In [2]:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

In [3]:

data=pd.read\_json('Sarcasm\_Headlines\_Dataset\_v2.json', lines=True)

In [4]:

data.head()

Out[4]:

|  | **is\_sarcastic** | **headline** | **article\_link** |
| --- | --- | --- | --- |
| **0** | 1 | thirtysomething scientists unveil doomsday clo... | https://www.theonion.com/thirtysomething-scien... |
| **1** | 0 | dem rep. totally nails why congress is falling... | https://www.huffingtonpost.com/entry/donna-edw... |
| **2** | 0 | eat your veggies: 9 deliciously different recipes | https://www.huffingtonpost.com/entry/eat-your-... |
| **3** | 1 | inclement weather prevents liar from getting t... | https://local.theonion.com/inclement-weather-p... |
| **4** | 1 | mother comes pretty close to using word 'strea... | https://www.theonion.com/mother-comes-pretty-c... |

In [5]:

df=data.drop('article\_link', axis=1)

In [6]:

df.head()

Out[6]:

|  | **is\_sarcastic** | **headline** |
| --- | --- | --- |
| **0** | 1 | thirtysomething scientists unveil doomsday clo... |
| **1** | 0 | dem rep. totally nails why congress is falling... |
| **2** | 0 | eat your veggies: 9 deliciously different recipes |
| **3** | 1 | inclement weather prevents liar from getting t... |
| **4** | 1 | mother comes pretty close to using word 'strea... |

In [7]:

df.shape

Out[7]:

(28619, 2)

In [8]:

df.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 28619 entries, 0 to 28618

Data columns (total 2 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 is\_sarcastic 28619 non-null int64

1 headline 28619 non-null object

dtypes: int64(1), object(1)

memory usage: 447.3+ KB

In [9]:

df.isnull().sum()

Out[9]:

is\_sarcastic 0

headline 0

dtype: int64

In [10]:

sns.countplot(df['is\_sarcastic'].value\_counts())

C:\Users\jkong\anaconda3\lib\site-packages\seaborn\\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

warnings.warn(

Out[10]:

<AxesSubplot:xlabel='is\_sarcastic', ylabel='count'>



In [23]:

import nltk

from nltk.tokenize import word\_tokenize, sent\_tokenize

from nltk.corpus import stopwords

import spacy

In [15]:

nlp=spacy.load('en\_core\_web\_sm')

In [21]:

def preprocess(text):

doc=nlp(text)

lemmas=[token.lemma\_ for token in doc]

a\_lemmas=[lemma.lower() for lemma in lemmas if lemma.isalpha() and lemma not in stopwords.words('english')]

lemmatized\_text=' '.join(a\_lemmas)

return lemmatized\_text

In [24]:

cleaned\_text=[]

for text in df.headline:

cleaned\_text.append(preprocess(text))

df['clean\_text']=cleaned\_text

In [25]:

df.head()

Out[25]:

|  | **is\_sarcastic** | **headline** | **clean\_text** |
| --- | --- | --- | --- |
| **0** | 1 | thirtysomething scientists unveil doomsday clo... | thirtysomethe scientist unveil doomsday clock ... |
| **1** | 0 | dem rep. totally nails why congress is falling... | dem rep totally nail congress fall short gende... |
| **2** | 0 | eat your veggies: 9 deliciously different recipes | eat veggie deliciously different recipe |
| **3** | 1 | inclement weather prevents liar from getting t... | inclement weather prevent liar get work |
| **4** | 1 | mother comes pretty close to using word 'strea... | mother come pretty close use word streaming co... |

In [28]:

from wordcloud import WordCloud

plt.figure(figsize=(20, 10))

wc=WordCloud(width=1500, height=1000, max\_words=1000).generate(' '.join(word for word in df.clean\_text))

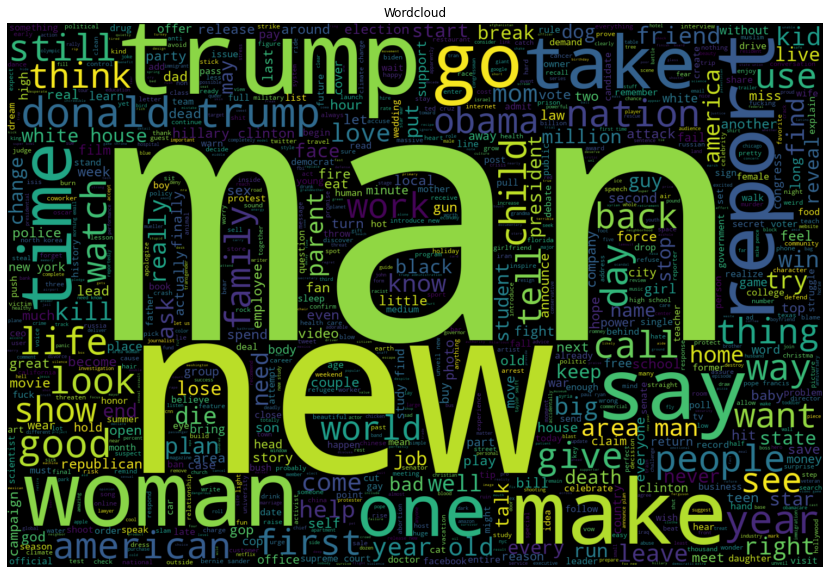
plt.axis('off')

plt.title('Wordcloud')

plt.imshow(wc, interpolation='bilinear')

Out[28]:

<matplotlib.image.AxesImage at 0x1c9d2a1b610>



In [30]:

X=df.clean\_text

y=df.is\_sarcastic

In [31]:

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import classification\_report

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear\_model import SGDClassifier

from sklearn.naive\_bayes import MultinomialNB

from sklearn.ensemble import VotingClassifier

from sklearn.ensemble import BaggingClassifier

from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier, ExtraTreesClassifier

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score,precision\_score,recall\_score,f1\_score,roc\_auc\_score

from sklearn.metrics import average\_precision\_score,roc\_auc\_score, roc\_curve, precision\_recall\_curve

from sklearn.preprocessing import LabelEncoder

from sklearn.feature\_extraction.text import TfidfVectorizer

In [32]:

def print\_metrices(pred, true):

print(confusion\_matrix(true, pred))

print(classification\_report(true, pred,))

print('Accuracy: ', accuracy\_score(pred, true))

print('Precision: ', precision\_score(pred, true, average='weighted'))

print('Recall: ', recall\_score(pred, true, average='weighted'))

print('F1: ', f1\_score(pred, true, average='weighted'))

In [33]:

X\_train, X\_test, y\_train, y\_test=train\_test\_split(X, y, test\_size=0.3, random\_state=42)

In [38]:

tfidf=TfidfVectorizer(ngram\_range=(1,3))

X\_tfidf\_train=tfidf.fit\_transform(X\_train.tolist())

X\_tfidf\_test=tfidf.transform(X\_test.tolist())

In [39]:

X\_train.head()

Out[39]:

12170 american express offer month paternity materni...

28552 watch dolphin knock stand paddleboarder board

6883 man enjoy thing inform wrong

28387 jonathan lipnicki star young dark helmet space...

12932 publicist worry kanye west support trump damag...

Name: clean\_text, dtype: object

In [40]:

lr=LogisticRegression(class\_weight='balanced')

lr.fit(X\_tfidf\_train, y\_train)

Out[40]:

LogisticRegression(class\_weight='balanced')

In [41]:

y\_pred\_lr=lr.predict(X\_tfidf\_test)

print\_metrices(y\_pred\_lr, y\_test)

[[3452 1003]

[ 982 3149]]

precision recall f1-score support

0 0.78 0.77 0.78 4455

1 0.76 0.76 0.76 4131

accuracy 0.77 8586

macro avg 0.77 0.77 0.77 8586

weighted avg 0.77 0.77 0.77 8586

Accuracy: 0.768809690193338

Precision: 0.7687789348367672

Recall: 0.768809690193338

F1: 0.7687897104004202

In [42]:

clf\_nb=MultinomialNB()

clf\_nb.fit(X\_tfidf\_train, y\_train)

y\_pred\_nb=clf\_nb.predict(X\_tfidf\_test)

print\_metrices(y\_pred\_nb, y\_test)

[[3855 600]

[1251 2880]]

precision recall f1-score support

0 0.75 0.87 0.81 4455

1 0.83 0.70 0.76 4131

accuracy 0.78 8586

macro avg 0.79 0.78 0.78 8586

weighted avg 0.79 0.78 0.78 8586

Accuracy: 0.7844164919636618

Precision: 0.7971659705535645

Recall: 0.7844164919636618

F1: 0.7862969173200055

In [43]:

from sklearn.svm import LinearSVC

svc=LinearSVC(C=10, random\_state=42, class\_weight='balanced')

svc.fit(X\_tfidf\_train, y\_train)

y\_pred\_svc=svc.predict(X\_tfidf\_test)

print\_metrices(y\_pred\_svc, y\_test)

[[3563 892]

[ 904 3227]]

precision recall f1-score support

0 0.80 0.80 0.80 4455

1 0.78 0.78 0.78 4131

accuracy 0.79 8586

macro avg 0.79 0.79 0.79 8586

weighted avg 0.79 0.79 0.79 8586

Accuracy: 0.7908222688096902

Precision: 0.7908482768396079

Recall: 0.7908222688096902

F1: 0.7908337270995445

In [46]:

clf\_df=DecisionTreeClassifier(criterion='gini', splitter='best', max\_depth=6, random\_state=42)

clf\_df.fit(X\_tfidf\_train, y\_train)

y\_pred\_dt=clf\_dt.predict(X\_tfidf\_test)

print\_metrices(y\_pred\_dt, y\_test)

[[4295 160]

[3312 819]]

precision recall f1-score support

0 0.56 0.96 0.71 4455

1 0.84 0.20 0.32 4131

accuracy 0.60 8586

macro avg 0.70 0.58 0.52 8586

weighted avg 0.70 0.60 0.52 8586

Accuracy: 0.5956207780107151

Precision: 0.8767633984836636

Recall: 0.5956207780107151

F1: 0.6675018565219158

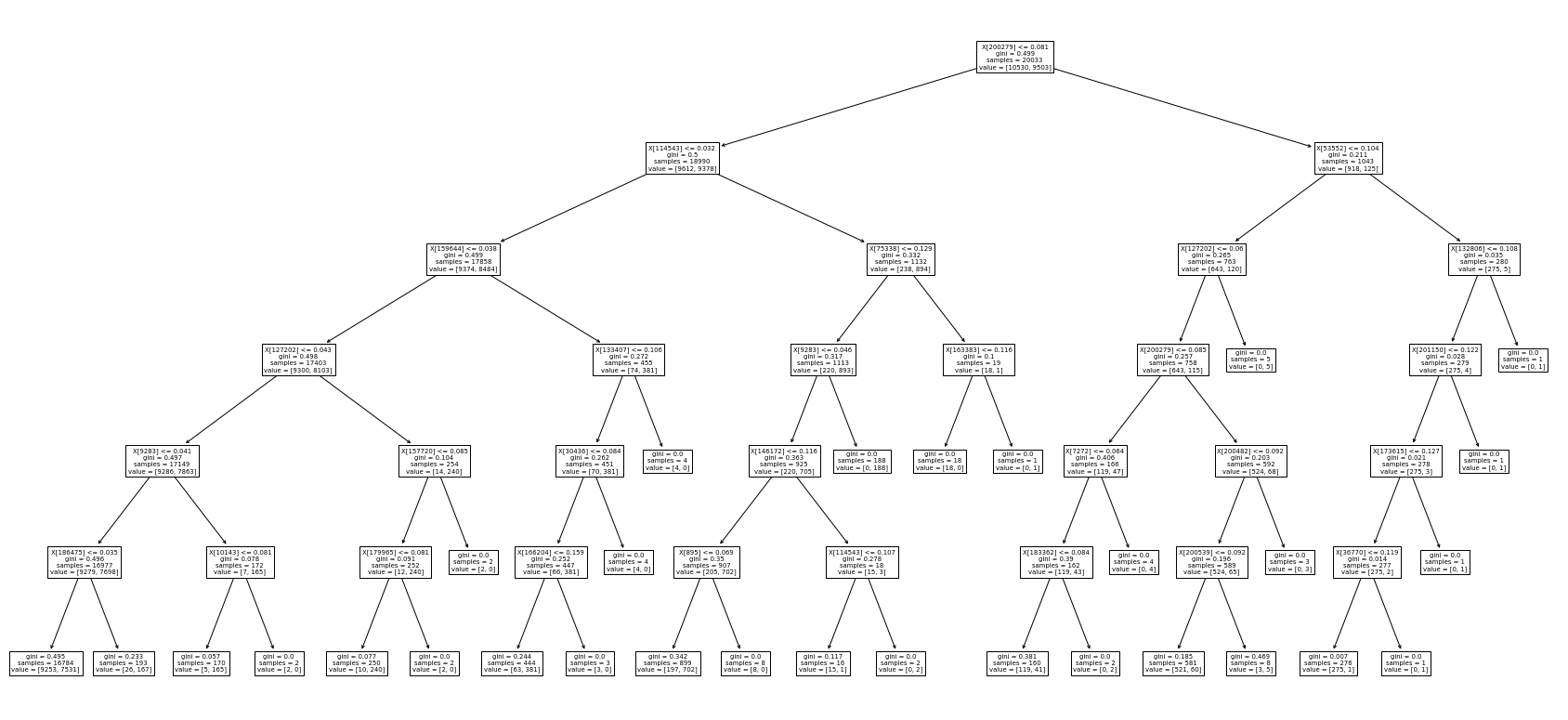
In [47]:

from sklearn import tree

plt.figure(figsize=(30, 14))

tree.plot\_tree(clf\_dt)

plt.show()



In [52]:

clf\_lr=LogisticRegression(class\_weight='balanced')

clf\_df=DecisionTreeClassifier(class\_weight='balanced')

clf\_rf=RandomForestClassifier(class\_weight='balanced')

clf\_svc=SVC(class\_weight='balanced')

voting\_clf=VotingClassifier(estimators=[('SVC', clf\_svc), ('DecisionTree', clf\_dt), ('LogReg', clf\_lr), ('RandomForest', clf\_rf)], voting='hard')

voting\_clf.fit(X\_tfidf\_train, y\_train)

y\_pred\_ensemble=voting\_clf.predict(X\_tfidf\_test)

In [53]:

print\_metrices(y\_pred\_ensemble, y\_test)

[[4099 356]

[2118 2013]]

precision recall f1-score support

0 0.66 0.92 0.77 4455

1 0.85 0.49 0.62 4131

accuracy 0.71 8586

macro avg 0.75 0.70 0.69 8586

weighted avg 0.75 0.71 0.70 8586

Accuracy: 0.711856510598649

Precision: 0.8006744802403947

Recall: 0.711856510598649

F1: 0.7271240780056136

In [54]:

acc\_table={

'Logistic Regression' : accuracy\_score(y\_pred\_lr, y\_test),

'LinearSVC' : accuracy\_score(y\_pred\_svc, y\_test),

'Decision Tree' : accuracy\_score(y\_pred\_dt, y\_test),

'Naive Bayes' : accuracy\_score(y\_pred\_nb, y\_test),

'Ensemble' : accuracy\_score(y\_pred\_ensemble, y\_test)

}

In [55]:

acc\_df=pd.DataFrame(acc\_table.items(), columns=['Model', 'Accuracy'])

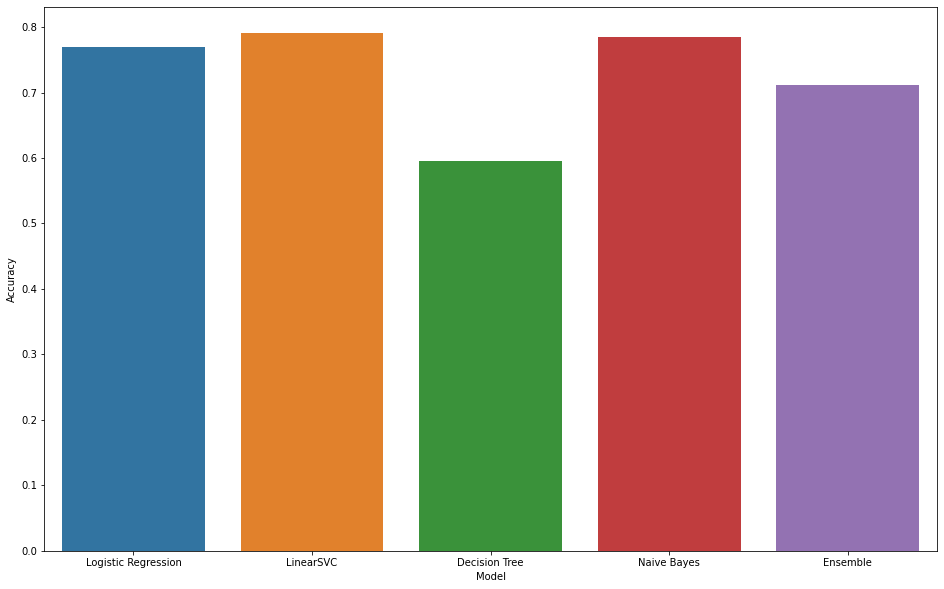
In [56]:

plt.figure(figsize=(16, 10))

sns.barplot(x=acc\_df['Model'], y=acc\_df['Accuracy'], data=acc\_df)

Out[56]:

<AxesSubplot:xlabel='Model', ylabel='Accuracy'>



In [57]:

from keras.preprocessing.text import Tokenizer

from keras.preprocessing.sequence import pad\_sequences

In [61]:

max\_words=1000

max\_len=100

tokenizer=Tokenizer(num\_words=max\_words, oov\_token='<OOV>')

tokenizer.fit\_on\_texts(X\_train)

train\_sequences=tokenizer.texts\_to\_sequences(X\_train)

train\_padded\_sequences=pad\_sequences(train\_sequences, maxlen=max\_len, padding='post')

test\_sequences=tokenizer.texts\_to\_sequences(X\_test)

test\_padded\_sequences=pad\_sequences(test\_sequences, maxlen=max\_len, padding='post')

In [62]:

print(train\_sequences[0])

print(train\_padded\_sequences[0])

[51, 1, 240, 151, 1, 1, 78]

[ 51 1 240 151 1 1 78 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0]

In [63]:

from keras.layers import LSTM, Activation, Dense, Dropout, Input, Embedding, GlobalAveragePooling1D

from keras.models import Model

from keras.models import Sequential

In [64]:

import numpy as np

training\_padded=np.array(train\_padded\_sequences)

training\_labels=np.array(y\_train)

testing\_padded=np.array(test\_padded\_sequences)

testing\_labels=np.array(y\_test)

In [65]:

vocab\_size=10000

embedding\_dim=16

In [69]:

Model=Sequential([

Embedding(vocab\_size, embedding\_dim, input\_length=max\_len),

GlobalAveragePooling1D(),

Dense(24, activation='relu'),

Dense(1, activation='sigmoid')

])

Model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

In [70]:

Model.summary()

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

embedding\_1 (Embedding) (None, 100, 16) 160000

global\_average\_pooling1d\_1 (None, 16) 0

(GlobalAveragePooling1D)

dense\_1 (Dense) (None, 24) 408

dense\_2 (Dense) (None, 1) 25

=================================================================

Total params: 160,433

Trainable params: 160,433

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

In [71]:

num\_epochs=30

history=Model.fit(training\_padded, training\_labels, epochs=num\_epochs, validation\_data=(testing\_padded, testing\_labels), verbose=2)

Epoch 1/30

627/627 - 2s - loss: 0.6855 - accuracy: 0.5494 - val\_loss: 0.6619 - val\_accuracy: 0.6551 - 2s/epoch - 3ms/step

Epoch 2/30

627/627 - 1s - loss: 0.5958 - accuracy: 0.6937 - val\_loss: 0.5578 - val\_accuracy: 0.7106 - 882ms/epoch - 1ms/step

Epoch 3/30

627/627 - 1s - loss: 0.5185 - accuracy: 0.7440 - val\_loss: 0.5210 - val\_accuracy: 0.7374 - 871ms/epoch - 1ms/step

Epoch 4/30

627/627 - 1s - loss: 0.4897 - accuracy: 0.7620 - val\_loss: 0.5200 - val\_accuracy: 0.7362 - 947ms/epoch - 2ms/step

Epoch 5/30

627/627 - 1s - loss: 0.4783 - accuracy: 0.7664 - val\_loss: 0.5107 - val\_accuracy: 0.7409 - 975ms/epoch - 2ms/step

Epoch 6/30

627/627 - 1s - loss: 0.4737 - accuracy: 0.7695 - val\_loss: 0.5159 - val\_accuracy: 0.7390 - 934ms/epoch - 1ms/step

Epoch 7/30

627/627 - 1s - loss: 0.4684 - accuracy: 0.7709 - val\_loss: 0.5101 - val\_accuracy: 0.7407 - 957ms/epoch - 2ms/step

Epoch 8/30

627/627 - 1s - loss: 0.4671 - accuracy: 0.7709 - val\_loss: 0.5138 - val\_accuracy: 0.7361 - 969ms/epoch - 2ms/step

Epoch 9/30

627/627 - 1s - loss: 0.4640 - accuracy: 0.7737 - val\_loss: 0.5173 - val\_accuracy: 0.7336 - 990ms/epoch - 2ms/step

Epoch 10/30

627/627 - 1s - loss: 0.4635 - accuracy: 0.7734 - val\_loss: 0.5115 - val\_accuracy: 0.7423 - 948ms/epoch - 2ms/step

Epoch 11/30

627/627 - 1s - loss: 0.4620 - accuracy: 0.7753 - val\_loss: 0.5121 - val\_accuracy: 0.7399 - 998ms/epoch - 2ms/step

Epoch 12/30

627/627 - 1s - loss: 0.4627 - accuracy: 0.7754 - val\_loss: 0.5192 - val\_accuracy: 0.7343 - 919ms/epoch - 1ms/step

Epoch 13/30

627/627 - 1s - loss: 0.4614 - accuracy: 0.7752 - val\_loss: 0.5389 - val\_accuracy: 0.7215 - 868ms/epoch - 1ms/step

Epoch 14/30

627/627 - 1s - loss: 0.4614 - accuracy: 0.7715 - val\_loss: 0.5176 - val\_accuracy: 0.7364 - 865ms/epoch - 1ms/step

Epoch 15/30

627/627 - 1s - loss: 0.4603 - accuracy: 0.7753 - val\_loss: 0.5303 - val\_accuracy: 0.7272 - 864ms/epoch - 1ms/step

Epoch 16/30

627/627 - 1s - loss: 0.4617 - accuracy: 0.7752 - val\_loss: 0.5151 - val\_accuracy: 0.7363 - 871ms/epoch - 1ms/step

Epoch 17/30

627/627 - 1s - loss: 0.4611 - accuracy: 0.7741 - val\_loss: 0.5159 - val\_accuracy: 0.7364 - 944ms/epoch - 2ms/step

Epoch 18/30

627/627 - 1s - loss: 0.4599 - accuracy: 0.7768 - val\_loss: 0.5194 - val\_accuracy: 0.7425 - 954ms/epoch - 2ms/step

Epoch 19/30

627/627 - 1s - loss: 0.4607 - accuracy: 0.7753 - val\_loss: 0.5188 - val\_accuracy: 0.7423 - 878ms/epoch - 1ms/step

Epoch 20/30

627/627 - 1s - loss: 0.4594 - accuracy: 0.7769 - val\_loss: 0.5159 - val\_accuracy: 0.7372 - 876ms/epoch - 1ms/step

Epoch 21/30

627/627 - 1s - loss: 0.4601 - accuracy: 0.7756 - val\_loss: 0.5178 - val\_accuracy: 0.7432 - 987ms/epoch - 2ms/step

Epoch 22/30

627/627 - 1s - loss: 0.4593 - accuracy: 0.7753 - val\_loss: 0.5277 - val\_accuracy: 0.7304 - 1s/epoch - 2ms/step

Epoch 23/30

627/627 - 1s - loss: 0.4592 - accuracy: 0.7758 - val\_loss: 0.5257 - val\_accuracy: 0.7419 - 869ms/epoch - 1ms/step

Epoch 24/30

627/627 - 1s - loss: 0.4594 - accuracy: 0.7756 - val\_loss: 0.5156 - val\_accuracy: 0.7407 - 872ms/epoch - 1ms/step

Epoch 25/30

627/627 - 1s - loss: 0.4603 - accuracy: 0.7740 - val\_loss: 0.5181 - val\_accuracy: 0.7420 - 868ms/epoch - 1ms/step

Epoch 26/30

627/627 - 1s - loss: 0.4589 - accuracy: 0.7750 - val\_loss: 0.5287 - val\_accuracy: 0.7428 - 874ms/epoch - 1ms/step

Epoch 27/30

627/627 - 1s - loss: 0.4593 - accuracy: 0.7753 - val\_loss: 0.5291 - val\_accuracy: 0.7290 - 965ms/epoch - 2ms/step

Epoch 28/30

627/627 - 1s - loss: 0.4594 - accuracy: 0.7724 - val\_loss: 0.5331 - val\_accuracy: 0.7398 - 864ms/epoch - 1ms/step

Epoch 29/30

627/627 - 1s - loss: 0.4595 - accuracy: 0.7750 - val\_loss: 0.5174 - val\_accuracy: 0.7435 - 858ms/epoch - 1ms/step

Epoch 30/30

627/627 - 1s - loss: 0.4581 - accuracy: 0.7767 - val\_loss: 0.5161 - val\_accuracy: 0.7412 - 864ms/epoch - 1ms/step

In [73]:

def plot\_graphs(history, string):

plt.plot(history.history[string])

plt.plot(history.history['val\_'+string])

plt.xlabel('Epochs')

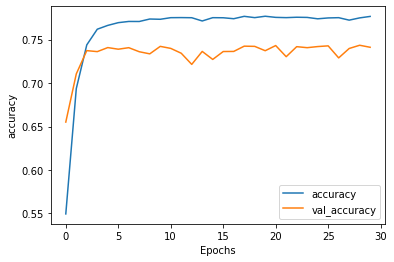
plt.ylabel(string)

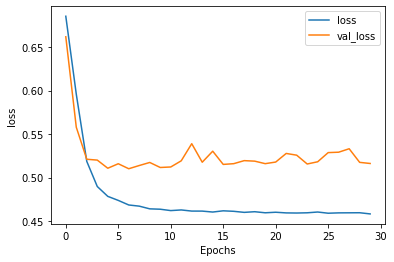
plt.legend([string, 'val\_'+string])

plt.show()

plot\_graphs(history, 'accuracy')

plot\_graphs(history, 'loss')





In [ ]: