In [37]:
```
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import nltk
import numpy as np
import re
import string
import seaborn as sns
```

Veriyi okuma

In [38]:
```
data = pd.read_csv("C:\\Users\\yepes\\OneDrive\\Bureau\\prot\\toxic_comment_classificatior
data
```

Out[38]:

	comment	toxicity
0	He got his money... now he lies in wait till a...	0.373134
1	Mad dog will surely put the liberals in mental...	0.605263
2	And Trump continues his lifelong cowardice by ...	0.666667
3	"while arresting a man for resisting arrest".\...	0.815789
4	A bus or subway is a public service, is it not...	0.000000
...	...	...
83817	Not really. I changed my registration to Democ...	0.000000
83818	Chris\n\nI checked the website for 'YOW'.\n\nT...	0.000000
83819	It's good to know that this SCOTUS will toss o...	0.000000
83820	Olie: don't sweat it, they either threw too mu...	0.000000
83821	I read what you wrote. You wrote a half-truth ...	0.000000

83822 rows × 2 columns

In [39]:
```
# Veriden bilgi alma
```

In [40]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 83822 entries, 0 to 83821
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   comment   83822 non-null  object 
 1   toxicity  83822 non-null  float64
dtypes: float64(1), object(1)
memory usage: 982.4+ KB
```

In [41]:
```
#veride farkli siniflar
```

In [42]:
```
data.toxicity.value_counts(normalize=True)
```

Out[42]:
```
0.000000    0.781931
0.800000    0.015891
1.000000    0.009699
0.833333    0.008494
0.142857    0.008041
              ...
0.408163    0.000012
0.926520    0.000012
0.506173    0.000012
0.000733    0.000012
0.020619    0.000012
Name: toxicity, Length: 1619, dtype: float64
```

In [43]:
```
missing_value = ["NaN"]
data
data.isnull()
print("\nData frame showing NaN values: \n\n ", data.isnull())
print("nShow NaN column : \n\n", data.isnull().sum())
print("\nCount total NaN at each column in a df : ", data.isnull().sum().sum())
```

```
Data frame showing NaN values:

         comment  toxicity
0        False     False
1        False     False
2        False     False
3        False     False
4        False     False
...        ...       ...
83817    False     False
83818    False     False
83819    False     False
83820    False     False
83821    False     False

[83822 rows x 2 columns]
nShow NaN column :

 comment     0
toxicity    0
dtype: int64

Count total NaN at each column in a df :  0
```

In [44]:
```
data[data.isnull().any(axis=1)]
```

Out[44]:

	comment	toxicity

In [45]:
```
data.dropna(subset= ["comment"], inplace=True)  #df.dropna(subset = ["column2"], inplace=1
```

In [46]:
```
missing_value = ["NaN"]
data
data.isnull()
print("\nData frame showing NaN values: \n\n ", data.isnull())
print("nShow NaN column : \n\n", data.isnull().sum())
print("\nCount total NaN at each column in a df : ", data.isnull().sum().sum())
```

```
Data frame showing NaN values:

         comment  toxicity
0        False     False
```

```
1         False     False
2         False     False
3         False     False
4         False     False
...         ...       ...
83817     False     False
83818     False     False
83819     False     False
83820     False     False
83821     False     False

[83822 rows x 2 columns]
nShow NaN column :

 comment     0
toxicity    0
dtype: int64

Count total NaN at each column in a df :  0
```

Preprocess

In [47]:
```
# Text preprocessing steps - remove numbers, capital letters, punctuation, '\n'

# remove all numbers with letters attached to them
alphanumeric = lambda x: re.sub('\w*\d\w*', ' ', x)

# '[%s]' % re.escape(string.punctuation),' ' - replace punctuation with white space
# .lower() - convert all strings to lowercase
punc_lower = lambda x: re.sub('[%s]' % re.escape(string.punctuation), ' ', x.lower())

# Remove all '\n' in the string and replace it with a space
remove_n = lambda x: re.sub("\n", " ", x)

# Remove all non-ascii characters
remove_non_ascii = lambda x: re.sub(r'[^\x00-\x7f]',r' ', x)

# Apply all the lambda functions wrote previously through .map on the comments column
data['comment'] = data['comment'].map(alphanumeric).map(punc_lower).map(remove_n).map(remo

data['comment'][0]
```

Out[47]:
```
'he got his money    now he lies in wait till after the election in   yrs     dirty politi
cians need to be afraid of tar and feathers again    but they aren t and so the people get
screwed '
```

Creating WordCloud Useful to show the words which occur most frequently for each category. Warning: Profanity ahead.

In [48]:
```
#pip install wordcloud
```

In [49]:
```
#conda env create -f toxic_comment_classification_dataset.csv
```

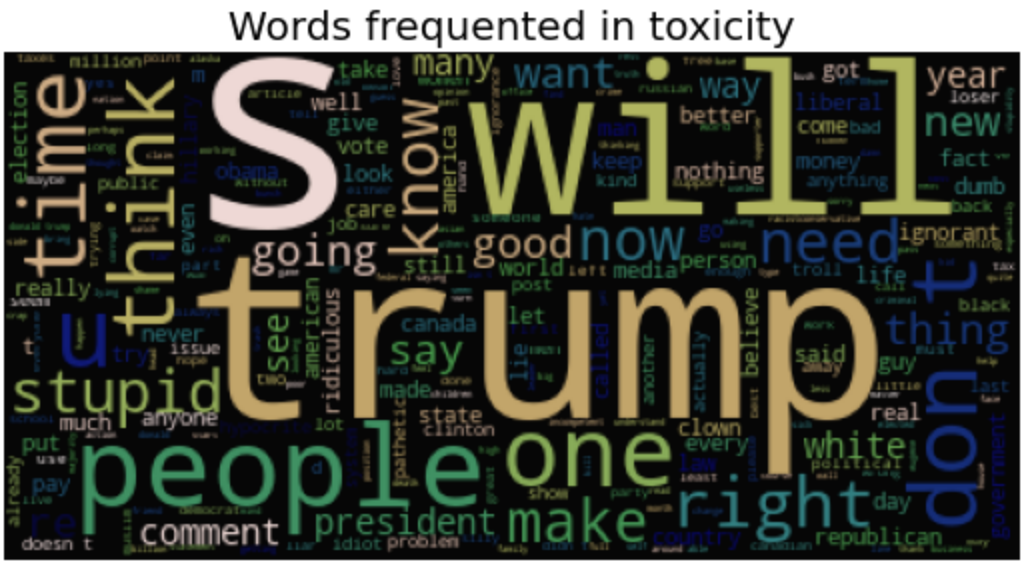
In [50]:
```
import wordcloud
from PIL import Image
```

```
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
from nltk.corpus import stopwords
```

In [51]:
```
def wordcloud(data, label):

    # Print only rows where the toxic category label value is 1 (ie. the comment is toxic,
    subset=data[data[label]==1]
    text=subset.comment.values
    wc= WordCloud(background_color="black",max_words=4000)

    wc.generate(" ".join(text))

    plt.figure(figsize=(20,20))
    plt.subplot(221)
    plt.axis("off")
    plt.title("Words frequented in {}".format(label), fontsize=20)
    plt.imshow(wc.recolor(colormap= 'gist_earth' , random_state=244), alpha=0.98)
```

In [52]:
```
#data_tox = data.loc[:,['comment','toxicity']]
```

In [53]:
```
wordcloud(data,'toxicity')
```



Import relevant packages for modelling

In [54]:
```
# Import packages for pre-processing
from sklearn import preprocessing
from sklearn.feature_selection import SelectFromModel

# Import tools to split data and evaluate model performance
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.metrics import f1_score, precision_score, recall_score, precision_recall_curv
from sklearn.metrics import roc_auc_score, roc_curve

# Import ML algos
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import MultinomialNB, BernoulliNB
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_extraction.text import TfidfVectorizer,CountVectorizer
```

In [55]:
```
# Data Transformation
```

In [56]:
```
def replace_value(val):
    if val>0:
        return 1;
    else:
        return 0
```

Create simple function that takes in a dataset and allows user to choose dataset, toxicity label, vectorizer and number of ngrams

In [57]:
```
 data["toxicity"]=data["toxicity"].apply(replace_value, 1)
```

In [58]:
```
 # Split our data into training and test data
x = data.comment
y = data["toxicity"]
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=2)


cv1 = TfidfVectorizer(stop_words='english')
x_train_cv1 = cv1.fit_transform(x_train) # Learn the vocabulary dictionary and return ter
x_test_cv1  = cv1.transform(x_test)

#-----------------Vocabulary is ok---------------

#-----------------Now Let's Teach ours Models on Vocabulary Trained data----------------


# ----------------Models definition---------------
LR_model     = LogisticRegression()
Rdmf_model = RandomForestClassifier(n_estimators=3, random_state=3)
#knn_model          = KNeighborsClassifier()
BerN_model          = BernoulliNB()
MultN_model          = MultinomialNB()
svm_model    = LinearSVC()

