblioteca VaderSentiment para asignar puntuaciones de sentimiento a cada texto y luego calcula la media de las puntuaciones negativas, positivas y compuestas para los textos que tienen una puntuación compuesta distinta de cero.

In [5]:
```
# adapted from https://github.com/tianyigu/Lord_of_the_ring_project/blob/master/LOTR_c

FRODO1 = cleanData("FRODO")
SAM1 = cleanData("SAM")
GANDALF1 = cleanData("GANDALF")
ARAGORN1 = cleanData("ARAGORN")
GOLLUM1 = cleanData("GOLLUM")
SMEAGOL1 = cleanData("SMEAGOL")
PIPPIN1 = cleanData("PIPPIN")
MERRY1 = cleanData("MERRY")
ARWEN1 = cleanData("ARWEN")
ORC1 = cleanData("ORC")

charlist = {"FRODO":FRODO1,"SAM":SAM1,"GANDALF":GANDALF1, "ARAGORN": ARAGORN1,"GOLLUM"

def sentiment(char):
    vader = SentimentIntensityAnalyzer()
    res_dic = [vader.polarity_scores(text) for text in charlist[char]]
    res_dic = [res_dic[i] for i in range(len(res_dic)) if res_dic[i]["compound"]!=0]
    res_neg = np.mean([res_dic[i]['neg'] for i in range(len(res_dic))])
    res_pos = np.mean([res_dic[i]['pos'] for i in range(len(res_dic))])
    res_com = np.mean([res_dic[i]['compound'] for i in range(len(res_dic))])
    return res_com


FRODO = sentiment('FRODO')
SAM = sentiment('SAM')
GANDALF = sentiment('GANDALF')
ARAGORN = sentiment('ARAGORN')
GOLLUM = sentiment('GOLLUM')
SMEAGOL = sentiment('SMEAGOL')
PIPPIN = sentiment('PIPPIN')
MERRY = sentiment('MERRY')
ARWEN = sentiment('ARWEN')

raw_data = {'Character': ['Frodo', 'Sam', 'Gandalf', 'Aragorn','Gollum','Smeagol','Pip
        'SentimentScore': [FRODO,SAM,GANDALF,ARAGORN,GOLLUM,SMEAGOL,PIPPIN,MERRY,ARWEN
df = pd.DataFrame(raw_data)

result1 = df
```

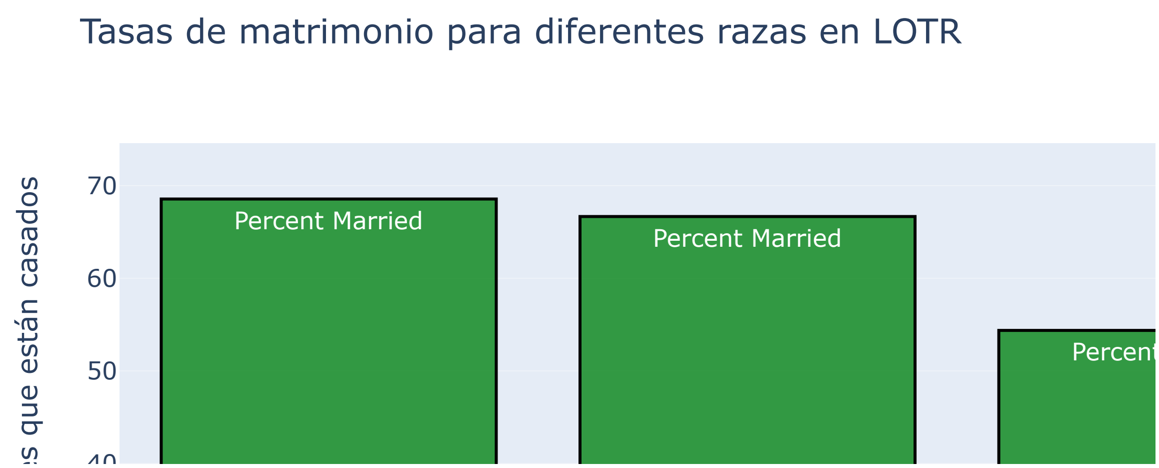
```
trace1 = go.Bar(
                x = result1.Character,
                y = result1.SentimentScore,
                name = "Sentiment Score -- High is Positive & Low is Negative",
                marker = dict(color = 'rgba(6, 133, 23, 0.8)',
                             line=dict(color='rgb(0,0,0)',width=1.5)),
                text = "% de Sentimientos")
data = [trace1]
layout = go.Layout(barmode = "group",title='Porcentaje de sentimientos de los Personaj
fig = go.Figure(data = data, layout = layout)
iplot(fig)
```

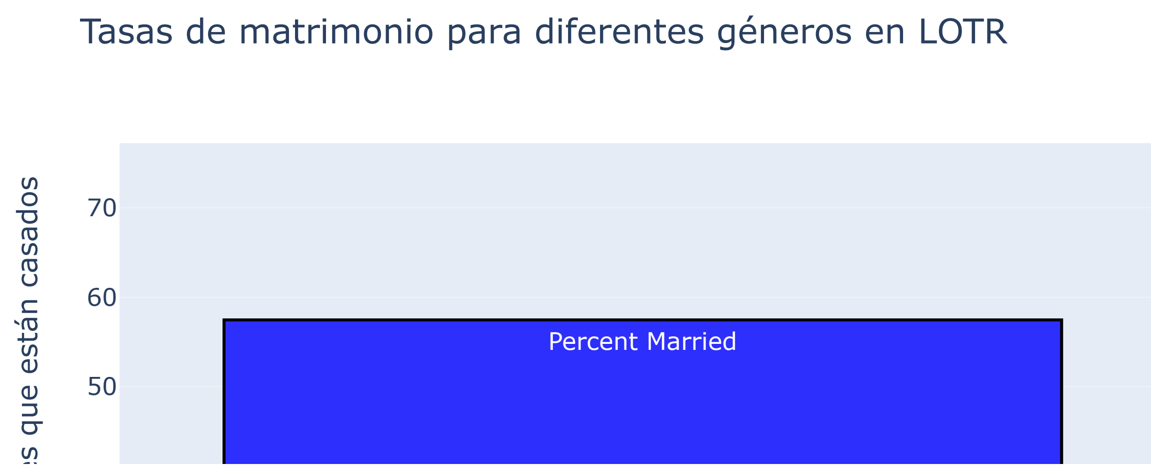


La mayoría de los personajes de la trilogía de El señor de los anillos están casados, incluidas casi todas las mujeres.

In [6]:
```
raceData = otherData
raceData = raceData[~raceData.race.isnull()]
raceData = raceData.reset_index(drop=True)
race = ["Men",'Hobbits','Elves','Dwarves','Dragons','Half-elven','Ainur','Orcs']
race2 = raceData.groupby(["gender","race"])["name"].count()
race2 = race2.reset_index()
race2 = race2[race2['race'].isin(race)]
race2 = race2[0:14]

married2 = countMarried(married,'race','spouse')
total = countTotal(otherData,'race','name')
```

```
menMarriedCount = grabValue(married2,0,1)
hobbitsMarriedCount = grabValue(married2,1,2)
elvesMarriedCount = grabValue(married2,2,3)
dwarvesMarriedCount = grabValue(married2,3,4)
ainurMarriedCount = grabValue(married2,4,5)
menTotalCount = grabValue(total,0,1)
hobbitsTotalCount = grabValue(total,1,2)
elvesTotalCount = grabValue(total,2,3)
dwarvesTotalCount = grabValue(total,4,5)
ainurTotalCount = grabValue(total,3,4)
menMarriedPercent = (menMarriedCount*100)/menTotalCount
hobbitsMarriedPercent = (hobbitsMarriedCount*100)/hobbitsTotalCount
elvesMarriedPercent = (elvesMarriedCount*100)/elvesTotalCount
dwarvesMarriedPercent = (dwarvesMarriedCount*100)/dwarvesTotalCount
ainurMarriedPercent = (ainurMarriedCount*100)/ainurTotalCount

raw_data = {'Race': ['Men', 'Hobbits','Elves','Dwarves','Ainur'],
        'PercentMarried': [menMarriedPercent,hobbitsMarriedPercent,elvesMarriedPercent
df = pd.DataFrame(raw_data)


result1 = df
trace1 = go.Bar(
                x = result1.Race,
                y = result1.PercentMarried,
                name = "Percent Married",
                marker = dict(color = 'rgba(6, 133, 23, 0.8)',
                             line=dict(color='rgb(0,0,0)',width=1.5)),
                text = "Percent Married")
data = [trace1]
layout = go.Layout(barmode = "group",title='Tasas de matrimonio para diferentes razas
fig = go.Figure(data = data, layout = layout)
iplot(fig)


total1 = countTotal(otherData,'gender','name')
married1 = countMarried(married,'gender','spouse')
maleMarriedCount = grabValue(married1,0,1)
femaleMarriedCount = grabValue(married1,1,2)
maleTotalCount = grabValue(total1,0,1)
femaleTotalCount = grabValue(total1,1,2)
maleMarriageRate = (maleMarriedCount*100)/maleTotalCount
femaleMarriageRate = (femaleMarriedCount*100)/femaleTotalCount
raw_data = {'Gender': ['Masculino', 'Femenino'],
        'PercentMarried': [maleMarriageRate,femaleMarriageRate]}
df = pd.DataFrame(raw_data)

result1 = df
trace1 = go.Bar(
                x = result1.Gender,
                y = result1.PercentMarried,
                name = "Percent Married",
                marker = dict(color = 'rgba(0, 0, 255, 0.8)',
                             line=dict(color='rgb(0,0,0)',width=1.5)),
                text = "Percent Married")
data = [trace1]
layout = go.Layout(barmode = "group",title='Tasas de matrimonio para diferentes génerc
fig = go.Figure(data = data, layout = layout)
iplot(fig)
```

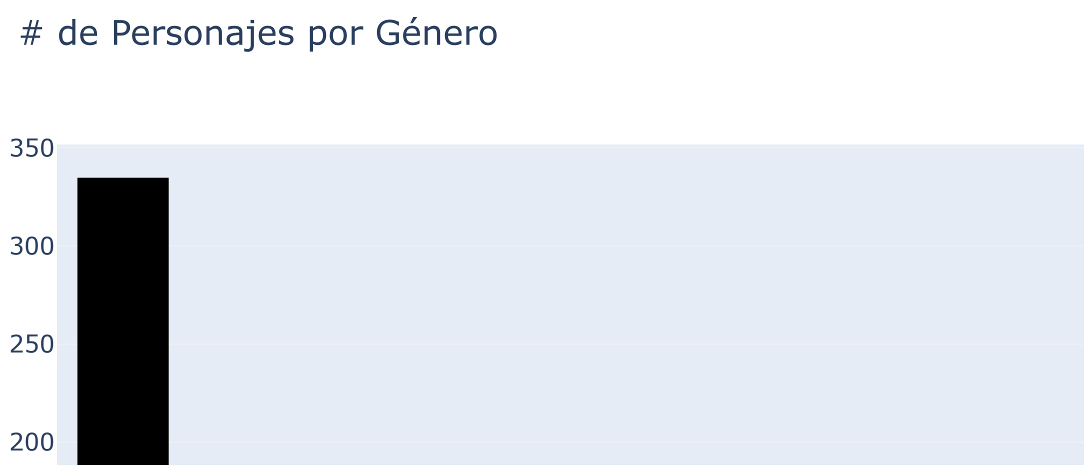

Tasas de matrimonio para diferentes razas en LOTR
70
60
50
40
s que están casados
Percent Married
Percent Married
Percent



A pesar de que la mayoría de los personajes están casados, sorprendentemente hay pocas mujeres en la trilogía de El señor de los anillos.

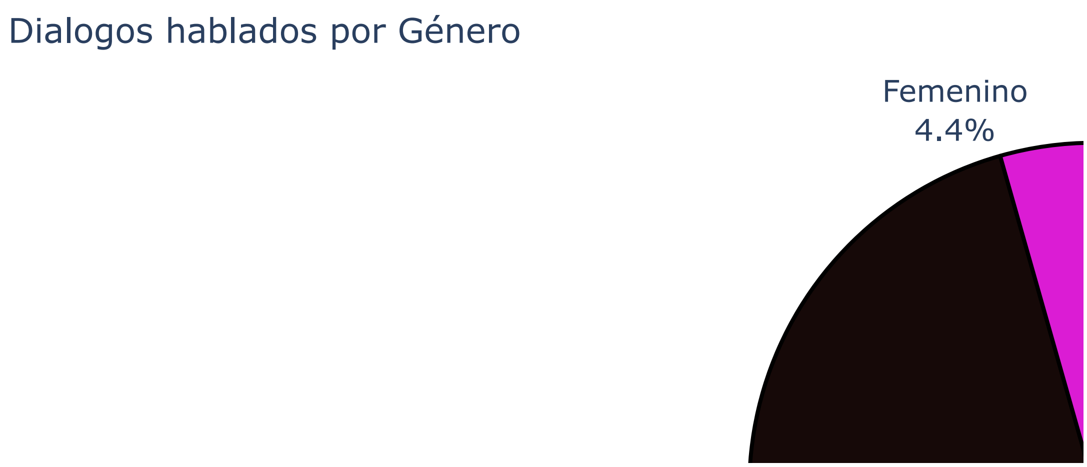
In [7]:
```
menCountM = countCharacters(12,13)
hobbitsCountM = countCharacters(11,12)
elvesCountM = countCharacters(9,10)
dwarvesCountM = countCharacters(8,9)
ainurCountM = countCharacters(6,7)
halfelvenCountM = countCharacters(10,11)
orcsCountM = countCharacters(13,14)
dragonsCountM = countCharacters(7,8)
menCountF = countCharacters(5,6)
hobbitsCountF = countCharacters(4,5)
elvesCountF = countCharacters(2,3)
dwarvesCountF = countCharacters(1,2)
ainurCountF = countCharacters(0,1)
halfelvenCountF = countCharacters(3,4)
orcsCountF = 0
dragonsCountF = 0

gender = ["Male","Female"]
race = ["Men",'Hobbits','Elves','Dwarves','Ainur','Orcs','Half-elven','Dragons']

male = [menCountM, hobbitsCountM, elvesCountM, dwarvesCountM,ainurCountM, halfelvenCou
female = [menCountF, hobbitsCountF, elvesCountF, dwarvesCountM,ainurCountF, halfelven(
```

```python
data = {'race' : race,
        'Male'    : male,
        'Female'    : female}

trace1 = go.Bar(
    x=data['race'],
    y=data['Male'],
    name='# de Personajes Masculinos',
    marker = dict(color = 'rgba(0, 0, 0, 1)', #0, 0, 255, 0.8
                             line=dict(color='rgb(0,0,0)',width=1.5))
)
trace2 = go.Bar(
    x=data['race'],
    y=data['Female'],
    name='# de Personajes Femeninos',
    marker = dict(color = 'rgba(255,0,255,1)',
                             line=dict(color='rgb(0,0,0)',width=1.5))
)

data = [trace1, trace2]
layout = go.Layout(title='# de Personajes por Género',barmode="group")

fig = go.Figure(data=data, layout=layout)
iplot(fig)
```



Solo alrededor del 5% del diálogo en El Señor de los Anillos es hablado por mujeres.

In [8]:
```
women = ['EOWYN','ARWEN']
raw_data = {'Gender': ['Masculino', 'Femenino'],
        'Lines': [sum(lineCounts[0:25])-(lineCounts[17]+lineCounts[11]), (lineCounts[1
df = pd.DataFrame(raw_data, columns = ['Gender', 'Lines'])

labels = df.Gender
values = df.Lines
colors = ["#160908", "#db1cd4"]

trace = go.Pie(labels=labels, values=values,
               hoverinfo='value', textinfo='label+percent',
               textfont=dict(size=15),
               marker=dict(colors=colors,
                           line=dict(color='#000000', width=2)),)
layout = go.Layout(title='Dialogos hablados por Género',
            annotations = [
            { "font": { "size": 20},
              "showarrow": False,
              "text": "% de Dialogos hablados por Género en LOTR",
                "x": 0.55,
                "y": -.2
            },])
fig = go.Figure(data=[trace], layout=layout)
iplot(fig)
```



Construir un modelo para identificar al hablante

A continuación, se construirá un modelo para identificar qué personaje está hablando para cada línea dada del cuadro de diálogo LOTR. Una característica informativa para predecir quién está hablando podría ser los bigramas (pares) o trigramas más comunes de combinaciones de palabras.

A Frodo le gusta decir "El Anillo" y "La Comarca".

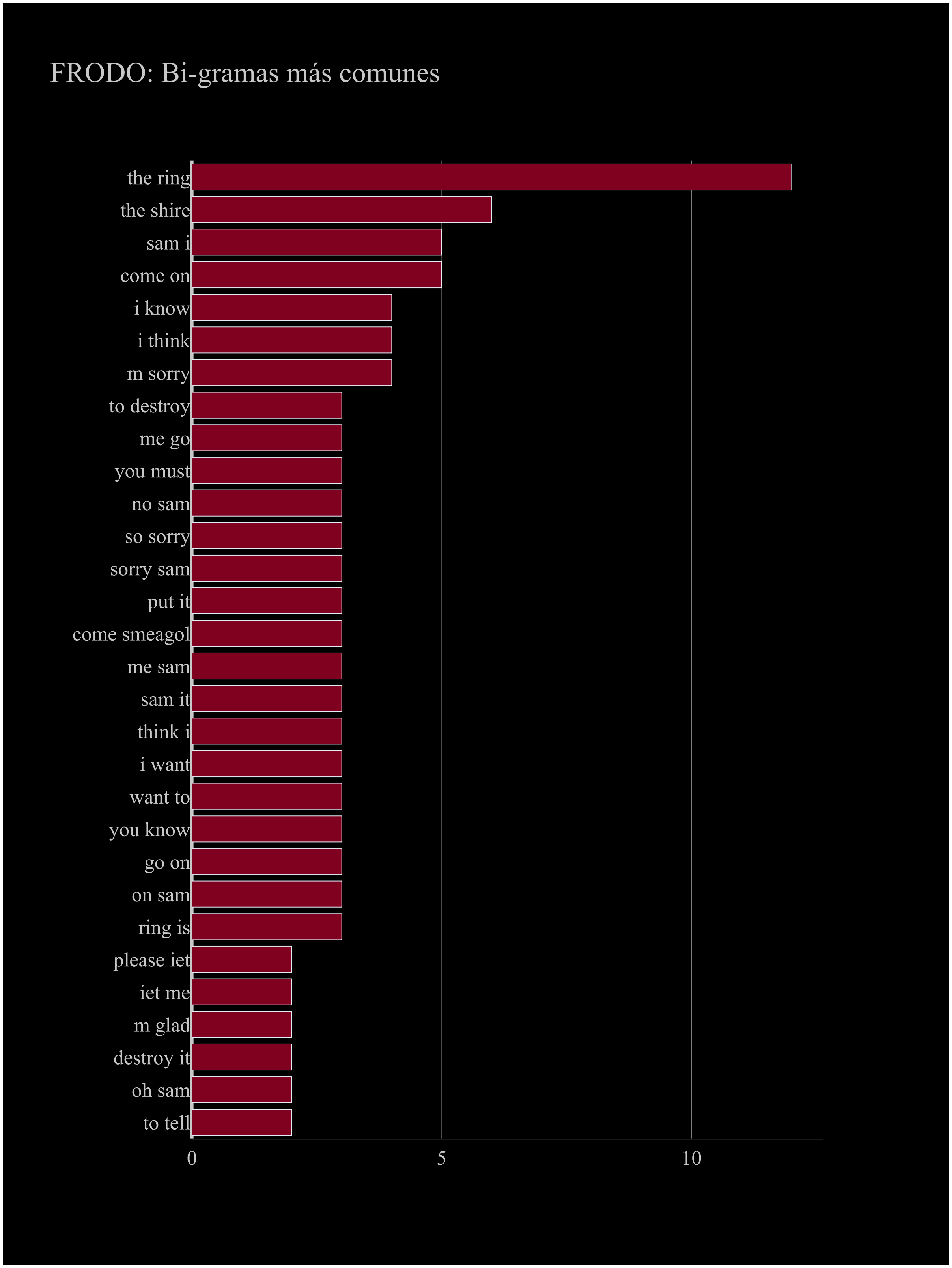
In [9]:
```
#Descargamos la libreria
nltk.download()
```

```
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml
```
Out[9]:
```
True
```

In [9]:
```
script2 = script[['char','dialog']]

def separateDf(df,column,value):
    separated = df[column] == value
    separated = df[separated]
    return separated

FRODO2 = separateDf(script2,'char',"FRODO")
SAM2 = separateDf(script2,'char',"SAM")
GANDALF2 = separateDf(script2,'char',"GANDALF")
ARAGORN2 = separateDf(script2,'char',"ARAGORN")
GOLLUM2 = separateDf(script2,'char',"GOLLUM")
SMEAGOL2 = separateDf(script2,'char',"SMEAGOL")
PIPPIN2 = separateDf(script2,'char',"PIPPIN")
MERRY2 = separateDf(script2,'char',"MERRY")
ARWEN2 = separateDf(script2,'char',"ARWEN")
ORC2 = separateDf(script2,'char',"ORC")

newdf = pd.concat([FRODO2,SAM2,GANDALF2,ARAGORN2,GOLLUM2,SMEAGOL2,PIPPIN2,MERRY2,ARWEN

def preprocess(text):
    text = text.strip()
    text = text.replace("' ", " ' ")
    signs = set(',.:;"?!')
    prods = set(text) & signs
    if not prods:
        return text

    for sign in prods:
        text = text.replace(sign, ' {} '.format(sign) )
    return text

a2c = {"FRODO":0,"SAM":1,"GANDALF":2, "ARAGORN": 3,"GOLLUM": 4, "SMEAGOL":5,"PIPPIN":6
y = np.array([a2c[a] for a in newdf.char])
y = to_categorical(y)
tokenize_regex = re.compile("[\w]+")
sw = set(stopwords.words("english"))

def preprocessText(text, ngram_order):
    """
```

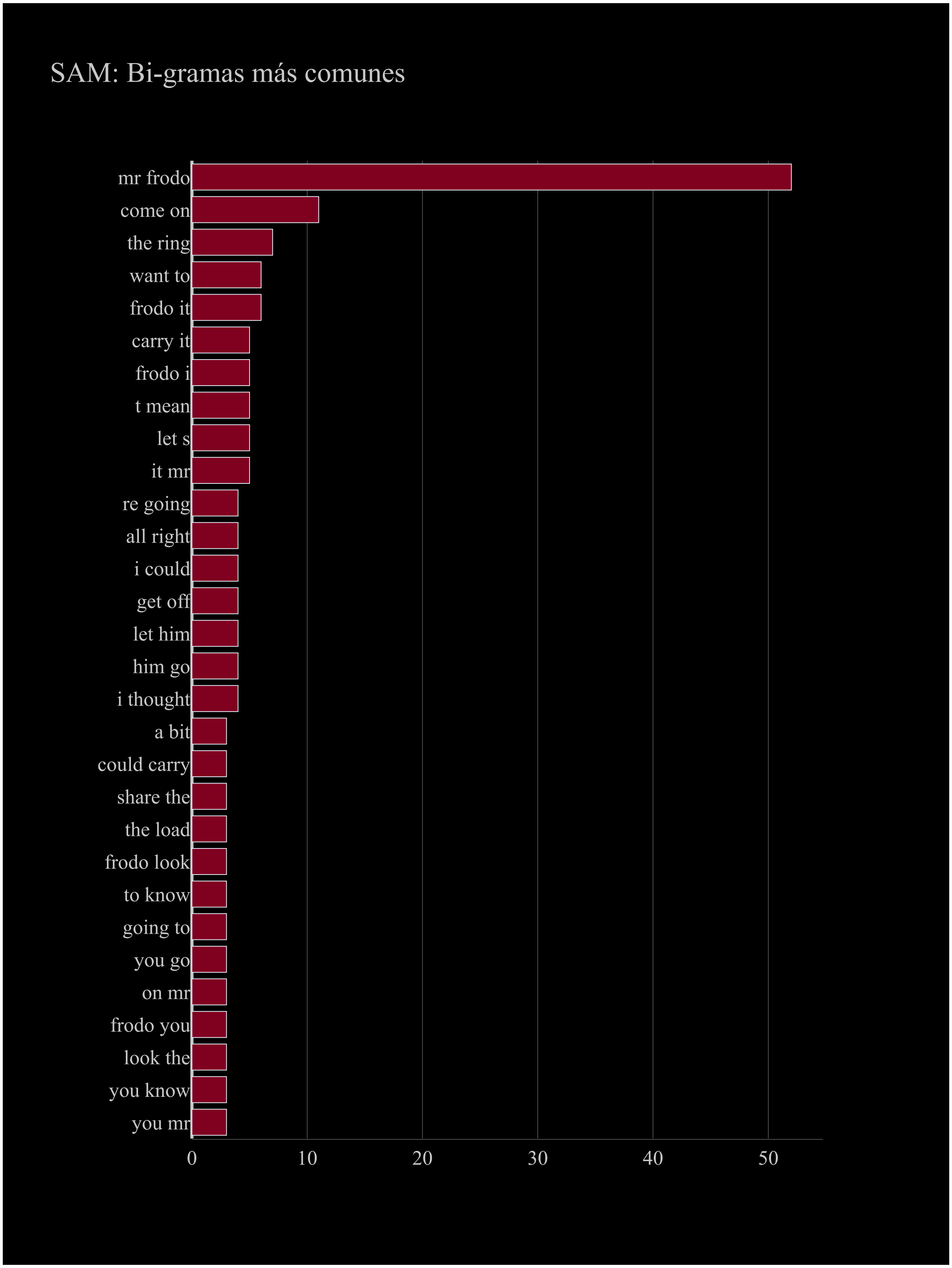
```
    Transform text into a list of ngrams. Feel free to play with the order parameter
    """
    text = text.lower()

    text = [" ".join(ngram) for ngram in ngrams((tokenize_regex.findall(text)), ngram_
            if (set(ngram) - sw)] # instead of filtering stopwords, let's just filter
                                  # with nothing but stopwords
    return text

def draw_word_histogram(texts, title, bars=30):
    """
    Draw a barplot for word frequency distribution.
    """
    # first, do the counting
    ngram_counter = Counter()
    for text in texts:
        ngram_counter.update(text)
    # for plotly, we need two lists: xaxis values and the corresponding yaxis values
    # this is how we split a list of two-element tuples into two lists
    features, counts = zip(*ngram_counter.most_common(bars))
    # now let's define the barplot
    bars = go.Bar(
        x=counts[::-1],  # inverse the values to have the largest on the top
        y=features[::-1],
        orientation="h",  # this makes it a horizontal barplot
        marker=dict(
            color='rgb(128, 0, 32)'  # this color is called oxblood... spooky, isn't i
        )
    )
    # this is how we customize the looks of our barplot
    layout = go.Layout(
        paper_bgcolor='rgb(0, 0, 0)',  # color of the background under the title and i
        plot_bgcolor='rgb(0, 0, 0)',  # color of the plot background
        title=title,
        autosize=False,  # otherwise the plot would be too small to contain axis label
        width=600,
        height=800,
        margin=go.layout.Margin(
            l=120, # to make space for y-axis labels
        ),
        font=dict(
            family='Serif',
            size=13, # a lucky number
            color='rgb(200, 200, 200)'
        ),
        xaxis=dict(
            showgrid=True,  # all the possible lines - try switching them off
            zeroline=True,
            showline=True,
            zerolinecolor='rgb(200, 200, 200)',
            linecolor='rgb(200, 200, 200)',
            gridcolor='rgb(200, 200, 200)',
        ),
        yaxis=dict(
            ticklen=8  # to add some space between yaxis labels and the plot
        )

    )
    fig = go.Figure(data=[bars], layout=layout)
    iplot(fig, filename='h-bar')
```

```
    return

frodo = newdf[newdf.char=="FRODO"].dialog.apply(preprocessText, ngram_order=2)
draw_word_histogram(frodo, "FRODO: Bi-gramas más comunes")
```



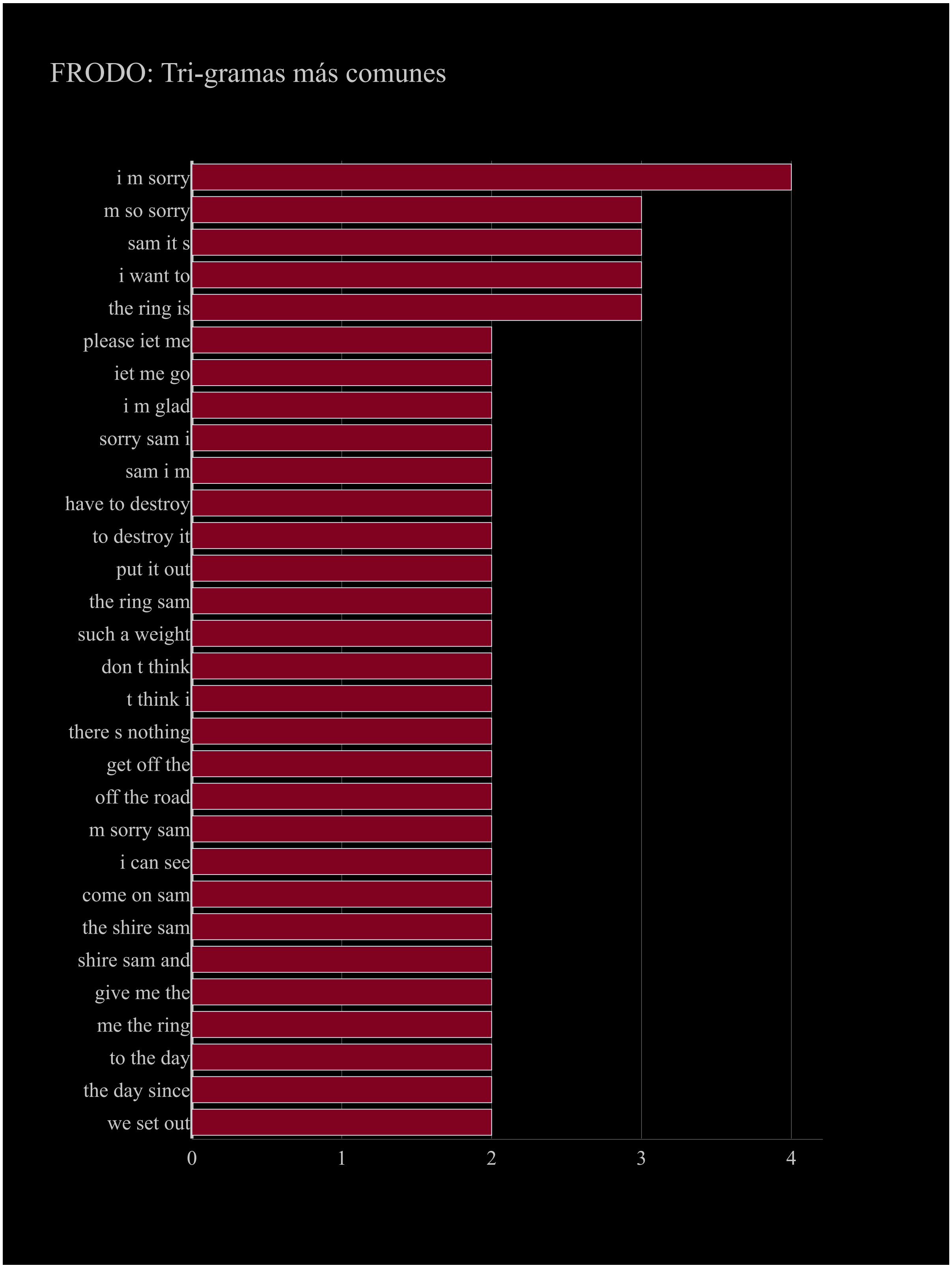
Mientras que a Sam le gusta decir "Sr. Frodo" y "vamos".

In [10]:
```
sam = newdf[newdf.char=="SAM"].dialog.apply(preprocessText, ngram_order=2)
draw_word_histogram(sam, "SAM: Bi-gramas más comunes")
```



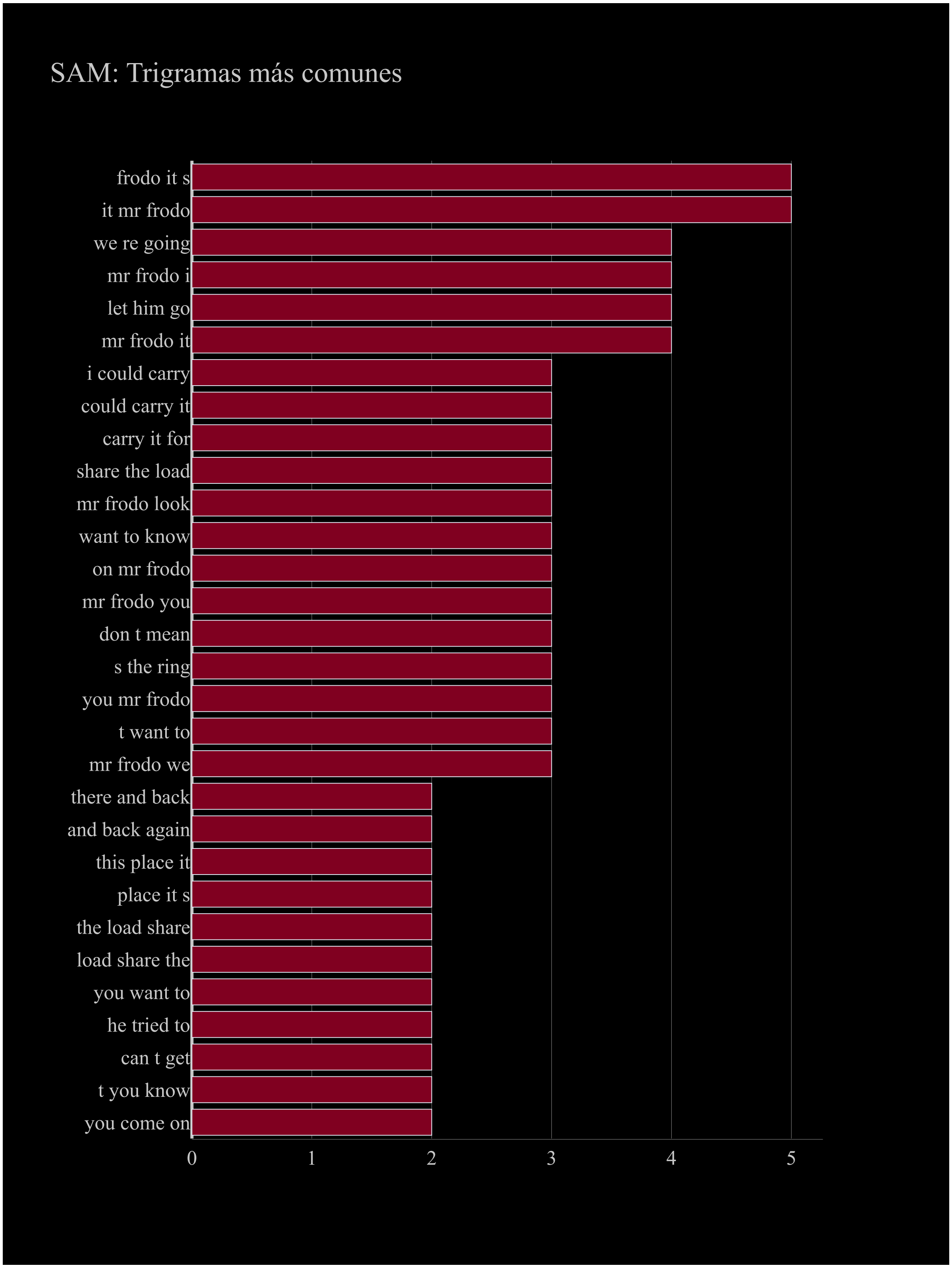
Frodo suele decir "lo siento".

In [11]:
```
frodo = newdf[newdf.char=="FRODO"].dialog.apply(preprocessText, ngram_order=3)
draw_word_histogram(frodo, "FRODO: Tri-gramas más comunes")
```



Y Sam suele decir "Yo podría llevarlo".

In [12]:
```
sam = newdf[newdf.char=="SAM"].dialog.apply(preprocessText, ngram_order=3)
draw_word_histogram(sam, "SAM: Trigramas más comunes")
```

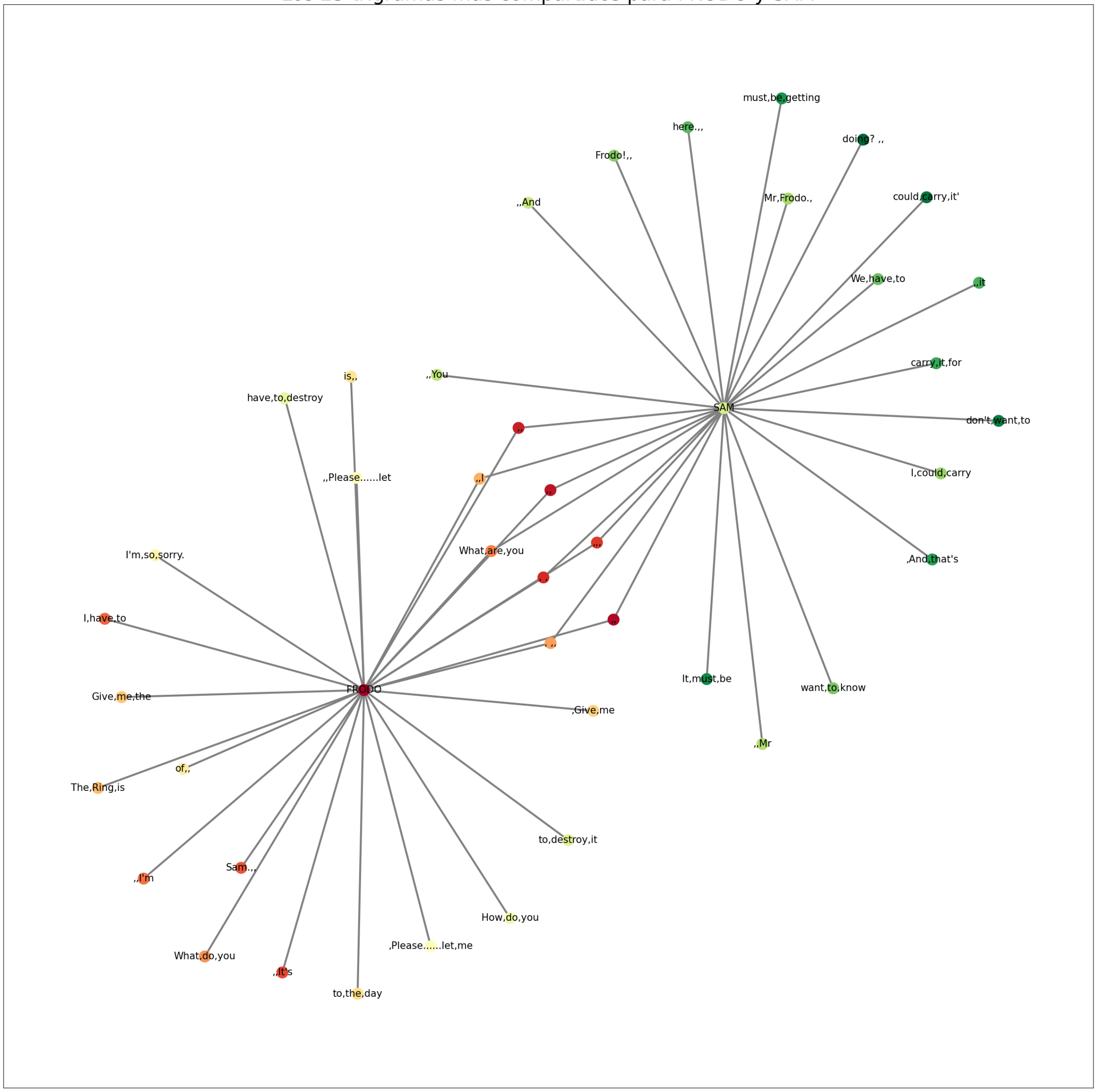


Las diferencias entre los bigramas y trigramas más comunes pueden ser características informativas para distinguir entre caracteres al construir nuestro modelo.

In [14]:
```
# adapted from https://www.kaggle.com/ash316/what-is-the-rock-cooking-ensembling-netwo
train_df = newdf
def generate_ngrams(text, n):
    words = text.split(' ')
```

```python
    iterations = len(words) - n + 1
    for i in range(iterations):
       yield words[i:i + n]
def net_diagram(*chars):
    ngrams = {}
    for title in train_df[train_df.char==chars[0]]['dialog']:
            for ngram in generate_ngrams(title, 3):
                ngram = ','.join(ngram)
                if ngram in ngrams:
                    ngrams[ngram] += 1
                else:
                    ngrams[ngram] = 1

    ngrams_mws_df = pd.DataFrame.from_dict(ngrams, orient='index')
    ngrams_mws_df.columns = ['count']
    ngrams_mws_df['char'] = chars[0]
    ngrams_mws_df.reset_index(level=0, inplace=True)

    ngrams = {}
    for title in train_df[train_df.char==chars[1]]['dialog']:
            for ngram in generate_ngrams(title, 3):
                ngram = ','.join(ngram)
                if ngram in ngrams:
                    ngrams[ngram] += 1
                else:
                    ngrams[ngram] = 1

    ngrams_mws_df1 = pd.DataFrame.from_dict(ngrams, orient='index')
    ngrams_mws_df1.columns = ['count']
    ngrams_mws_df1['char'] = chars[1]
    ngrams_mws_df1.reset_index(level=0, inplace=True)
    char1=ngrams_mws_df.sort_values('count',ascending=False)[:25]
    char2=ngrams_mws_df1.sort_values('count',ascending=False)[:25]
    df_final=pd.concat([char1,char2])
    g = nx.from_pandas_edgelist(df_final,source='char',target='index')
    cmap = plt.cm.RdYlGn
    colors = [n for n in range(len(g.nodes()))]
    k = 0.35
    pos=nx.spring_layout(g, k=k)
    nx.draw_networkx(g,
                     pos,
                     node_size=300,
                     cmap = cmap,
                     node_color=colors,
                     edge_color='grey',
                     font_size=15,
                     width=3)
    plt.title("Los 25 trigramas más compartidos para %s y %s" %(chars[0],chars[1]), fo
    plt.gcf().set_size_inches(30,30)
    plt.show()
    plt.savefig('network.png')
net_diagram('FRODO','SAM')
```

Los 25 trigramas más compartidos para FRODO y SAM



<Figure size 640x480 with 0 Axes>

Construir un modelo usando Keras.

Este código es una adaptación de un modelo de clasificación de texto utilizando la biblioteca Keras.

In [15]:

```
# adapted from https://www.kaggle.com/nzw0301/simple-keras-fasttext-val-loss-0-31
from keras.utils import pad_sequences
from sklearn.utils import class_weight

def create_docs(df, n_gram_max=4):
    def add_ngram(q, n_gram_max):
        ngrams = []
        for n in range(1, n_gram_max+1):
            for w_index in range(len(q)-n+1):
                ngrams.append('--'.join(q[w_index:w_index+n]))
        return q + ngrams
```

```
    docs = []
    for doc in df.dialog:
        doc = preprocess(doc).split()
        docs.append(' '.join(add_ngram(doc, n_gram_max)))
    return docs

min_count = 15
docs = create_docs(newdf)
tokenizer = Tokenizer(lower=True, filters='')
tokenizer.fit_on_texts(docs)
num_words = sum([1 for _, v in tokenizer.word_counts.items() if v >= min_count])
tokenizer = Tokenizer(num_words=num_words, lower=True, filters='')
tokenizer.fit_on_texts(docs)
docs = tokenizer.texts_to_sequences(docs)
maxlen = None
docs = pad_sequences(sequences=docs, maxlen=maxlen)
input_dim = np.max(docs) + 1
embedding_dims = 20

def create_model(embedding_dims=20, optimizer='adam'):
    model = Sequential()
    model.add(Embedding(input_dim=input_dim, output_dim=embedding_dims))
    model.add(GlobalAveragePooling1D())
    model.add(Dense(10, activation='softmax'))

    model.compile(loss='categorical_crossentropy',
        optimizer=optimizer,
        metrics=['accuracy'])
    return model

epochs = 20
x_train, x_test, y_train, y_test = train_test_split(docs, y, test_size=0.2)

# Flatten y_train to a 1-dimensional array
y_train_flat = np.argmax(y_train, axis=1)

# Calculate class weights manually
class_labels = np.unique(y_train_flat)
class_counts = np.bincount(y_train_flat)
total_samples = np.sum(class_counts)
class_weights = total_samples / (len(class_labels) * class_counts)
# Convert class weights to a dictionary
class_weights_dict = dict(enumerate(class_weights))

model = create_model()
hist = model.fit(x_train, y_train,
                batch_size=16,
                validation_data=(x_test, y_test),
                epochs=epochs,
                class_weight=class_weights_dict,
                callbacks=[EarlyStopping(patience=2, monitor='val_loss')])
```

```
Epoch 1/20
69/69 [==============================] - 1s 6ms/step - loss: 2.3035 - accuracy: 0.096
5 - val_loss: 2.2996 - val_accuracy: 0.1055
Epoch 2/20
69/69 [==============================] - 0s 4ms/step - loss: 2.3021 - accuracy: 0.121
1 - val_loss: 2.2998 - val_accuracy: 0.1273
Epoch 3/20
69/69 [==============================] - 0s 3ms/step - loss: 2.3011 - accuracy: 0.144
8 - val_loss: 2.2978 - val_accuracy: 0.1200
Epoch 4/20
69/69 [==============================] - 0s 4ms/step - loss: 2.3003 - accuracy: 0.164
8 - val_loss: 2.2989 - val_accuracy: 0.1455
Epoch 5/20
69/69 [==============================] - 0s 4ms/step - loss: 2.2989 - accuracy: 0.164
8 - val_loss: 2.2968 - val_accuracy: 0.1418
Epoch 6/20
69/69 [==============================] - 0s 4ms/step - loss: 2.2978 - accuracy: 0.167
6 - val_loss: 2.2960 - val_accuracy: 0.1818
Epoch 7/20
69/69 [==============================] - 0s 3ms/step - loss: 2.2964 - accuracy: 0.191
3 - val_loss: 2.2959 - val_accuracy: 0.2364
Epoch 8/20
69/69 [==============================] - 0s 4ms/step - loss: 2.2950 - accuracy: 0.203
1 - val_loss: 2.2931 - val_accuracy: 0.2618
Epoch 9/20
69/69 [==============================] - 0s 3ms/step - loss: 2.2923 - accuracy: 0.237
7 - val_loss: 2.2929 - val_accuracy: 0.1927
Epoch 10/20
69/69 [==============================] - 0s 3ms/step - loss: 2.2909 - accuracy: 0.271
4 - val_loss: 2.2921 - val_accuracy: 0.2691
Epoch 11/20
69/69 [==============================] - 0s 3ms/step - loss: 2.2872 - accuracy: 0.264
1 - val_loss: 2.2897 - val_accuracy: 0.2545
Epoch 12/20
69/69 [==============================] - 0s 3ms/step - loss: 2.2838 - accuracy: 0.313
3 - val_loss: 2.2878 - val_accuracy: 0.3091
Epoch 13/20
69/69 [==============================] - 0s 4ms/step - loss: 2.2805 - accuracy: 0.297
8 - val_loss: 2.2856 - val_accuracy: 0.2836
Epoch 14/20
69/69 [==============================] - 0s 4ms/step - loss: 2.2770 - accuracy: 0.304
2 - val_loss: 2.2801 - val_accuracy: 0.3200
Epoch 15/20
69/69 [==============================] - 0s 4ms/step - loss: 2.2722 - accuracy: 0.343
4 - val_loss: 2.2783 - val_accuracy: 0.3091
Epoch 16/20
69/69 [==============================] - 0s 4ms/step - loss: 2.2683 - accuracy: 0.292
3 - val_loss: 2.2761 - val_accuracy: 0.2836
Epoch 17/20
69/69 [==============================] - 0s 4ms/step - loss: 2.2636 - accuracy: 0.295
1 - val_loss: 2.2709 - val_accuracy: 0.2909
Epoch 18/20
69/69 [==============================] - 0s 4ms/step - loss: 2.2575 - accuracy: 0.312
4 - val_loss: 2.2717 - val_accuracy: 0.3091
Epoch 19/20
69/69 [==============================] - 0s 4ms/step - loss: 2.2522 - accuracy: 0.307
8 - val_loss: 2.2629 - val_accuracy: 0.3091
Epoch 20/20
69/69 [==============================] - 0s 4ms/step - loss: 2.2463 - accuracy: 0.319
7 - val_loss: 2.2630 - val_accuracy: 0.3127
```

El código anterior prepra los dialogos para que el modelo pueda aprender sobre ellos y asi enseña al modelo a clasificar los textos que se usaron de ejemplo para después se hacen pruebas para saber que tan bien aprendió.

Intentemos usar CountVectorizer() y TfidfVectorizer()

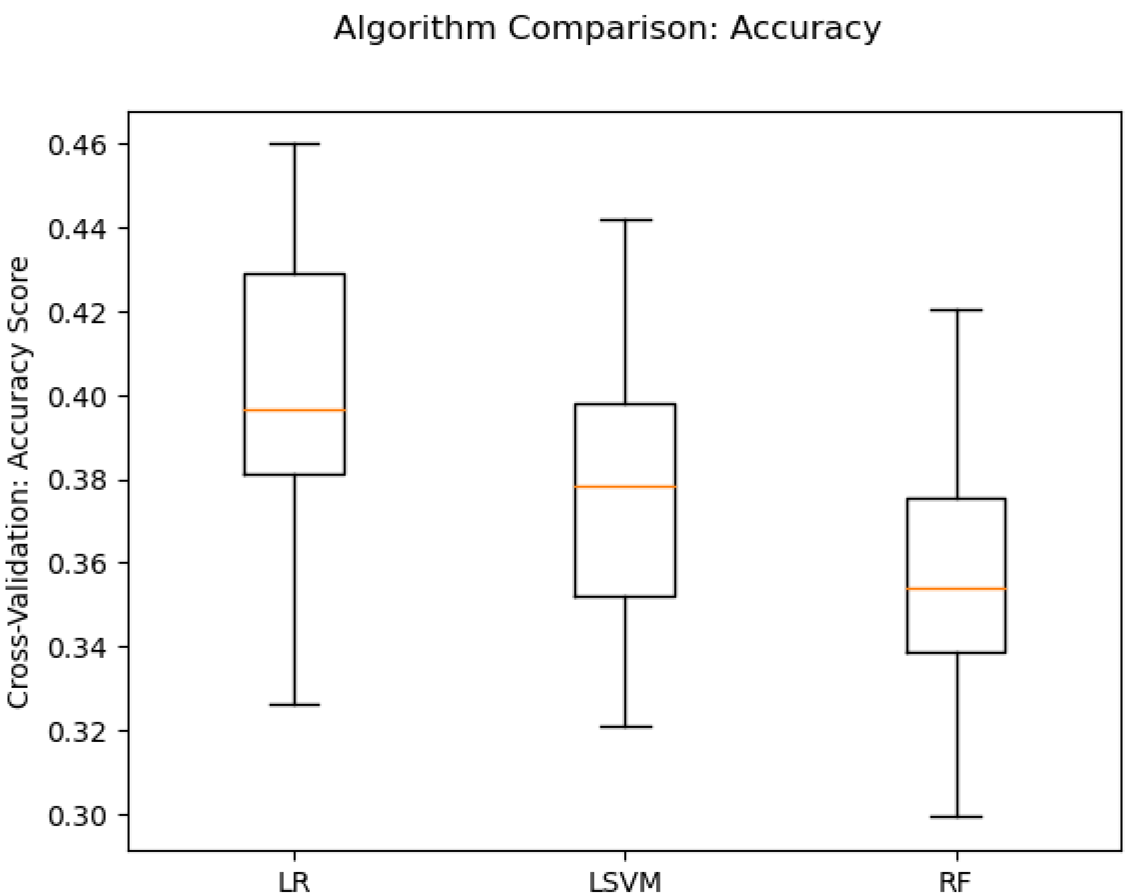
In [17]:
```
script2 = script[['char','dialog']]

def separateDf(df,column,value):
    separated = df[column] == value
    separated = df[separated]
    return separated

FRODO2 = separateDf(script2,'char',"FRODO")
SAM2 = separateDf(script2,'char',"SAM")
GANDALF2 = separateDf(script2,'char',"GANDALF")
ARAGORN2 = separateDf(script2,'char',"ARAGORN")
GOLLUM2 = separateDf(script2,'char',"GOLLUM")
SMEAGOL2 = separateDf(script2,'char',"SMEAGOL")
PIPPIN2 = separateDf(script2,'char',"PIPPIN")
MERRY2 = separateDf(script2,'char',"MERRY")
ARWEN2 = separateDf(script2,'char',"ARWEN")
ORC2 = separateDf(script2,'char',"ORC")

newdf = pd.concat([FRODO2,SAM2,GANDALF2,ARAGORN2,GOLLUM2,SMEAGOL2,PIPPIN2,MERRY2,ARWEN

X = newdf['dialog']
y = newdf['char']

vect = CountVectorizer()
X2 = vect.fit_transform(X)
X2 = X2.astype('float')
lb = LabelEncoder()
y2 = lb.fit_transform(y)

tfidf = TfidfVectorizer(binary=True)
X3 = tfidf.fit_transform(X)
X3 = X3.astype('float')
lb = LabelEncoder()
y3 = lb.fit_transform(y)
```

Con CountVectorizer() obtenemos una precisión de ~25 % cuando tratamos de identificar cuál de los 9 caracteres diferentes.

In [21]:
```
# adapted from https://machinelearningmastery.com/compare-machine-learning-algorithms-
script2 = script[['char','dialog']]
def compareAccuracy(a, b):
    print('\nCompare Multiple Classifiers: \n')
    print('K-Fold Cross-Validation Accuracy: \n')
    names = []
    models = []
    resultsAccuracy = []
    models.append(('LR', LogisticRegression(class_weight='balanced')))
    models.append(('LSVM', LinearSVC(class_weight='balanced')))
    models.append(('RF', RandomForestClassifier(class_weight='balanced')))
    for name, model in models:
```

```
        model.fit(a, b)
        kfold = model_selection.KFold(n_splits=10, shuffle=True) # Set shuffle=True
        accuracy_results = model_selection.cross_val_score(model, a,b, cv=kfold, scori
        resultsAccuracy.append(accuracy_results)
        names.append(name)
        accuracyMessage = "%s: %f (%f)" % (name, accuracy_results.mean(), accuracy_res
        print(accuracyMessage)

    # Boxplot
    fig = plt.figure()
    fig.suptitle('Algorithm Comparison: Accuracy')
    ax = fig.add_subplot(111)
    plt.boxplot(resultsAccuracy)
    ax.set_xticklabels(names)
    ax.set_ylabel('Cross-Validation: Accuracy Score')
    plt.show()

def defineModels():
    print('\nLR = LogisticRegression')
    print('LSVM = LinearSVM')
    print('RF = RandomForestClassifier')

compareAccuracy(X2, y2)
defineModels()
```

```
Compare Multiple Classifiers:

K-Fold Cross-Validation Accuracy:

LR: 0.400666 (0.040141)
LSVM: 0.377245 (0.033251)
RF: 0.359748 (0.034585)
```



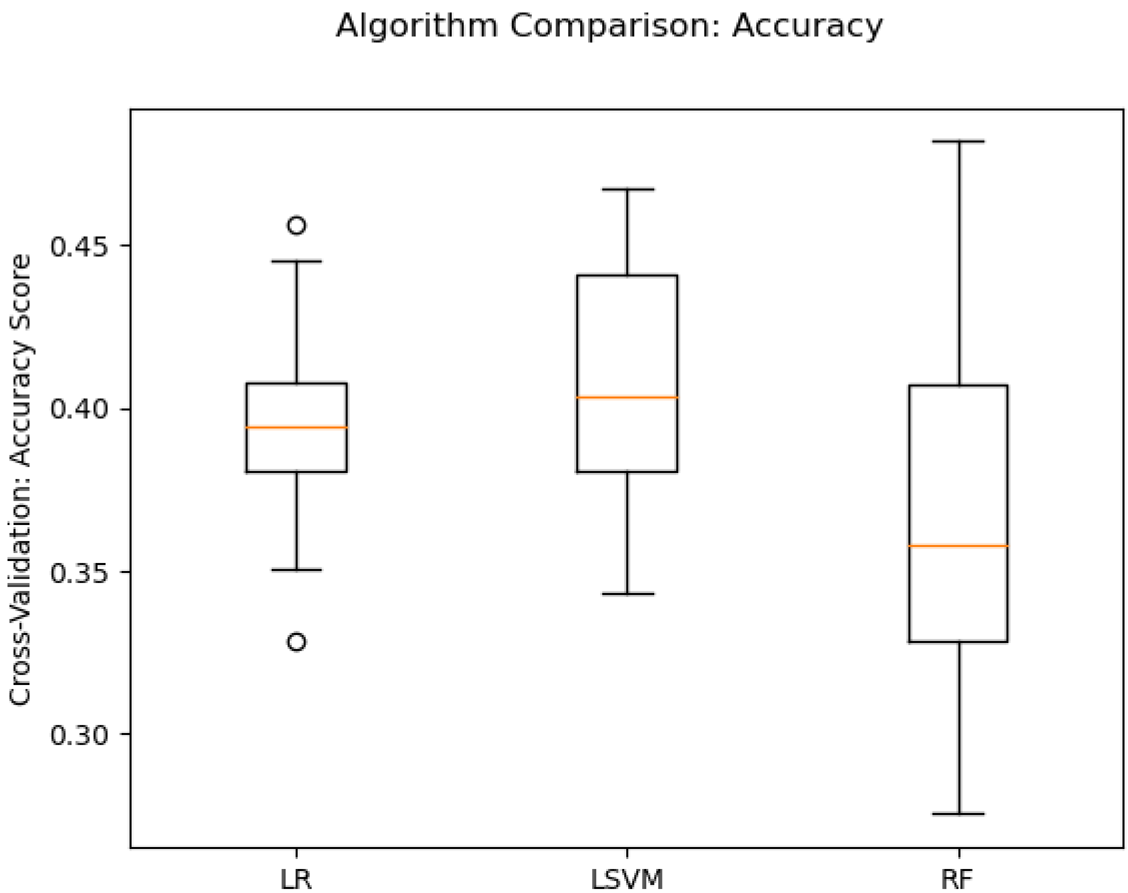
```
LR = LogisticRegression
LSVM = LinearSVM
RF = RandomForestClassifier
```

El código muestra una tecnica que se llama validación cruzada de K-fold para mostrar la clasificacion de los datos y muestra el diagrma de caja para compara el rendimiento del modelo.

Obtenemos alrededor del 25% de precisión con TfidfVectorizer() también.

In [22]:
```
compareAccuracy(X3,y3)
defineModels()
```

```
Compare Multiple Classifiers:

K-Fold Cross-Validation Accuracy:

LR: 0.394711 (0.036743)
LSVM: 0.410039 (0.038996)
RF: 0.369311 (0.060610)
```




```
LR = LogisticRegression
LSVM = LinearSVM
RF = RandomForestClassifier
```

In [27]:
```
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=1, train_sizes=np.linspace(.1, 1.0, 5)):
    """
    Plots a learning curve. http://scikit-learn.org/stable/modules/learning_curve.html
    """
    plt.figure()
```

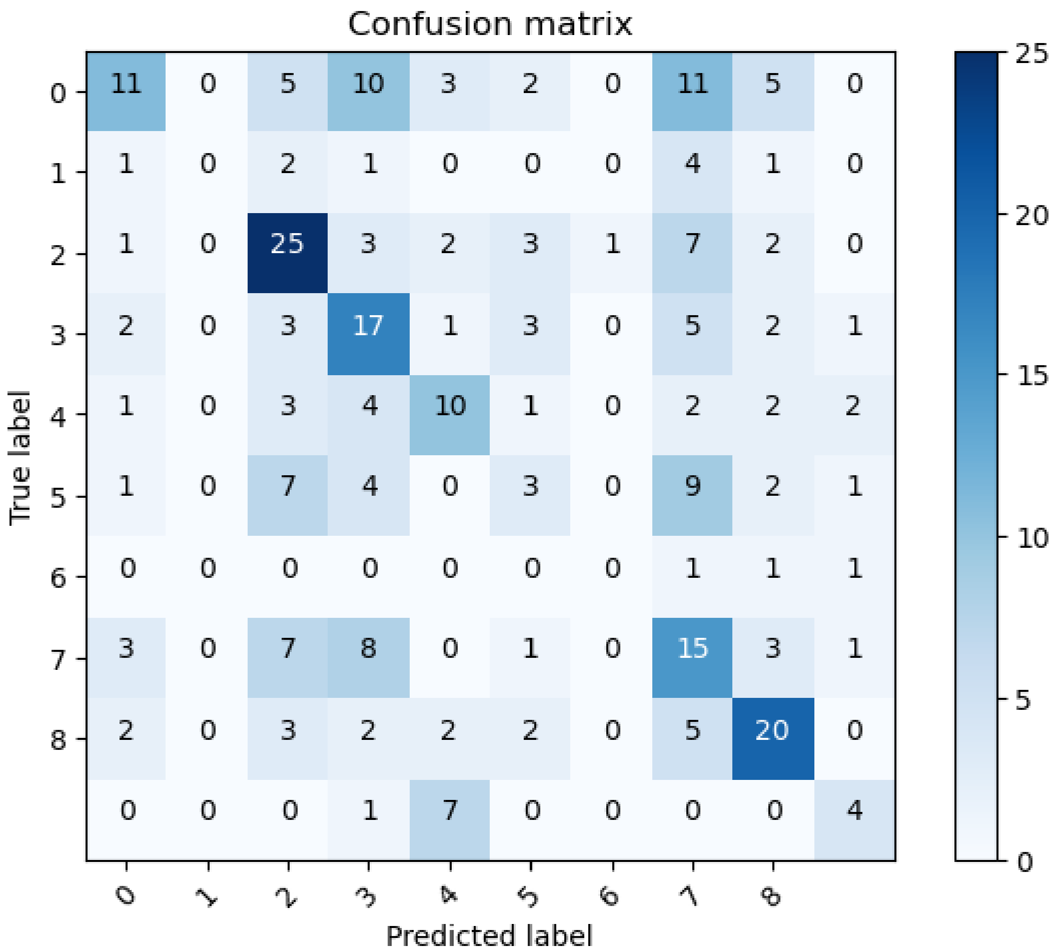
```
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()
    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")
    plt.legend(loc="best")
    return plt

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    """
    http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

def evaluateRandomForestClassifier(a, b, c, d):
    model = RandomForestClassifier(class_weight='balanced')
    model.fit(a, b)
    kfold = model_selection.KFold(n_splits=10, shuffle=True, random_state=7)
    accuracy = model_selection.cross_val_score(model, a,b, cv=kfold, scoring='accuracy
```

```
    mean = accuracy.mean()
    stdev = accuracy.std()
    print('RandomForestClassifier - Accuracy: %s (%s)' % (mean, stdev),'\n')
    prediction = model.predict(c)
    cnf_matrix = confusion_matrix(d, prediction)
    np.set_printoptions(precision=2)
    class_names = dict_characters
    plt.figure()
    plot_confusion_matrix(cnf_matrix, classes=class_names,title='Confusion matrix')
    plt.figure()
    plot_learning_curve(model, 'Learning Curve For RandomForestClassifier', a, b, (0,1
    print('\n',dict_characters)
```

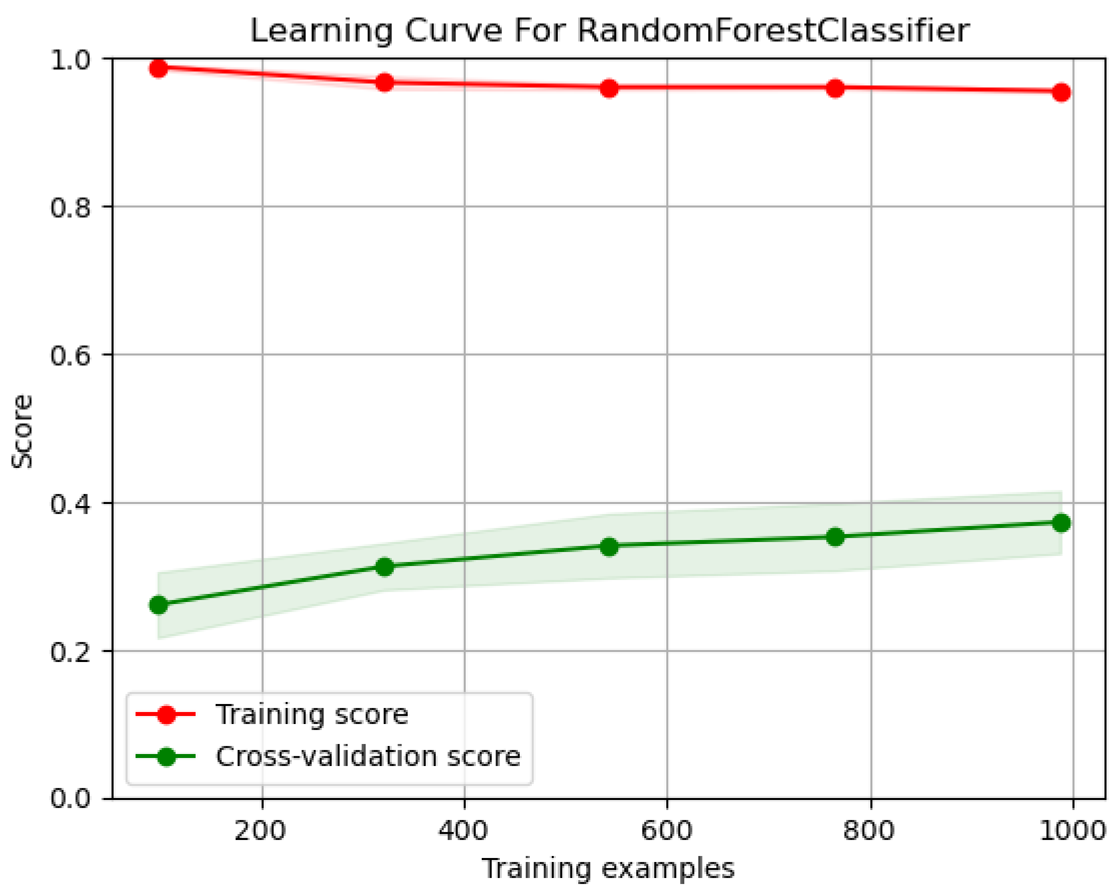
En este paso el codigo es un ejemplo de cómo evaluar el rendimiento de un método para enseñar a una computadora a clasificar cosas.

In [29]:
```
X_train, X_test, y_train, y_test = train_test_split(X3, y3, test_size=0.2)
dict_characters = {0: 'Frodo', 1: 'Sam', 2: 'Gandalf', 3:'Aragorn', 4: 'Gollum', 5: 'S
evaluateRandomForestClassifier(X_train, y_train, X_test, y_test)
```

```
RandomForestClassifier - Accuracy: 0.36704753961634695 (0.032189982177120456)

Confusion matrix, without normalization
[[11  0  5 10  3  2  0 11  5  0]
 [ 1  0  2  1  0  0  0  4  1  0]
 [ 1  0 25  3  2  3  1  7  2  0]
 [ 2  0  3 17  1  3  0  5  2  1]
 [ 1  0  3  4 10  1  0  2  2  2]
 [ 1  0  7  4  0  3  0  9  2  1]
 [ 0  0  0  0  0  0  0  1  1  1]
 [ 3  0  7  8  0  1  0 15  3  1]
 [ 2  0  3  2  2  2  0  5 20  0]
 [ 0  0  0  1  7  0  0  0  0  4]]

 {0: 'Frodo', 1: 'Sam', 2: 'Gandalf', 3: 'Aragorn', 4: 'Gollum', 5: 'Smeagol', 6: 'Pi
ppen', 7: 'Merry', 8: 'Arwen'}
```

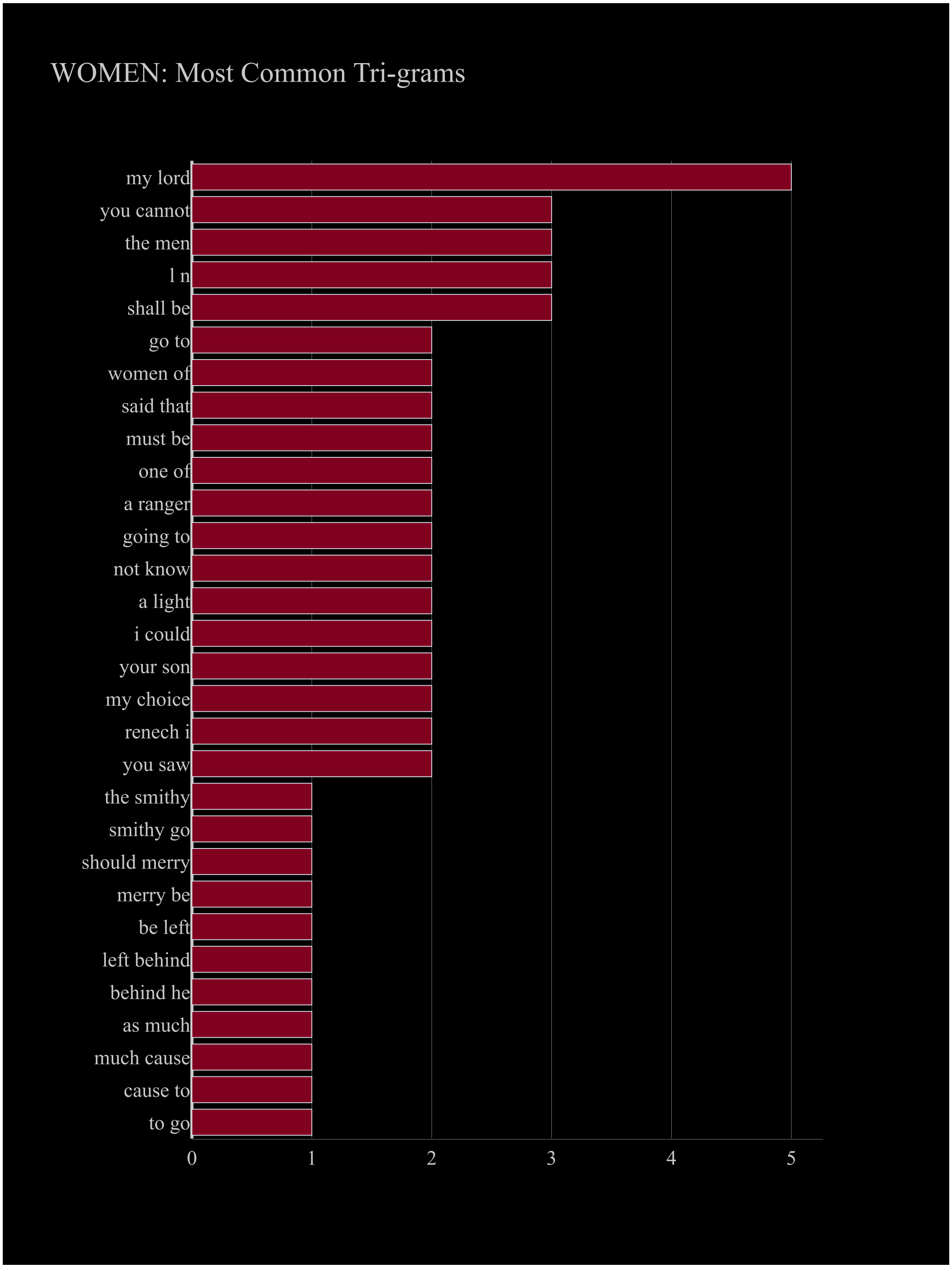


<Figure size 640x480 with 0 Axes>



Las mujeres en El Señor de los Anillos tienden a decir mucho "Mi Señor".

In [30]:
```
script3 = script2
script3['gender'] = np.where((script3['char']=='EOWYN') | (script3['char']=='ARWEN'),
lineCounts2 = script3['gender'].value_counts()
script4 = script3[['gender','dialog']]
MAN2 = separateDf(script4,'gender',"MAN")
WOMAN2 = separateDf(script4,'gender',"WOMAN")
newdf2 = pd.concat([MAN2,WOMAN2])
newdf2 = shuffle(newdf2)

men = script4[script4.gender=="WOMAN"].dialog.apply(preprocessText, ngram_order=2)
draw_word_histogram(men, "WOMEN: Most Common Tri-grams")
```



Nuevamente intentaré construir un modelo usando Keras.

In [32]:
```
newdf2 = newdf2[newdf2['dialog'].notnull()]
a2c = {"MAN":0,"WOMAN":1}
docs = create_docs(newdf2)

min_count = 15
docs = create_docs(newdf2)
```

```
tokenizer = Tokenizer(lower=True, filters='')
tokenizer.fit_on_texts(docs)
num_words = sum([1 for _, v in tokenizer.word_counts.items() if v >= min_count])
tokenizer = Tokenizer(num_words=num_words, lower=True, filters='')
tokenizer.fit_on_texts(docs)
docs = tokenizer.texts_to_sequences(docs)
maxlen = None
docs = pad_sequences(sequences=docs, maxlen=maxlen)
input_dim = np.max(docs) + 1
embedding_dims = 20

y = np.array([a2c[a] for a in newdf2.gender])
y = to_categorical(y)
x_train, x_test, y_train, y_test = train_test_split(docs, y, test_size=0.2)

def create_model2(embedding_dims=20, optimizer='adam'):
    model = Sequential()
    model.add(Embedding(input_dim=input_dim, output_dim=embedding_dims))
    model.add(GlobalAveragePooling1D())
    model.add(Dense(2, activation='softmax'))

    model.compile(loss='categorical_crossentropy',
                  optimizer=optimizer,
                  metrics=['accuracy'])
    return model

model = create_model2()
hist = model.fit(x_train, y_train,
                 batch_size=16,
                 validation_data=(x_test, y_test),
                 epochs=epochs,
                 class_weight=class_weight.compute_class_weight('balanced', np.unique(r
                 callbacks=[EarlyStopping(patience=2, monitor='val_loss')]
                 )
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[32], line 38
     31     return model
     33 model = create_model2()
     34 hist = model.fit(x_train, y_train,
     35                  batch_size=16,
     36                  validation_data=(x_test, y_test),
     37                  epochs=epochs,
---> 38                  class_weight=class_weight.compute_class_weight('balanced', n
p.unique(newdf2.gender), newdf2.gender),
     39                  callbacks=[EarlyStopping(patience=2, monitor='val_loss')]
     40                  )

TypeError: compute_class_weight() takes 1 positional argument but 3 were given
```

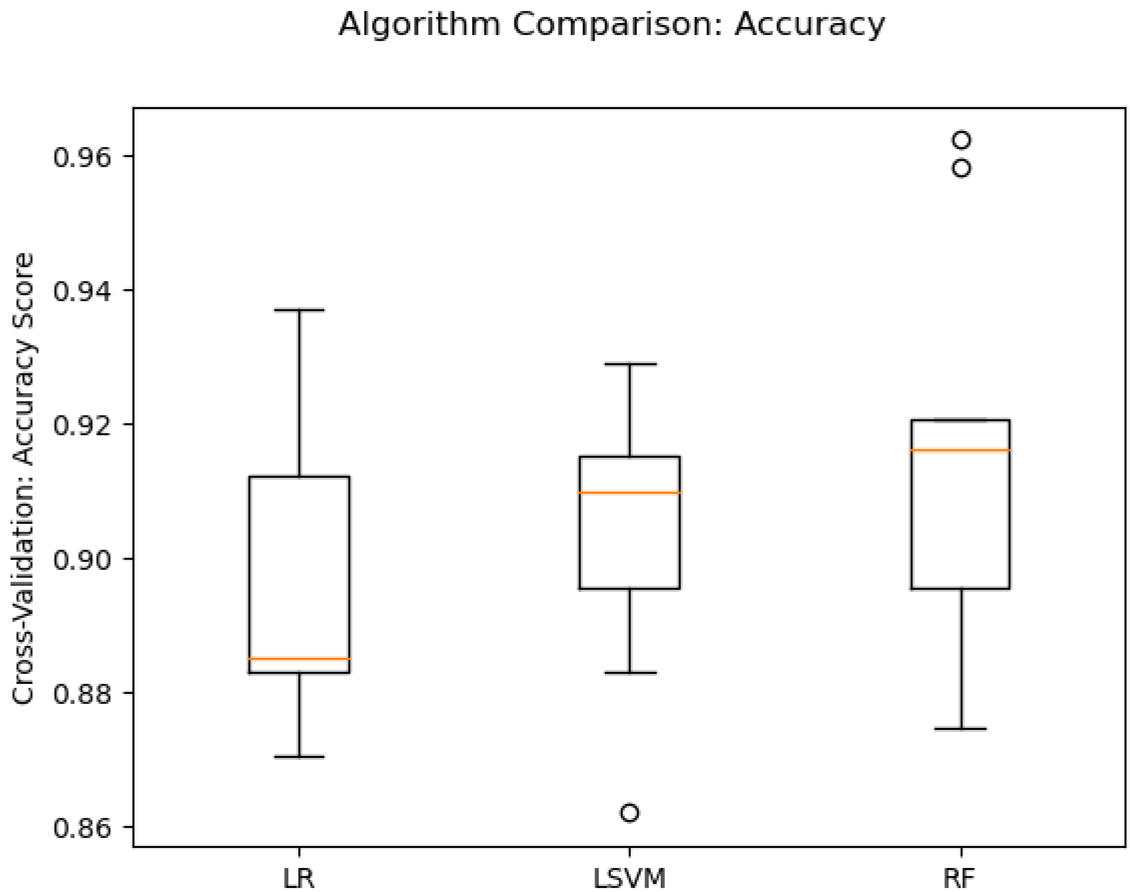
Este código muestra cómo crear y entrenar un modelo de aprendizaje automático para clasificar textos por género utilizando la biblioteca Keras. Primero, el código prepara los datos para que la computadora pueda entenderlos mejor. Luego, el código crea un modelo, y le enseña a clasificar los textos por género utilizando ejemplos. Finalmente, el código prueba el modelo para ver qué tan bien aprendió. Si el modelo no está aprendiendo lo suficientemente rápido, el código detiene el entrenamiento para que no pierda tiempo.

Eso parece haber funcionado razonablemente bien. Intentemos usar CountVectorizer() y TfidfVectorizer() de scikit-learn ahora también.

Con CountVectorizer() obtenemos ~90% de precisión cuando tratamos de identificar el género del hablante de cada línea en el texto de El Señor de los Anillos.

In [34]:
```
X = newdf2['dialog'].values.astype('U')
y = newdf2['gender'].values.astype('U')

vect = CountVectorizer()
X2 = vect.fit_transform(X)
X2 = X2.astype('float')
lb = LabelEncoder()
y2 = lb.fit_transform(y)

tfidf = TfidfVectorizer(binary=True)
X3 = tfidf.fit_transform(X)
X3 = X3.astype('float')
lb = LabelEncoder()
y3 = lb.fit_transform(y)

compareAccuracy(X2,y2)
defineModels()
```

```
Compare Multiple Classifiers:

K-Fold Cross-Validation Accuracy:

LR: 0.896626 (0.022798)
LSVM: 0.902892 (0.018487)
RF: 0.915448 (0.026903)
```

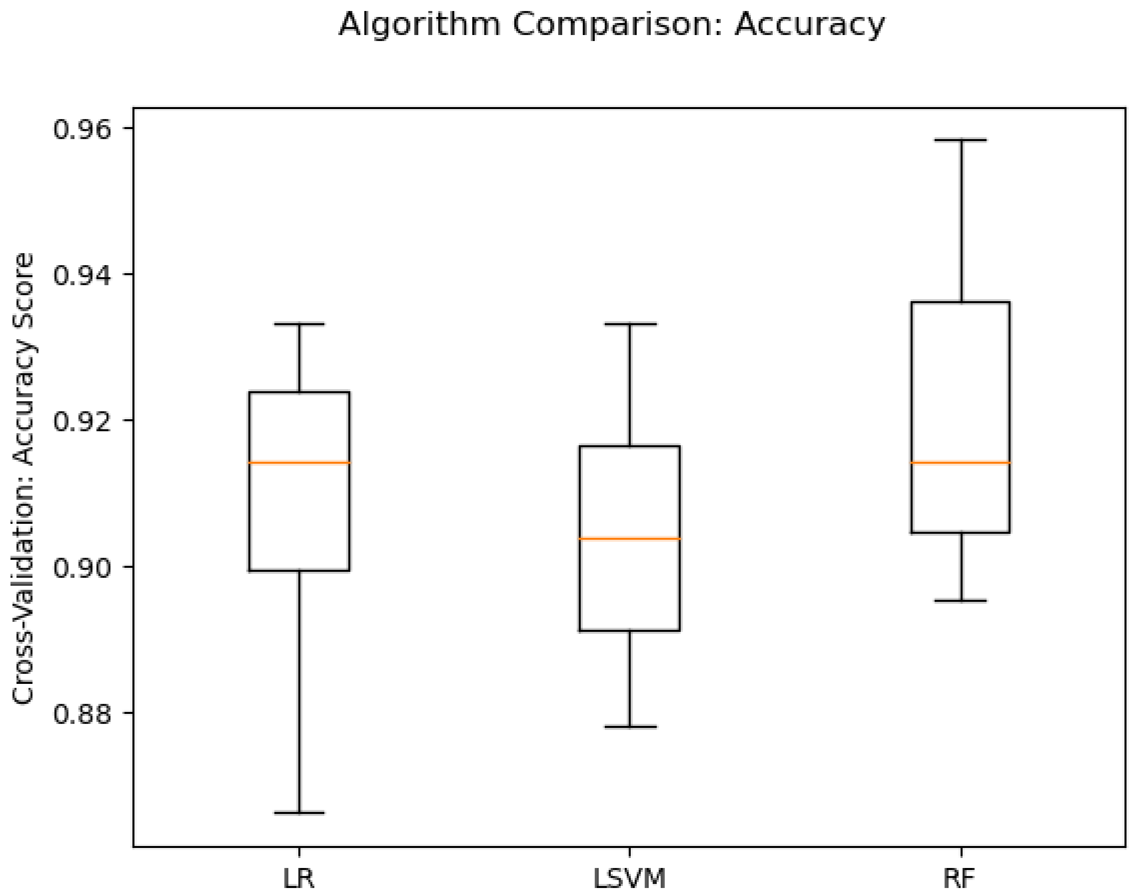



```
LR = LogisticRegression
LSVM = LinearSVM
RF = RandomForestClassifier
```

Obtenemos alrededor del 90% de precisión con TfidfVectorizer() también.

In [35]:
```
compareAccuracy(X3,y3)
defineModels()
```

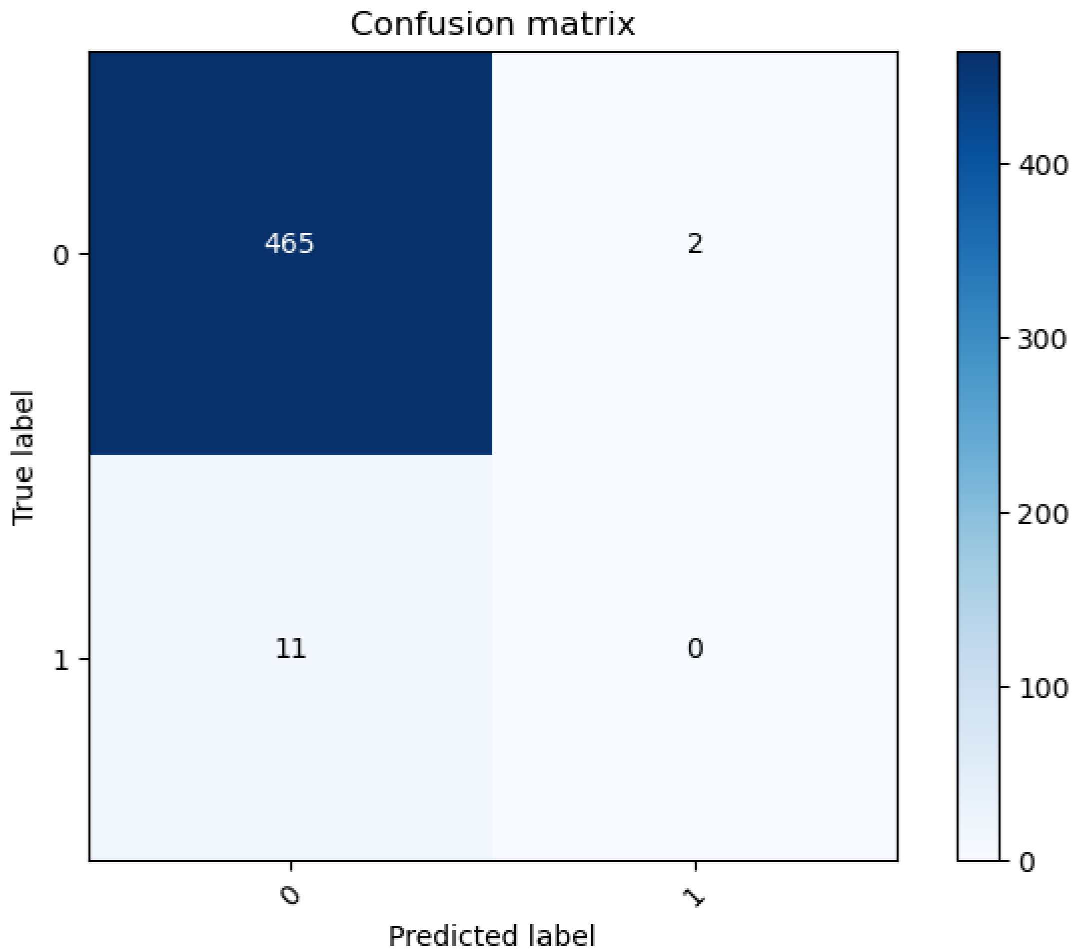
```
Compare Multiple Classifiers:

K-Fold Cross-Validation Accuracy:

LR: 0.909587 (0.020765)
LSVM: 0.903715 (0.018966)
RF: 0.920463 (0.019828)
```



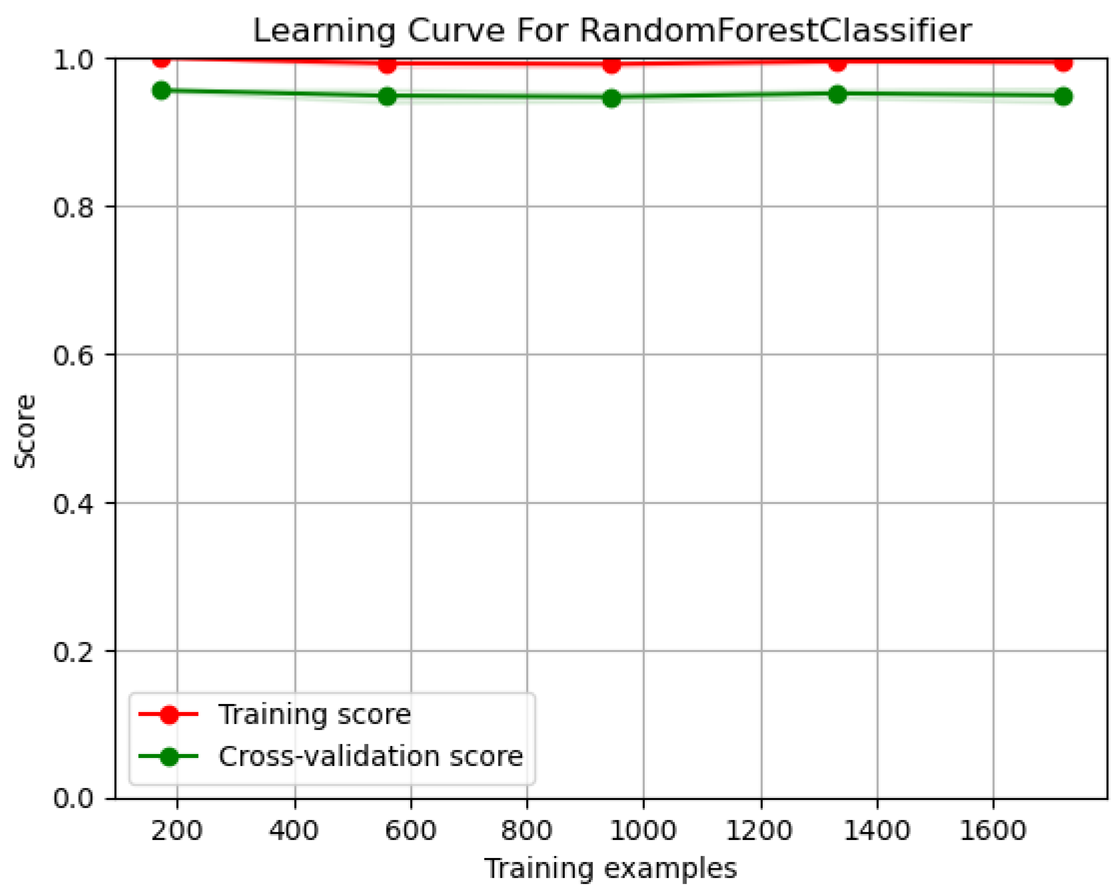

```
LR = LogisticRegression
LSVM = LinearSVM
RF = RandomForestClassifier
```

In [36]:
```
X_train, X_test, y_train, y_test = train_test_split(X3, y3, test_size=0.2)
dict_characters= dict_characters = {0: 'MEN', 1: 'WOMEN'}
evaluateRandomForestClassifier(X_train, y_train, X_test, y_test)
```

```
RandomForestClassifier - Accuracy: 0.9523832897033158 (0.01978535980779089)

Confusion matrix, without normalization
[[465   2]
 [ 11   0]]

 {0: 'MEN', 1: 'WOMEN'}
```



<Figure size 640x480 with 0 Axes>



This RandomForestClassifier() seems to work reasonably well.

In [37]:
```
model = RandomForestClassifier(class_weight='balanced')
model.fit(X3, y3)
kfold = model_selection.KFold(n_splits=10, shuffle=True, random_state=7)
accuracy_results = model_selection.cross_val_score(model, X3, y3, cv=kfold, scoring='a
accuracyMessage = "%s: %f (%f)" % ("RandomForestClassifier", accuracy_results.mean(),
print(accuracyMessage)
eli5.show_prediction(model,doc='X3',vec=vect,targets=y2,top=10)
```

```
  Cell In[37], line 4
    accuracy_results = model_selection.cross_val_score(model, X3, y3, cv=kfold, scori
ng='a

^
SyntaxError: unterminated string literal (detected at line 4)
```

Este código está creando una herramienta llamada RandomForestClassifier para hacer predicciones basadas en datos. Luego, se ajusta esta herramienta a los datos para que pueda aprender de ellos. Después, se utiliza la validación cruzada para evaluar qué tan bien está haciendo predicciones la herramienta.

In [38]:
```
gender = ["Male","Female"]
race = ["Men",'Hobbits','Elves','Dwarves','Ainur','Orcs','Half-elven','Dragons']

male = [menCountM, hobbitsCountM, elvesCountM, dwarvesCountM,ainurCountM, halfelvenCou
female = [menCountF, hobbitsCountF, elvesCountF, dwarvesCountM,ainurCountF, halfelven(
data = {'race' : race,
        'Male'   : male,
        'Female'   : female}

trace1 = go.Bar(
    x=data['race'],
    y=data['Male'],
    name='# de Personajes Masculinos',
    marker = dict(color = 'rgba(0, 0, 0, 1)', #0, 0, 255, 0.8
                             line=dict(color='rgb(0,0,0)',width=1.5))
)
trace2 = go.Bar(
    x=data['race'],
    y=data['Female'],
    name='# de Personajes Femeninos',
    marker = dict(color = 'rgba(255,0,255,1)',
                             line=dict(color='rgb(0,0,0)',width=1.5))
)

data = [trace1, trace2]
layout = go.Layout(title='# de Personajes por Género',barmode="group")

fig = go.Figure(data=data, layout=layout)
iplot(fig)
```

Conclusiones

El aprender el uso de las librerias keras ha sido algo importante porque te ayuda crear modelos de aprendizaje profundo, mediante el usos de redes neuranles, es lo que hicimos en el codigo en la parte donde se entrena al algoritmo para analizar los sentimientos de los dialogos de cada personaje, se uso en conjunto la libreria tensorflow y también con la ayuda de la biblioteca VaderSentiment que sirve para el análisis de texto que detecta la polaridad (por ejemplo, una opinión positiva o negativa) en este ejemplo analiza las palabras de los dialogos que le mandamos. Esto nos sirve para aprender a entrenar los modelos de aprendizaje automatico y ver cual nos combiene usar.