#------------Model Fitting OR Learning--------------

LR_model.fit(x_train_cv1, y_train)

Rdmf_model.fit(x_train_cv1, y_train)

#knn_model.fit(x_train_cv1, y_train)

BerN_model.fit(x_train_cv1, y_train)

MultN_model.fit(x_train_cv1, y_train)

svm_model.fit(x_train_cv1, y_train)


#---------------Model Testing On Test Data---- Getting Our Model Learning Percentage with

LR_model_score=LR_model.score(x_test_cv1,y_test)
```

```
Rdmf_model_score=Rdmf_model.score(x_test_cv1,y_test)

#knn_model_score=knn_model.score(x_test_cv1,y_test)

BerN_model_score=BerN_model.score(x_test_cv1,y_test)

MultN_model_score=MultN_model.score(x_test_cv1,y_test)

svm_model_score=svm_model.score(x_test_cv1,y_test)



print("LR_model    :  ",LR_model_score)
print("Rdmf_model  :  ",Rdmf_model_score)
#print("knn_model   :  ",knn_model_score)
print("BerN_model  :  ",BerN_model_score)
print("MultN_model: ",MultN_model_score)
print("svm_model   : ",svm_model_score)
```

```
LR_model    :   0.9157752415795125
Rdmf_model  :   0.8973635026046844
BerN_model  :   0.8948582335865113
MultN_model:  0.848649938362429
svm_model   :  0.9350618364019565
```

In [59]:

```
def cv_tf_train_test_function(df_done,label,vectorizer,ngram):

    ''' Train/Test split'''
    # Split the data into X and y data sets
    X = df_done.comment
    y = df_done[label]

   # y= mydata['DEATH_EVENT']
#x = mydata.drop('DEATH_EVENT', axis=1)

    # Split our data into training and test data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=

    ''' Count Vectorizer/TF-IDF '''

    # Create a Vectorizer object and remove stopwords from the table
    cv1 = vectorizer(ngram_range=(ngram), stop_words='english')

    X_train_cv1 = cv1.fit_transform(X_train) # Learn the vocabulary dictionary and return
    X_test_cv1  = cv1.transform(X_test)      # Learn a vocabulary dictionary of all tokens

    # Output a Dataframe of the CountVectorizer with unique words as the labels
    # test = pd.DataFrame(X_train_cv1.toarray(), columns=cv1.get_feature_names())

    ''' Initialize all model objects and fit the models on the training data '''
    lr = LogisticRegression()
    lr.fit(X_train_cv1, y_train)
    print('lr done')

   # knn = KNeighborsClassifier(n_neighbors=5)
   # knn.fit(X_train_cv1, y_train)

    bnb = BernoulliNB()
    bnb.fit(X_train_cv1, y_train)
    print('bnb done')

    mnb = MultinomialNB()
    mnb.fit(X_train_cv1, y_train)
    print('mnb done')
```

```
    svm_model = LinearSVC()
    svm_model.fit(X_train_cv1, y_train)

    randomforest = RandomForestClassifier(n_estimators=3, random_state=3)
    randomforest.fit(X_train_cv1, y_train)
    print('rdf done')

    #testing on test data

    #randomforest.score(x_test_cv1,y_test_cv1)



    # Create a list of F1 score of all models
    f1_score_data = {'F1 Score':[f1_score(lr.predict(X_test_cv1), y_test),
                                 f1_score(bnb.predict(X_test_cv1), y_test), f1_score(mnb.pred
                                 f1_score(svm_model.predict(X_test_cv1), y_test), f1_score(ra

    # Create DataFrame with the model names as column labels
    df_f1 = pd.DataFrame(f1_score_data, index=['Log Regression', 'BernoulliNB', 'Multinomi

    return df_f1
```

In [60]:

```
#Let's create a TF-IDF vectorizer object for each category and calculate the F1 scores aci
```

In [61]:

```
import time

t0 = time.time()

df_tox_cv = cv_tf_train_test_function(data, 'toxicity', TfidfVectorizer, (1,1))
df_tox_cv.rename(columns={'F1 Score': 'F1 Score(toxicity)'}, inplace=True)

t1 = time.time()

total = 'Time taken: {} seconds'.format(t1-t0)
print(total)

df_tox_cv

# Various permutations of the dataset, category, vectorizer and n-gram

cv_tf_train_test_function(data, 'toxicity', CountVectorizer, (1,1))


cv_tf_train_test_function(data, 'toxicity', TfidfVectorizer, (1,1))
```

```
lr done
bnb done
mnb done
rdf done
Time taken: 26.385239839553833 seconds
C:\Users\yepes\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:762: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
lr done
```

```
bnb done
mnb done
C:\Users\yepes\anaconda3\lib\site-packages\sklearn\svm\_base.py:976: ConvergenceWarning: L
iblinear failed to converge, increase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
rdf done
lr done
bnb done
mnb done
rdf done
```

Out[61]:

	F1 Score
Log Regression	0.768727
BernoulliNB	0.769002
MultinomialNB	0.475034
SVM	0.837237
Random Forest	0.747283

In [62]:

```
f1_all = pd.concat([df_tox_cv], axis=1)
f1_all
f1_all_trp = f1_all.transpose()
f1_all_trp
```

Out[62]:

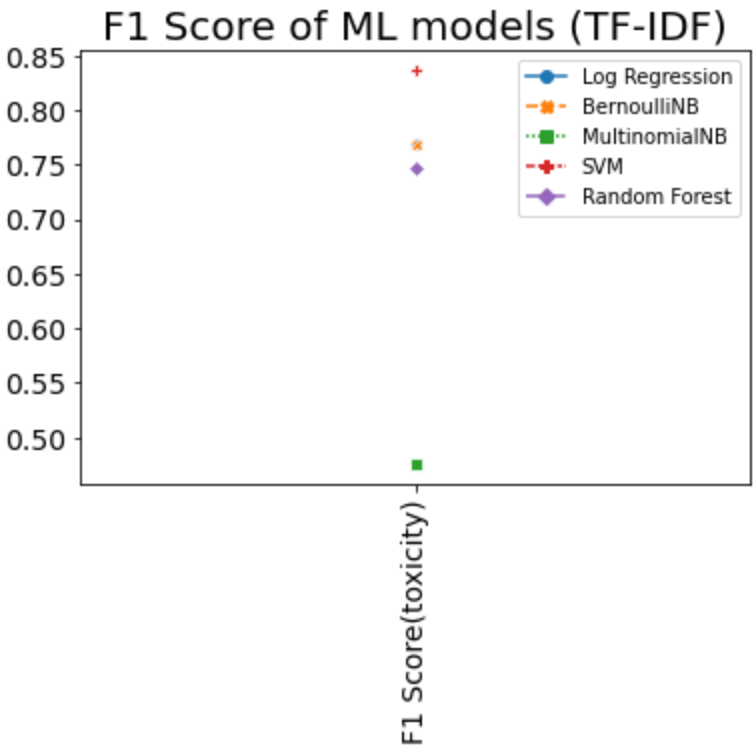
	Log Regression	BernoulliNB	MultinomialNB	SVM	Random Forest
F1 Score(toxicity)	0.768727	0.769002	0.475034	0.837237	0.747283

In [63]:

```
#plt.figure(figsize=(10,10))
sns.lineplot(data=f1_all_trp, markers=True)
plt.xticks(rotation='90', fontsize=14)
plt.yticks(fontsize=14)
#plt.legend(loc='best')
plt.title('F1 Score of ML models (TF-IDF)', fontsize=20)
```

Out[63]:

```
Text(0.5, 1.0, 'F1 Score of ML models (TF-IDF)')
```

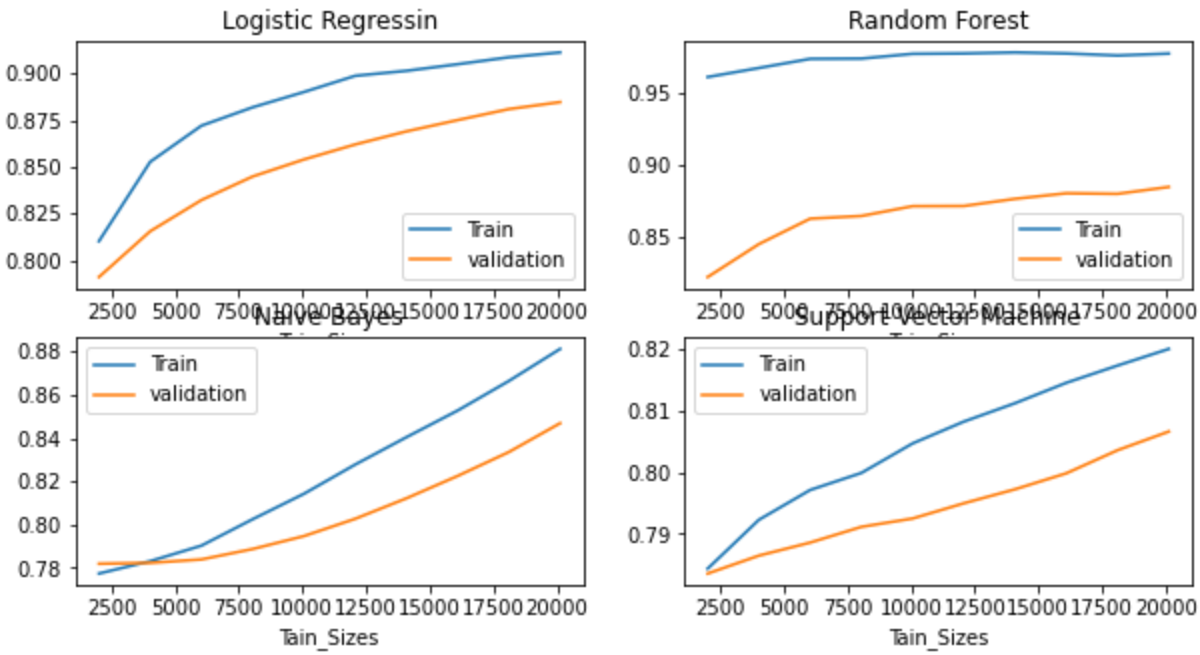


In [64]:

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import learning_curve



N, LR_train_score, LR_val_score=learning_curve(LR_model,x_test_cv1,y_test,train_sizes=np.l
N, G_train_score, G_val_score=learning_curve(BerN_model,x_test_cv1,y_test,train_sizes=np.l
N, SVC_train_score, SVC_val_score=learning_curve(MultN_model ,x_test_cv1,y_test,train_size
N, NLP_train_score, NLP_val_score=learning_curve(svm_model,x_test_cv1,y_test,train_sizes=n
N, RF_train_score, RF_val_score=learning_curve(Rdmf_model,x_test_cv1,y_test,train_sizes=np


print(N)

plt.figure(figsize=(10,5))
plt.subplot(2,2,1)
plt.plot(N,LR_train_score.mean(axis=1),label='Train')
plt.plot(N,LR_val_score.mean(axis=1),label='validation')
plt.xlabel('Tain_Sizes')
plt.title('Logistic Regressin')
plt.legend()


plt.subplot(2,2,2)
plt.plot(N,RF_train_score.mean(axis=1),label='Train')
plt.plot(N,RF_val_score.mean(axis=1),label='validation')
plt.xlabel('Tain_Sizes')
plt.title('Random Forest')
plt.legend()


plt.subplot(2,2,3)

plt.plot(N,G_train_score.mean(axis=1),label='Train')
plt.plot(N,G_val_score.mean(axis=1),label='validation')
plt.xlabel('Tain_Sizes')
plt.title('Naive Bayes')
plt.legend()
```

```
plt.subplot(2,2,4)
plt.plot(N,SVC_train_score.mean(axis=1),label='Train')
plt.plot(N,SVC_val_score.mean(axis=1),label='validation')
plt.xlabel('Tain_Sizes')
plt.title('Support Vector Machine')
plt.legend()
plt.show()
```

```
[ 2011  4023  6035  8046 10058 12070 14081 16093 18105 20117]
```



In [65]:

```
x = data.comment
y = data['toxicity']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)

# Initiate a Tfidf vectorizer
tfv = TfidfVectorizer(ngram_range=(1,1), stop_words='english')

x_train_fit = tfv.fit_transform(x_train)  # Convert the X data into a document term matri
x_test_fit = tfv.transform(x_test)  # Converts the X_test comments into Vectorized format

randomforest = RandomForestClassifier(n_estimators=3, random_state=3)

# Train our SVM model with the X training data converted into Count Vectorized format wit
randomforest.fit(x_train_fit, y_train)
randomforest.predict(x_test_fit)
```

Out[65]:

```
array([0, 1, 0, ..., 0, 0, 0], dtype=int64)
```

In [71]:

```
# Sample Prediction
comment1 =  ['You are good person']
#comment2 = ['suck']

comment1_vect = tfv.transform(comment1)
randomforest.predict_proba(comment1_vect)[:,1]
```

Out[71]:

```
array([0.])
```

In [ ]:

In [ ]:

In [ ]: