Phase 4 Project Notebook

- **Authors: Steven Yan, Ivanko Zakharchuk**
- **Instructors: Fangfang Lee, Justin Tannenbaum**

EDA

Importing Packages

In [20]:

```
import pandas as pd
import numpy as np
import re
import pickle
import string
import matplotlib.pyplot as plt

import plotly
from plotly import graph_objs
plotly.offline.init_notebook_mode()
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = 'all'
from plotly.offline import iplot

from sklearn.model_selection import train_test_split

import nltk
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from yellowbrick.text import FreqDistVisualizer
from wordcloud import WordCloud
from nltk.stem.porter import PorterStemmer
from textblob import TextBlob
from textblob import Word

from sklearn import linear_model
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_sco
re, f1_score, roc_auc_score, classification_report,balanced_accuracy_score, precision_rec
all_curve, plot_precision_recall_curve

from yellowbrick.style import set_palette
set_palette('yellowbrick')

from utils import *
%reload_ext autoreload
%autoreload 2
```

Cornell Dataset

We started with this dataset because it contains the tweet text already, while most datasets only contain IDs. As part of the application, we specified that we would only use IDs in the publication of the dataset. It allowed us to start working on the preprocessing steps to get through the dataset.

In [21]:

```
df = pd.read_csv("data/labeled_data.csv")
```

```
df.head()
```

Out[21]:

	Unnamed: 0	count	hate_speech	offensive_language	neither	class	tweet
0	0	3	0	0	3	2	!!! RT @mayasolovely: As a woman you shouldn't...
1	1	3	0	3	0	1	!!!!! RT @mleew17: boy dats cold...tyga dwn ba...
2	2	3	0	3	0	1	!!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...
3	3	3	0	2	1	1	!!!!!!!!! RT @C_G_Anderson: @viva_based she lo...
4	4	6	0	6	0	1	!!!!!!!!!!!!! RT @ShenikaRoberts: The shit you...

In [22]:

```
df.shape
```

Out[22]:

```
(24783, 7)
```

In [23]:

```
df = df.drop("Unnamed: 0", axis=1)
df = df.rename(columns={"hate_speech": 'hate', "offensive_language": 'offensive'})
```

Target Variable

Once we have identified our target variable, we want to visualize the distribution. The figure below indicates that overwhelmingly tweets categorized as offensive totaling over 19,000, while hate tweets comprise a mere 1430.

The major challenge of automated hate speech detection is the separation of hate speech from offensive language. The methodology behind this study was to collect tweets that contained terms from the Hatebase.org lexicon.

Hate speech, as defined by ALA, is any form of expression intending to vilify, humiliate, or incite hatred against a group or an individual on the basis of race, religion, skin color, sexual or gender identity, ethnicity, disability, or national origin.

While it is protected by the First Amendment, if it incites criminal activity or threats of violence against a person or group, then it can be criminalized.

In [24]:

```
hate = len(df[df['class'] == 0])
off = len(df[df['class'] == 1])
neu = len(df[df['class'] == 2])
dist = [
    graph_objs.Bar(
        x=["hate","offensive","neutral"],
        y=[hate, off, neu],
)]
plotly.offline.iplot({"data":dist, "layout":graph_objs.Layout(title="Class Distribution"
)})
```

Reassign Labels

In [25]:

```
df['class'] = df['class'].replace([2], 1)
df['class'] = df['class'].replace([0, 1], [1, 0])
df = df.rename(columns={"class": "target"})
```

Preprocessing Function

Here is the thought process involved with each of the specific steps we identified working with the dataset to prepare the data for the modeling process:

- **We removed callouts or usernames, which is preceded by @. They contain no useful information.**
- **We removed character references, which includes HTML character references, but also emojis, unicode characters. We decided not to convert any emojis into sentiment words.**
- **We removed the hash from the hashtags and decided to keep the hashtag text because they are often words or word-like and are used to connect similar ideas across the platform. We could analyze the hashtags in a future project.**
- **We removed the Twitter codes RT and QT for retweet and quotetweet. We decided to keep the retweet words, while others have removed all the text after RT.**
- **We removed the HTML links since a lot of users linking a website reference as part of the tweet.**
- **We then removed any punctuation. We did not convert contractions into the uncontracted words.**
- **We then lowercased all the tweets for tokenizing.**
- **We removed any numbers and number containing words for tokenization and vectorizing.**
- **We removed any extra whitespace(s) between words and any leading and trailing whitespaces.**

Additional steps before modeling includes stopword removal, tokenization, lemmatizing, stemming, and/or vectorizing.

In [26]:

```
preprocess_tweet(df, 'tweet')
```

Train-Test Split

In [27]:

```
X = df.tweet
y = df.target
X_tr, X_val, y_tr, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [28]:

```
train = pd.concat([X_tr, y_tr], axis=1).reset_index()
train = train.drop(columns=['index'], axis=1)
train.head()
```

Out[28]:

	tweet	target
0	well how else will white ppl get us to forget ...	1
1	funny thing isits not just the people doing it...	0
2	nigga messed with the wrong bitch	0
3	bitch ass nigggaaa	0
4	so that real bitch	0

In [29]:

```
val = pd.concat([X_val, y_val], axis=1).reset_index()
val = val.drop(columns=['index'], axis=1)
```

Removing Stopwords/Short Words and Tokenizing

We removed the punctuation from the stopwords since we already removed the apostrophes. We could do a more thorough analysis to capture more stopwords to add to the stopwords list.

In [30]:

```
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
stop_list = [''.join(c for c in s if c not in string.punctuation) for s in stop_words]

train.tweet = train.tweet.apply(lambda x: re.sub(r'\b\w{1,2}\b', '', str(x)))
val.tweet = val.tweet.apply(lambda x: re.sub(r'\b\w{1,2}\b', '', str(x)))

train_tokens = tokenize(train, 'tweet')
val_tokens = tokenize(val, 'tweet')
train_tokenz = no_stopwords(train_tokens)
val_tokenz = no_stopwords(val_tokens)
```

Visualizing Difference Between Hate and Offensive

In [31]:

```
zero = train[train.target == 0]
one = train[train.target == 1]

zero_tokens = tokenize(zero, 'tweet')
one_tokens = tokenize(one, 'tweet')
zero_tokenz = no_stopwords(zero_tokens)
one_tokenz = no_stopwords(one_tokens)
```

Frequency Distributions

In [32]:

```
# vec = CountVectorizer()
# docs = vec.fit_transform(zero_tokenz)
# features = vec.get_feature_names()
# visualizer = FreqDistVisualizer(features=features, orient='h', n=25, size=(1080, 720))
# visualizer.fit(docs)
# custom_viz = visualizer.ax
```

```
# custom_viz.set_xlabel('Number of Tokens', fontsize=20)
# custom_viz.set_ylabel('Token', fontsize=20)
# custom_viz.set_title("Frequency Distribution of Top 25 Tokens for Non-Hate Tweets", fon
tsize=24)
# custom_viz.figure.show()
```

In [33]:

```
# vec = CountVectorizer()
# docs = vec.fit_transform(one_tokenz)
# features = vec.get_feature_names()

# visualizer = FreqDistVisualizer(features=features, orient='h', n=25, size=(1080, 720))
# visualizer.fit(docs)
# custom_viz = visualizer.ax
# custom_viz.set_xlabel('Number of Tokens', fontsize=20)
# custom_viz.set_ylabel('Token', fontsize=20)
# custom_viz.set_title("Frequency Distribution of Top 25 Tokens for Hate Tweets", fontsiz
e=24)
# custom_viz.figure.show()
```

WordClouds for Imbalanced Dataset

The wordclouds

In [36]:

```
# text = ' '.join(zero_tokenz)

# # Initialize wordcloud object
# wc = WordCloud(background_color='white', max_words=50)

# # Generate and plot wordcloud
# plt.imshow(wc.generate(text))
# plt.axis('off')
# plt.show()
```

In [37]:

```
# text = ' '.join(one_tokenz)

# # Initialize wordcloud object
# wc = WordCloud(background_color='white', max_words=50)

# # Generate and plot wordcloud
# plt.imshow(wc.generate(text))
# plt.axis('off')
# plt.show()
```

In [17]:

```
hate_list = np.setdiff1d(one_tokenz, zero_tokenz)
hate_list
```

Out[17]:

```
array(['absolved', 'accord', 'acknowledged', 'activity', 'aflcio', 'aged',
       'agg', 'ahhhahahaha', 'ahmed', 'airlines', 'aklve', 'alaqsa',
       'alcoholics', 'alls', 'alsarabsss', 'amazement', 'americathey',
       'amigo', 'anglo', 'anon', 'antiracist', 'antisemite',
       'antizionist', 'apartheid', 'appearance', 'argentino', 'ariza',
       'arkansas', 'aryan', 'aslina', 'attorney', 'axin', 'azflooding',
       'azmonsoon', 'backpedals', 'baiters', 'baka', 'balless',
```

```
'ballless', 'banner', 'banwagoning', 'barge', 'barnyard',
'bateman', 'batshit', 'bazinga', 'bdubs', 'beamthat', 'beiber',
'believes', 'belton', 'benghazzi', 'benton', 'bernstine', 'beta',
'bias', 'bibles', 'bidens', 'bikes', 'birthdayyyy', 'bisexual',
'bitcheslook', 'blacklisted', 'blaspheme', 'blondeproblems',
'boris', 'boyraping', 'brainwash', 'brainwashed', 'bran', 'brits',
'bromance', 'broner', 'buckcity', 'buffets', 'buku', 'bulldozed',
'bundle', 'butcountry', 'butthole', 'buyfoodlittleguy',
'californias', 'cantstanducunt', 'capital', 'carve', 'catholics',
'caused', 'causung', 'ceasefirelets', 'celtic', 'cement', 'chava',
'chelsey', 'chimpout', 'chinatown', 'ching', 'chong', 'chood',
'chromeasome', 'chu', 'chuu', 'chyna', 'circulated', 'clans',
'clash', 'clashes', 'clones', 'clout', 'cob', 'codeword',
'combined', 'complains', 'comthablesmh', 'condone', 'conduct',
'confronts', 'connection', 'coulter', 'cousintoucher', 'coworkeri',
'cracks', 'creation', 'credibilityshot', 'criminally', 'crisco',
'crusader', 'cspan', 'dammn', 'dans', 'darling', 'dds', 'dealcry',
'dealt', 'deedee', 'deeds', 'deeeeaaaadd', 'deen', 'defence',
'delbert', 'democr', 'deviancy', 'devin', 'dicklicker', 'dickwad',
'dietoday', 'digital', 'dome', 'donts', 'doubles', 'doughnuts',
'downsize', 'drakes', 'drreams', 'dryer', 'dss', 'dtla', 'ducked',
'duis', 'dumby', 'ebloa', 'eda', 'enduring', 'engineering',
'enraged', 'entertains', 'escape', 'establishments', 'ethiopian',
'evaaaa', 'everycunt', 'exact', 'explanation', 'faaaaggggottttt',
'facedniggers', 'faggotsfag', 'fagjo', 'fagsplease', 'fairytale',
'farmers', 'farrakhan', 'farve', 'fathom', 'faux', 'faves',
'favorited', 'feminist', 'fergusonriot', 'fieldssuburbs',
'fightpacquiao', 'fisted', 'fitz', 'flattering', 'flinched',
'flopping', 'flowing', 'foolishness', 'forced', 'forsake', 'fredo',
'fsu', 'fuckheads', 'fuckry', 'fudg', 'fuggin', 'furrybah',
'gainz', 'ganks', 'gates', 'gayer', 'gaywad', 'gaywrites',
'gazelles', 'gee', 'genetic', 'genos', 'gerryshalloweenparty',
'gettingreal', 'gezus', 'girlboy', 'glitter', 'gobbling', 'goddam',
'goddamit', 'goldbar', 'goper', 'grier', 'grilled', 'gusta',
'gypsies', 'hahahahahahaha', 'hairstyle', 'haiti', 'halfbreeds',
'hamster', 'happenings', 'happppppy', 'harassment', 'hayseed',
'healedback', 'hebrew', 'heil', 'helpful', 'hesgay', 'hesitation',
'hesters', 'heterosexual', 'heyyyyyyyyyyy', 'highlights', 'hindis',
'hires', 'hiring', 'historically', 'hitched', 'hoesand', 'hoetru',
'hollering', 'homewreckers', 'homophobic', 'honcho', 'honeybooboo',
'honour', 'hoomie', 'horrific', 'hound', 'huff', 'hugging',
'husbandry', 'hustlin', 'hypebeasts', 'ians', 'idfk', 'immoral',
'immune', 'imperfections', 'inclined', 'increase', 'indentured',
'indicator', 'indiviuals', 'infatuation', 'infest', 'infiltration',
'influenced', 'injust', 'inspect', 'intvw', 'invites', 'inviting',
'involve', 'islamnation', 'isolated', 'itwas', 'jackies', 'jai',
'japped', 'japs', 'jennas', 'jerkin', 'jigaboos', 'jock', 'judged',
'julie', 'jumpers', 'juvie', 'kakao', 'kamikaze', 'kbye',
'kennies', 'kindergarden', 'knife', 'knob', 'knockdowns',
'knooooooow', 'knowur', 'latinkings', 'leftisthomosexual',
'leftists', 'legitimizing', 'lego', 'legshis', 'lexii', 'liberty',
'lid', 'liesaboutvinscully', 'lifestyle', 'limelight', 'listeners',
'loooooool', 'losangeles', 'lotto', 'lrg', 'lucas', 'lustboy',
'lynch', 'macs', 'madonnas', 'magazine', 'malt', 'manhood', 'mao',
'maoists', 'mariachi', 'maryland', 'mayoral', 'mccartney', 'medal',
'memphistn', 'merely', 'mernin', 'metlife', 'mexicannigger', 'mgr',
'mideast', 'midlaner', 'midwest', 'migrating', 'mikey',
'milesthompson', 'milwaukie', 'minorities', 'mirin', 'mischief',
'misty', 'moccasin', 'mohamed', 'molester', 'mongerls', 'mongrels',
'monkeys', 'monkies', 'moslems', 'mouthy', 'muzzy', 'naacp',
'nahhhhhaahahahaha', 'nations', 'nazis', 'nbombs', 'nebraska',
'neveraskablackperson', 'newyorkcity', 'nggas', 'nicely', 'niger',
'niggass', 'niggerous', 'nigglets', 'niggress', 'nikejordan',
'nochill', 'nonenglish', 'noneuropeans', 'nontraditional',
'nonwhites', 'notices', 'ntx', 'nurturing', 'nws', 'obese', 'odb',
'ofmine', 'okcupid', 'okiecops', 'okies', 'olympic', 'openwide',
'oppressing', 'oppressive', 'orchids', 'osamas', 'ove', 'ovenjew',
'overbreeding', 'overrun', 'oversensitive', 'panthers',
'parenthetical', 'paypay', 'peasant', 'peckin', 'pedestrian',
'peds', 'pennsylvanians', 'peoplehate', 'perish',
'pgachampionship', 'phelps', 'phillip', 'phillips', 'phrase',
'pickananny', 'pickers', 'picky', 'placement', 'placing', 'plant',
```

```
       'plantation', 'polynesians', 'pontiac', 'ponytails', 'porto',
       'pos', 'potheads', 'powered', 'premium', 'preparations',
       'prestigious', 'priesthood', 'printer', 'printers', 'propery',
       'proslavery', 'psychiatry', 'pundits', 'pussyed', 'pwi', 'queersi',
       'rabchenko', 'racismisaliveandwellbro', 'radical', 'ramlogan',
       'randos', 'rapists', 'rasta', 'reasonswecantbetogether',
       'receptionist', 'receptionthis', 'reconnaissance', 'recruited',
       'referred', 'refused', 'regionally', 'rejects', 'religions',
       'repping', 'reptile', 'reside', 'restau', 'retared', 'retweeettt',
       'rfn', 'rhode', 'ricans', 'roid', 'roleplayinggames', 'romeo',
       'route', 'salvadoran', 'samesex', 'sandusky', 'schitt', 'scope',
       'scully', 'segal', 'servant', 'sewer', 'sexist', 'shabbat',
       'sharpie', 'sheboons', 'ship', 'shock', 'shoving', 'sickening',
       'sidekicklike', 'sion', 'sistas', 'sixes', 'skater', 'skidmarks',
       'slightlyadjusted', 'slum', 'snipe', 'soetoroobama', 'sopa',
       'sophi', 'soxs', 'spaz', 'spicskkk', 'sprinkler', 'stacey',
       'stalkin', 'standn', 'stds', 'stephenking', 'stereotypi',
       'stoopid', 'stopsavinthesehoes', 'stu', 'stubborn', 'stuckup',
       'studies', 'styl', 'styles', 'subhuman', 'subordinate', 'summers',
       'suspicious', 'swaagg', 'swags', 'swill', 'sycksyllables',
       'tapout', 'taxing', 'teabagged', 'teabaggerswho', 'teammate',
       'teenage', 'tehgodclan', 'templars', 'terroristscongies',
       'texarkana', 'thenetherlands', 'therelike', 'thetime',
       'theyfaggots', 'thingsiwillteachmychild', 'thnk', 'timmys',
       'tittyy', 'tmt', 'toms', 'tomyfacebro', 'toosoon', 'traditions',
       'tragedy', 'trannygo', 'transformthursday', 'transmitter',
       'trashiest', 'trayvonmartin', 'trout', 'tsm', 'tunis', 'tunwhat',
       'tusks', 'tweetlikepontiacholmes', 'units', 'unselfish',
       'unwashed', 'uwi', 'vaca', 'vanessa', 'vddie', 'vegasshowgirls',
       'vhia', 'vin', 'vinitahegwood', 'waahh', 'wacthh', 'wagging',
       'wallet', 'warehouse', 'warrior', 'weapon', 'wedges', 'weirdos',
       'welldid', 'wenchs', 'westvirginia', 'wher', 'whitepowerill',
       'whitest', 'whomp', 'whooooo', 'whse', 'wifebeater', 'willed',
       'wishywashy', 'witcho', 'woohoo', 'wooooow', 'worryol', 'wrongbut',
       'wrongwitch', 'yamming', 'yaselves', 'yeawhat', 'youuuuu', 'zak',
       'zigeuner', 'zion', 'zipperheads', 'zzzzzz'], dtype='<U23')
```

The traditional epithets are not found in exclusively in the hate category, only the less traditional words often in the form of hashtags can be found exclusively as hate speech. That would make sense. in terms pf

- **sexual orientation: teabagged, girlboy, azflooding, azmonsoon, molester, cousintoucher, theyfaggots, dicklicker**
- **sex: wenchs**
- **race/ethnicity/religion: osamas, spicskkk, niggerous, nigglets. nigress, ovenjew, westvirginia, texarkana, ching, chong, maoists, mexicannigger**

One clear distinction is the difference in use of nigga versus the n word. When people say the f word against homosexuals, it is more often in the derogatory sense. The p word can be just offensive or sexist, i.e. males use the p word to denigrate guys, which can be offensive but not considered hate speech.

Stemming and Lemmatization

Since there is so much colloquial use of words amongst tweets, we did not anticipate that stemming or lemmatization to have a significant impact on the predictive value of the model.

In [18]:

```
train_stem = stemming(train_tokenz)
val_stem = stemming(val_tokenz)
```

In [19]:

```
from nltk.corpus import stopwords
lemmatization(train)
lemmatization(val)
stop_words = set(stopwords.words('english'))
stop_list = [''.join(c for c in s if c not in string.punctuation) for s in stop_words]
```

```
train.lem = train['lem'].apply(lambda x: ' '.join([item for item in x.split() if item no
t in stop_list]))
val.lem = val['lem'].apply(lambda x: ' '.join([item for item in x.split() if item not in
stop_list]))
```

Out[19]:

```
0    well how else will white ppl get forget our ho...
1    funny thing isits not just the people doing it...
2                       nigga messed with the wrong bitch
3                                   bitch as nigggaaa
4                                      that real bitch
Name: lem, dtype: object
```

Out[19]:

```
0                              got missed call from bitch
1    fucking with bad bitch you gone need some mone...
2    lol credit aint where near good but know the r...
3    wipe the cum out them faggot contact lens wild...
4    nigga cheat they bitch and dont expect pay bac...
Name: lem, dtype: object
```

Baseline Model

In [20]:

```
X_tr = train.lem
X_val = val.lem
y_tr = train.target
y_val = val.target
```

TD-IDF Vectorizer

In [21]:

```
vec = TfidfVectorizer()
tfidf_tr = vec.fit_transform(X_tr)
tfidf_val = vec.transform(X_val)
```

Multinomial Naive Bayes

In [33]:

```
nb = MultinomialNB().fit(tfidf_tr, y_tr)
y_pr_nb_tr = nb.predict(tfidf_tr)
y_pr_nb_val = nb.predict(tfidf_val)
get_metrics(tfidf_tr, y_tr, tfidf_val, y_val, y_pr_nb_tr, y_pr_nb_val, nb)

make_confusion_matrix(cf = confusion_matrix(y_tr, y_pr_nb_tr),
                      X = tfidf_val,
                      y = y_val,
                      model = nb,
                      cmap='Purples',
                      title='Confusion Matrix for Naive Bayes')
```

```
/Users/examsherpa/opt/anaconda3/envs/learn-env/lib/python3.9/site-packages/sklearn/metric
s/_classification.py:1221: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_divi
sion` parameter to control this behavior.

Training F1 Score:  0.012195121951219513
Validation F1 Score:  0.0
Training Recall Score:  0.0061403508771929825
Validation Recall Score:  0.0
Training Precision Score:  0.875
Validation Precision Score:  0.0
```

```
Validation Precision Score:  0.0
Training Average Precision Score:  0.30883247804093567
Validation Average Precision Score:  0.16283262448260288
```

Random Forest Classifier

In [34]:

```
rf = RandomForestClassifier(n_estimators=100).fit(tfidf_tr, y_tr)
y_pr_rf_tr = rf.predict(tfidf_tr)
y_pr_rf_val = rf.predict(tfidf_val)
get_metrics(tfidf_tr, y_tr, tfidf_val, y_val, y_pr_rf_tr, y_pr_rf_val, rf)

make_confusion_matrix(cf = confusion_matrix(y_val, y_pr_rf_val),
                      X = tfidf_val,
                      y = y_val,
                      model = rf,
                      cmap='Oranges',
                      title='Confusion Matrix for Random Forest')
```

```
Training F1 Score:  0.9824868651488616
Validation F1 Score:  0.19944598337950137
Training Recall Score:  0.9842105263157894
Validation Recall Score:  0.12413793103448276
Training Precision Score:  0.9807692307692307
Validation Precision Score:  0.5070422535211268
Training Average Precision Score:  0.9852601324302912
Validation Average Precision Score:  0.3022896801121844
```

Logistic Regression

In [35]:

```
log = LogisticRegression().fit(tfidf_tr, y_tr)
y_pr_log_tr = log.predict(tfidf_tr)
y_pr_log_val = log.predict(tfidf_val)
get_metrics(tfidf_tr, y_tr, tfidf_val, y_val, y_pr_log_tr, y_pr_log_val, log)

make_confusion_matrix(cf = confusion_matrix(y_val, y_pr_rf_val),
                      X = tfidf_val,
                      y = y_val,
                      model = rf,
                      cmap='PuBu',
                      title='Confusion Matrix for Logistic Regression')
```

```
Training F1 Score:  0.25584795321637427
Validation F1 Score:  0.18487394957983194
Training Recall Score:  0.15350877192982457
Validation Recall Score:  0.11379310344827587
Training Precision Score:  0.7675438596491229
Validation Precision Score:  0.4925373134328358
Training Average Precision Score:  0.5826117025159812
Validation Average Precision Score:  0.3497021490008816
```

API Calling (University of Copenhagen Dataset)

To address class imbalance, we went to a dataset where 136,052 tweets were retrieved and 3383 annotated as sexist, 1972 as racist, and 11559 as neither. Sexist tweets contained n-grams that consisted of the following words: woman, girl, bitch, feminist, sexist, and racist tweets contained n-grams that consists of the following words: islam and muslim.

The hate tweet IDs were compiled and sent to the Twitter API to retrieve the corresponding tweet text, and not

The hate tweet IDs were compiled and sent to the Twitter API to retrieve the corresponding tweet text, and not all the API calls produced a result. Ultimately, another 3000 labeled hate tweets were added to the original labeled dataset.

Since this was a European study, that would make sense contextually. Once again, it would be hard to discern between offensive and hate tweets based on those sexist terms. All of those words could be part of normal discourse.

In [58]:

```
df2 = pd.read_csv('data/hate_add.csv')
df2.columns = ['id', 'label']
df2.label.value_counts()
```

Out[58]:

```
none       11559
sexism      3378
racism      1969
Name: label, dtype: int64
```

Fixing Class Imbalance

Undersampling and Oversampling Methods

In [17]:

```
df3 = pd.read_csv('data/baseline_df.csv')
df3
```

Out[17]:

	Unnamed: 0	F1 Score	Recall	Precision	PR AUC
0	Naive Bayes Baseline	NaN	0.000000	NaN	0.164594
1	Random Forest Baseline	0.160920	0.096953	0.472973	0.300614
2	Decision Tree Baseline	0.279503	0.249307	0.318021	0.123155
3	Logistic Regression Baseline	0.177677	0.108033	0.500000	0.351195
4	Naive Bayes RUS	0.250960	0.814404	0.148335	0.321911
5	Random Forest RUS	0.344031	0.742382	0.223893	0.323411
6	Logistic Regression RUS	0.334559	0.756233	0.214792	0.329453
7	Naive Bayes CNN	0.110667	0.997230	0.058584	0.198122
8	Random Forest CNN	0.165556	0.825485	0.092004	0.226680
9	Logistic Regression CNN	0.141797	0.952909	0.076598	0.252673
10	Naive Bayes SMOTE-ENN	0.391421	0.639889	0.184800	0.282544
11	Random Forest SMOTE-ENN	0.286778	0.404432	0.379221	0.296305
12	Logistic Regression SMOTE-ENN	0.286778	0.639889	0.184800	0.311180

With additional labeled as hate speech data and API Requests we were able to get more twits and balance main dataset.

The code for requests and balansing can be found in data_collection.ipynb

Load Corpus and Look at the Data

In [62]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
import seaborn as sns; sns.set()


import re
import string

import nltk
from nltk.tokenize import RegexpTokenizer
from nltk.probability import FreqDist
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from wordcloud import WordCloud

from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score
from sklearn.metrics import roc_curve, auc, classification_report
from sklearn.metrics import precision_score, recall_score, confusion_matrix, plot_confus
ion_matrix

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import TfidfTransformer


plt.style.use('fivethirtyeight')
%config InlineBackend.figure_format = 'retina'
```

In [63]:

```
df = pd.read_csv("data/balanced_data_combined.csv")
df = df.drop(columns = 'Unnamed: 0')
```

In [64]:

```
print(df.shape)
df.head()
```

(8337, 2)

Out[64]:

	text	class
0	Drasko they didn't cook half a bird you idiot ...	1
1	Hopefully someone cooks Drasko in the next ep ...	1
2	of course you were born in serbia...you're as ...	1
3	These girls are the equivalent of the irritati...	1
4	RT @YesYoureRacist: At least you're only a tin...	1

In [65]:

```
# class 0 - not hate speech
# class 1 - hate speech
```

In [66]:

```
# Crreate class description for each row in data
df['class_descr'] = df['class'].map(lambda x: 'hate_speech' if x==1 else 'not_hate_speec
h')
```

In [67]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8337 entries, 0 to 8336
Data columns (total 3 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   text         8335 non-null   object
 1   class        8337 non-null   int64 
 2   class_descr  8337 non-null   object
dtypes: int64(1), object(2)
memory usage: 195.5+ KB
```

In [68]:

```
# Drop NaN values in text column

df.dropna(subset=['text'], inplace=True)
```

Visualize Amount of tweets classes

In [70]:

```
#print(df['class'].value_counts(normalize=True))

# Class Imbalance

fig, ax = plt.subplots(figsize=(10,6))
ax = sns.countplot(df['class'], palette='Set2')

ax.set_title('Amount of Tweets Per Label',fontsize = 20)
ax.set_xlabel('Type of Tweet',fontsize = 15)
ax.set_ylabel('Count',fontsize = 15)
ax.set_xticklabels(['Not Hate Speech','Hate Speech'],fontsize = 13)

total = float(len(df)) # one person per row
for p in ax.patches:
    height = p.get_height()
    ax.text(p.get_x()+p.get_width()/2.,
            height + 3,
            '{:1.2f}'.format(height/total * 100) + '%',
            ha="center")
```

Out[70]:

```
Text(0.5,1,'Amount of Tweets Per Label')
```

Out[70]:

```
Text(0.5,0,'Type of Tweet')
```

Out[70]:

```
Text(0,0.5,'Count')
```

Out[70]:

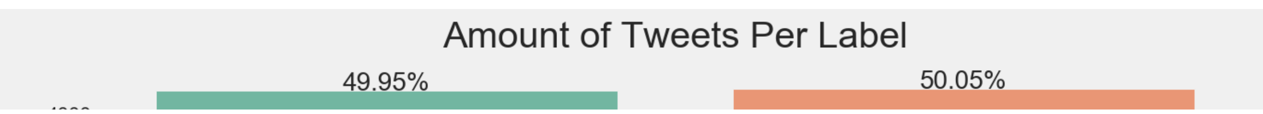
```
[Text(0,0,'Not Hate Speech'), Text(0,0,'Hate Speech')]
```

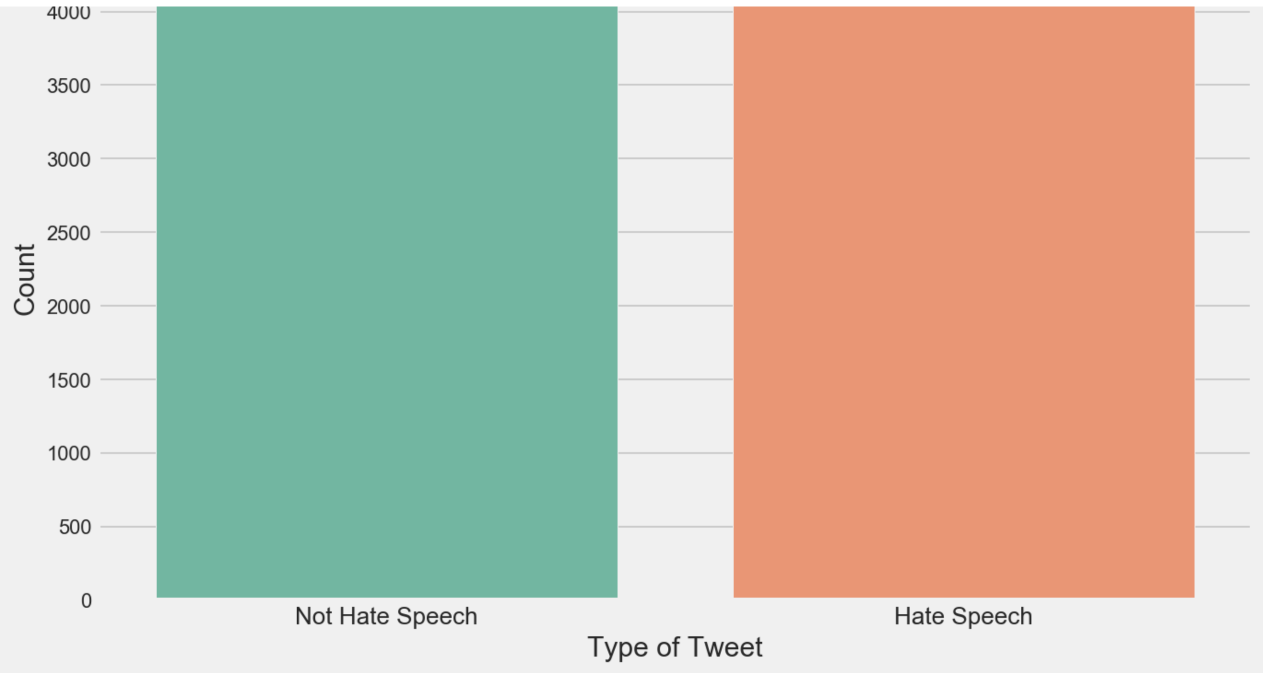
Out[70]:

```
Text(0,4166,'49.95%')
```

Out[70]:

```
Text(1,4175,'50.05%')
```





Create Document - Term Matrix

- **Preprotcessing and cleaning**
- **Tokenize**
- **Stemming and Lemming**
- **Document-Term Matrix**

Data Preprotcessing and Cleaning

In [71]:

```
# removing excess
# removing punctuation
# lovercase letters
# remove numbers or numerical values
# remove non-sesial text (/n)
```

In [72]:

```
# Create function with text cleaning techniques usinx regex

def clean_text_step1(text):
    '''
    Looking for speciffic patterns in the text and
    removing them or replacing with space
    Function returns string
    '''

    # make text lowercase
    text = text.lower()

    # string punctuations
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)

    # removing patterns and replace it with nothing
    text = re.sub('\[.*?\]', '', text)

    # removing digits if they surounded with text or digit
    text = re.sub('\w*\d\w*', '', text)

    # make just 1 space if there is more then 1
    text = re.sub('\s+', ' ', text)

    # replace new line symbol with space
```

```
    text = re.sub('\n', ' ', text)

    # removing any quotes
    text = re.sub('\"+', '', text)

    # removing &amp;
    text = re.sub('(\&amp\;)', '', text)

    # cleaning from user name
    text = re.sub('(@[^\s]+)', '', text)

    # looking for # and replacing it
    text = re.sub('(#[^\s]+)', '', text)

    # removing `rt`
    text = re.sub('(rt)', '', text)

    # looking for `httptco`
    text = re.sub('(httptco)', '', text)

    # looking for `mkr`
    text = re.sub('(mkr)', '', text)

    text = re.sub('(sexist)', '', text)

    text = re.sub('(like)', '', text)

    text = re.sub('(women)', '', text)

    return text
```

In [73]:

```
# applying function for cleaning text data

df['text'] = df['text'].apply(clean_text_step1)
```

In [74]:

```
df.head()
```

Out[74]:

	text	class	class_descr
0	drasko they didnt cook half a bird you idiot	1	hate_speech
1	hopefully someone cooks drasko in the next ep of	1	hate_speech
2	of course you were born in serbiayoure as fuck...	1	hate_speech
3	these girls are the equivalent of the irritati...	1	hate_speech
4	yesyoureracist at least youre only a tiny bit...	1	hate_speech

In [75]:

```
# Function to filter data with words that contain more then 2 characters
def txt_filtering(row, n =2):
    new_words = []
    for w in row['text'].split(' '):
        if len(w) > 2:
            new_words.append(w)
    row['text'] = ' '.join(new_words)
    return row
```

In [76]:

```
df = df.apply(txt_filtering, axis = 1)
```

Tokenization Data (splitting into smaller pieces) and removing

stopwords

In [77]:

```
stopwords_list = stopwords.words('english')
```

In [78]:

```
def tokenize_text(text):

    """
    Tocanize document and create visualization of most recent words
    Wiil filter data with stopwords
    """
    tokens = nltk.word_tokenize(text)

    stopwords_removed = [token for token in tokens if token not in stopwords_list]

    return stopwords_removed
```

In [79]:

```
processed_data = list(map(tokenize_text, df['text']))
```

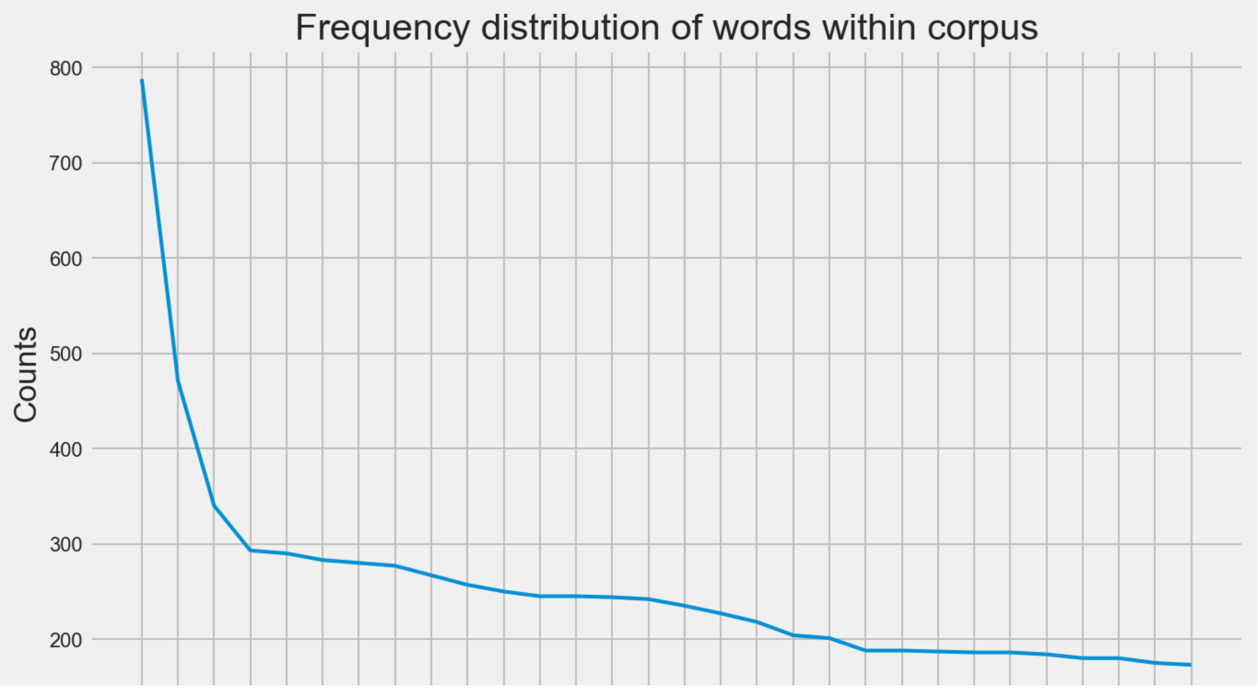
Lets Plot frequency distribution of tokens in corpus

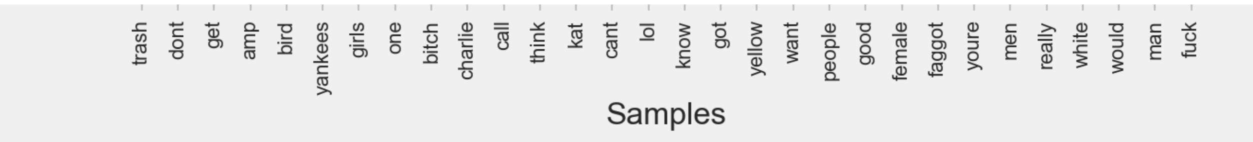
In [80]:

```
def plot_frequency(data):
    """
    Ploting words frequency distribution
    from corpus. data should be list of lists with strings
    """
    words_lst = []
    for tweet in data:
        for word in tweet:
            words_lst.append(word)

    fdist = FreqDist(words_lst)
    plt.figure(figsize=(10,6))
    fdist.plot(30, title = "Frequency distribution of words within corpus")
    plt.show()
```

In [81]:

```
plot_frequency(processed_data)
```





In [82]:

```
processed_data[:2]
```

Out[82]:

```
[['drasko', 'didnt', 'cook', 'half', 'bird', 'idiot'],
 ['hopefully', 'someone', 'cooks', 'drasko', 'next']]
```

In [83]:

```
total_vocab = set()
for tweet in processed_data:
    total_vocab.update(tweet)
len(total_vocab)
```

Out[83]:

```
17984
```

Create Docunent-Term Matrix

In [84]:

```
# look at the corpus
df.head()
```

Out[84]:

	text	class	class_descr
0	drasko they didnt cook half bird you idiot	1	hate_speech
1	hopefully someone cooks drasko the next	1	hate_speech
2	course you were born serbiayoure fucked serbia...	1	hate_speech
3	these girls are the equivalent the irritating ...	1	hate_speech
4	yesyoureracist least youre only tiny bit racis...	1	hate_speech

In [85]:

```
# Using CountVextorizer from sklearn
# in data_dtm each row represents different document
# and each collumn represents word from vocab

cv = CountVectorizer(stop_words = 'english')
df_cv = cv.fit_transform(df.text)
data_dtm = pd.DataFrame(df_cv.toarray(), columns= cv.get_feature_names())
data_dtm.index = df.index
data_dtm.head()
```

Out[85]:

	aaaaaaaaand	aaand	aamaadmipay	aamattyhealy	aamessinger	aan	aandapples	aap	aarcayne	aaron	...	zstonecipher
0	0	0	0	0	0	0	0	0	0	0	...	0
1	0	0	0	0	0	0	0	0	0	0	...	0
2	0	0	0	0	0	0	0	0	0	0	...	0
3	0	0	0	0	0	0	0	0	0	0	...	0
4	0	0	0	0	0	0	0	0	0	0	...	0

5 rows × 17778 columns

Lematizing Data

In [86]:

```
# function to creat a list with all lemmatized words

def lematizing_text(data):

    """
    Lematizing words from the corpus data
    Returns list of strings with lematized
    words in each string
    """

    lemmatizer = WordNetLemmatizer()
    lemmatized_output = []

    for tweet in data:
        lemmed = ' '.join([lemmatizer.lemmatize(w) for w in tweet])
        lemmatized_output.append(lemmed)

    return lemmatized_output
```

In [87]:

```
lemmatized_data = lematizing_text(processed_data)
```

In [88]:

```
lemmatized_data[:5]
```

Out[88]:

```
['drasko didnt cook half bird idiot',
 'hopefully someone cook drasko next',
 'course born serbiayoure fucked serbian film',
 'girl equivalent irritating asian girl couple year ago well done',
 'yesyoureracist least youre tiny bit racist racist dick']
```

Most Frequent Words for Each Class

In [89]:

```
df_freq_hate = df[df['class']==1]
df_freq_not_hate = df[df['class']==0]
```

In [90]:

```
data_hate = df_freq_hate['text']
data_not_hate = df_freq_not_hate['text']
```

In [91]:

```
def freq_wrds_class(data, n = 20, show= True):

    """
    Returns list of 2 tuples that represents frequency
    of words in document

    data - Series with string data
    n - number of most common words to show
    """

    protc_data = list(map(tokenize_text, data))
```

```
    total_vocab = set()
    for comment in protc_data:
        total_vocab.update(comment)

    if show:
        print('Total words in vocab : {}'.format(len(total_vocab)))
        print (30*'-')
        print('Top {} most frequent words:'.format(n))
        flat_data = [item for sublist in protc_data for item in sublist]
        freq = FreqDist(flat_data)
        return freq.most_common(n)
    flat_data = [item for sublist in protc_data for item in sublist]
    freq = FreqDist(flat_data)

    return freq
```

In [92]:

```
# Top 20 hate words:
freq_wrds_class(data_hate, show=True)
```

```
Total words in vocab : 9703
------------------------------
Top 20 most frequent words:
```

Out[92]:

```
[('dont', 302),
 ('bitch', 257),
 ('girls', 245),
 ('kat', 244),
 ('call', 210),
 ('get', 196),
 ('faggot', 187),
 ('think', 178),
 ('female', 176),
 ('fuck', 173),
 ('cant', 170),
 ('men', 165),
 ('ass', 152),
 ('one', 147),
 ('know', 146),
 ('nigga', 139),
 ('woman', 137),
 ('white', 133),
 ('fucking', 132),
 ('hate', 128)]
```

In [93]:

```
# Top 20 non-hate words:
freq_wrds_class(data_not_hate)
```

```
Total words in vocab : 11672
------------------------------
Top 20 most frequent words:
```

Out[93]:

```
[('trash', 672),
 ('bird', 287),
 ('yankees', 281),
 ('charlie', 257),
 ('yellow', 213),
 ('dont', 169),
 ('birds', 167),
 ('amp', 166),
 ('get', 144),
 ('lol', 140),
 ('got', 131),
 ('one', 130),
 ('monkey', 111),
```

```
 ('ghetto', 109),
 ('colored', 108),
 ('good', 94),
 ('know', 89),
 ('new', 88),
 ('love', 84),
 ('day', 84)]
```

Normalized word frequencies:

In [94]:

```
def normalized_word_fqncy(data, n=25):

    frqncy = freq_wrds_class(data, n, show = False)
    total_w_count = sum(frqncy.values())
    top = frqncy.most_common(25)
    print("Word \t\t Normalized Frequency")
    print()
    for word in top:
        normalized_frequency = word[1]/total_w_count
        print("{} \t\t {:.4}".format(word[0], normalized_frequency))
```

In [95]:

```
normalized_word_fqncy(data_hate)
```

```
Word     Normalized Frequency

dont     0.009045
bitch     0.007697
girls     0.007338
kat     0.007308
call     0.006289
get     0.00587
faggot     0.005601
think     0.005331
female     0.005271
fuck     0.005181
cant     0.005091
men     0.004942
ass     0.004552
one     0.004403
know     0.004373
nigga     0.004163
woman     0.004103
white     0.003983
fucking     0.003953
hate     0.003834
amp     0.003804
youre     0.003684
want     0.003624
people     0.003594
trash     0.003474
```

In [96]:

```
normalized_word_fqncy(data_not_hate)
```

```
Word     Normalized Frequency

trash     0.01933
bird     0.008254
yankees     0.008081
charlie     0.007391
yellow     0.006126
dont     0.00486
birds     0.004803
amp     0.004774
get     0.004141
```

```
lol     0.004026
got     0.003768
one     0.003739
monkey     0.003192
ghetto     0.003135
colored     0.003106
good     0.002703
know     0.00256
new     0.002531
love     0.002416
day     0.002416
game     0.002387
want     0.002387
make     0.002358
would     0.00233
people     0.00233
```

Visualization

In [97]:

```
# Seperate frequency of each class

hate_freq = freq_wrds_class(data_hate, show =False)
not_hate_freq = freq_wrds_class(data_not_hate, show =False)
```

In [98]:

```
# create counts of hate and not hate with values and words

hate_bar_counts = [x[1] for x in hate_freq.most_common(25)]
hate_bar_words = [x[0] for x in hate_freq.most_common(25)]

not_hate_bar_counts = [x[1] for x in not_hate_freq.most_common(25)]
not_hate_bar_words = [x[0] for x in not_hate_freq.most_common(25)]
```

In [99]:

```
# set the color of our bar graphs
color = cm.viridis_r(np.linspace(.8,.16, 30))
```

In [101]:

```
new_figure = plt.figure(figsize=(14,6))

ax = new_figure.add_subplot(121)
ax2 = new_figure.add_subplot(122)

# Generate a line plot on first axes
ax.bar(hate_bar_words, hate_bar_counts, color=color)
ax.plot(colormap='PRGn')

# Draw a scatter plot on 2nd axes
ax2.bar(not_hate_bar_words, not_hate_bar_counts, color=color )

ax.title.set_text('Hate Words')
ax2.title.set_text('Not Hate Words')

for ax in new_figure.axes:
    plt.sca(ax)
    plt.xticks(rotation=60)

plt.tight_layout(pad=0)
new_figure.suptitle('Top 25 Most Frequent Words per Label', fontsize =18, y =1.05)

# plt.savefig('../images/word_count_graphs.png')

plt.show()
```

Out[101]:

```
<BarContainer object of 25 artists>
```

Out[101]:

```
[]
```

Out[101]:

```
<BarContainer object of 25 artists>
```

Out[101]:

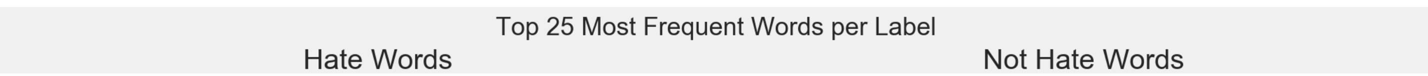
```
([0,
  1,
  2,
  3,
  4,
  5,
  6,
  7,
  8,
  9,
  10,
  11,
  12,
  13,
  14,
  15,
  16,
  17,
  18,
  19,
  20,
  21,
  22,
  23,
  24],
 <a list of 25 Text xticklabel objects>)
```

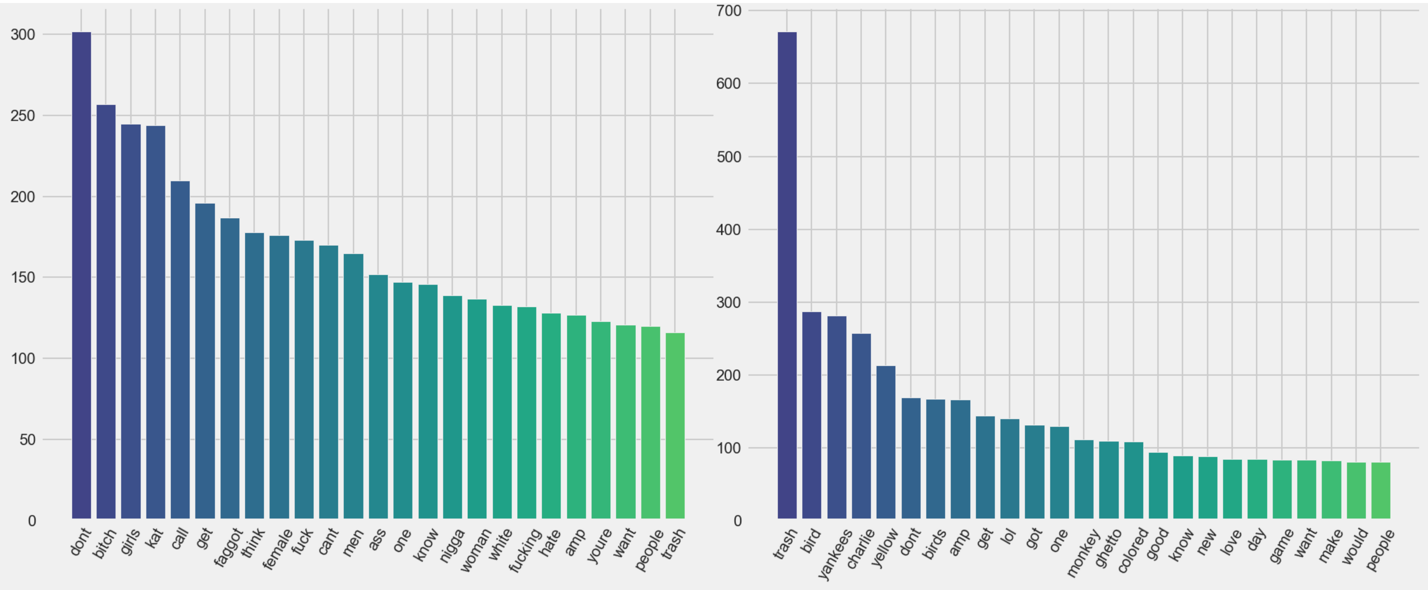
Out[101]:

```
([0,
  1,
  2,
  3,
  4,
  5,
  6,
  7,
  8,
  9,
  10,
  11,
  12,
  13,
  14,
  15,
  16,
  17,
  18,
  19,
  20,
  21,
  22,
  23,
  24],
 <a list of 25 Text xticklabel objects>)
```

Out[101]:

```
Text(0.5,1.05,'Top 25 Most Frequent Words per Label')
```





Create Word Clouds

In [102]:

```
hate_dictionary = dict(zip(hate_bar_words, hate_bar_counts))
not_hate_dictionary = dict(zip(not_hate_bar_words, not_hate_bar_counts))
```

In [103]:

```
def wordcloud(dic, save = False, name = None):


    wordcloud = WordCloud(colormap='Spectral', background_color='mintcream').generate_fr
om_frequencies(dic)

    # Display the generated image w/ matplotlib:
    plt.figure(figsize=(8,6), facecolor='k')
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis("off")
    plt.tight_layout(pad=0)
    # plt.title('Hate Speech Word Cloud', color = "w")
    if save :
        #plt.savefig('../images/{}_wordcloud.png'.format(name))
        plt.show()
    return
```

In [104]:

```
wordcloud(hate_dictionary, save = True, name = 'hate_speech')
```

In [105]:

```
wordcloud(not_hate_dictionary, save = True, name = 'not_hate_speech')
```

Checking unique words in both classes

In [106]:

```
hate_wocab = freq_wrds_class(data_hate, n = 9703)
```

```
Total words in vocab : 9703
------------------------------
Top 9703 most frequent words:
```

In [107]:

```
not_hate_wocab = freq_wrds_class(data_not_hate, n = 11672)
```

```
Total words in vocab : 11672
------------------------------
Top 11672 most frequent words:
```

In [108]:

```
hate_words = [t[0] for t in hate_wocab]
not_hate_words = [t[0] for t in not_hate_wocab]
```

In [109]:

```
exclusive_hate_wrds = [w for w in hate_words if w not in not_hate_words]
print(len(exclusive_hate_wrds))
```

```
6312
```

In [110]:

```
exclusive_hate_wrds[:10]
```

Out[110]:

```
['faggot',
 'fuck',
 'nigga',
 'fucking',
 'niggas',
 'bitches',
 'feminists',
 'niggers',
 'yesyoure',
 'feminist']
```

In [111]:

```
len(set(exclusive_hate_wrds))
```

Out[111]:

```
6312
```

As we can see a majority of hate speech words are racist, sexist and homophobic slurs that exceed cultural slang. The fact that these words are unique to the "Hate Speech" label affirm that it's indeed hate speech that should be flagged and taken down.

Visualizing Unique Words with Venn Diagram

In [113]:

```
import matplotlib_venn as venn
from matplotlib_venn import venn2, venn2_circles, venn3, venn3_circles
import matplotlib.pyplot as plt
```

In [114]:

```
plt.figure(figsize=(10,10), facecolor='w')
venn2([set(hate_words), set(not_hate_words)], set_labels = ('Hate Speech', 'Not Hate Spee
ch'))
plt.title('Comparison of Unique Words in Each Corpus Label')
# plt.savefig('../images/venn.png')
plt.show()
```

Out[114]:

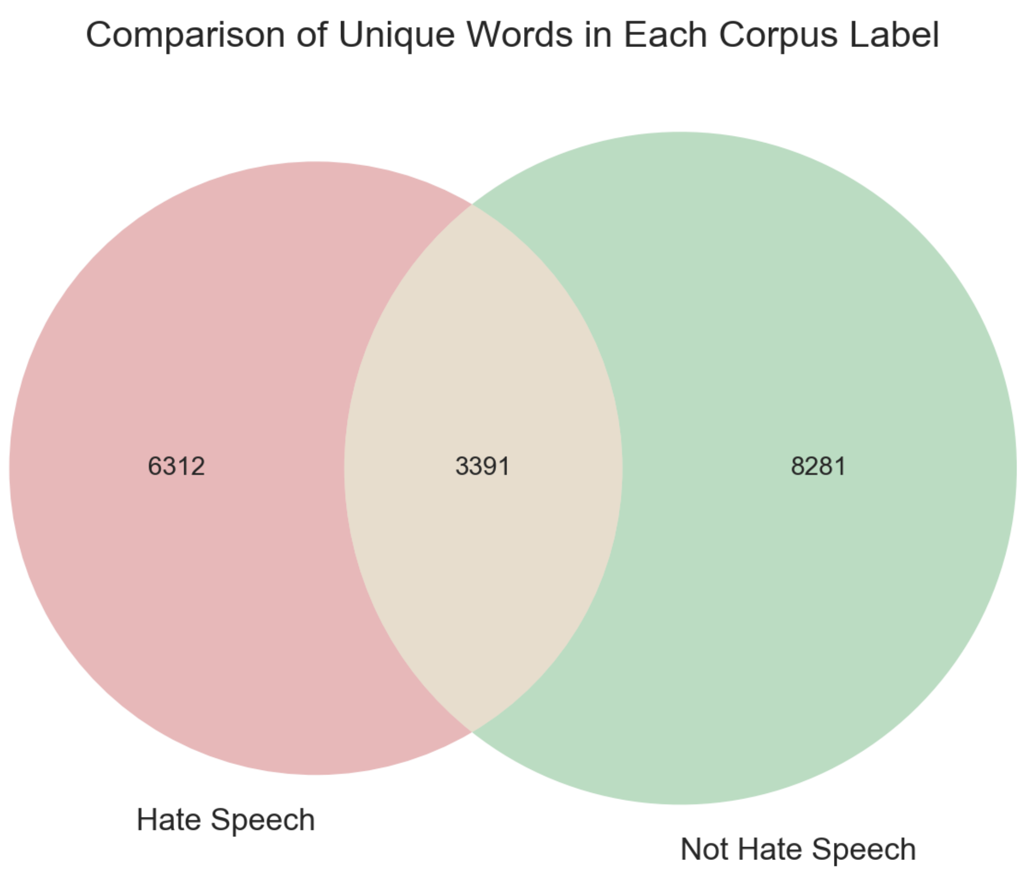
```
<Figure size 720x720 with 0 Axes>
```

Out[114]:

```
<matplotlib_venn._common.VennDiagram at 0x7fc89f262a90>
```

Out[114]:

```
Text(0.5,1,'Comparison of Unique Words in Each Corpus Label')
```



Create Baseline Models

In [115]:

```
X_lem = lemmatized_data
y_lem = df['class']
```

In [116]:

```
X_train_lem, X_test_lem, y_train_lem, y_test_lem = train_test_split(X_lem, y_lem, test_s
ize=0.20, random_state=1)
tfidf = TfidfVectorizer()  # can add unigram , add stop words possible

tfidf_data_train_lem = tfidf.fit_transform(X_train_lem)  # make sure in train
tfidf_data_test_lem = tfidf.transform(X_test_lem)    # make sure on test

tfidf_data_train_lem
```

Out[116]:

```
<6668x14428 sparse matrix of type '<class 'numpy.float64'>'
 with 53078 stored elements in Compressed Sparse Row format>
```

In [117]:

```
non_zero_cols = tfidf_data_train_lem.nnz / float(tfidf_data_train_lem.shape[0])
print("Average Number of Non-Zero Elements in Vectorized Tweets: {}".format(non_zero_cols
))

percent_sparse = 1 - (non_zero_cols / float(tfidf_data_train_lem.shape[1]))
print('Percentage of columns containing ZERO: {}'.format(percent_sparse))
```

```
Average Number of Non-Zero Elements in Vectorized Tweets: 7.960107978404319
Percentage of columns containing ZERO: 0.9994482874980313
```

99.9% of the columns contain a zero, meaning that's a very sparse matrix

In [118]:

```
# Lets Keep All models Results in dictionary for future visualization
eval_metrics_dict = {}
```

Random Forest Baseline

In [121]:

```
rf_classifier_lem = RandomForestClassifier(n_estimators=100, random_state=0)
rf_classifier_lem.fit(tfidf_data_train_lem, y_train_lem)
rf_test_preds_lem = rf_classifier_lem.predict(tfidf_data_test_lem)
```

Out[121]:

```
RandomForestClassifier(random_state=0)
```

In [122]:

```
rf_precision = precision_score(y_test_lem, rf_test_preds_lem)
rf_recall = recall_score(y_test_lem, rf_test_preds_lem)
rf_acc_score = accuracy_score(y_test_lem, rf_test_preds_lem)
rf_f1_score = f1_score(y_test_lem, rf_test_preds_lem)
print('Random Forest with Lemmatization Features:')

print('Precision: {:.4}'.format(rf_precision))
print('Recall: {:.4}'.format(rf_recall))

print("Testing Accuracy: {:.4}".format(rf_acc_score))
print("F1 Score: {:.4}".format(rf_f1_score))
```

```
Random Forest with Lemmatization Features:
```

Random Forest with Lemmatization features:
Precision: 0.854
Recall: 0.9285
Testing Accuracy: 0.886
F1 Score: 0.8897

In [124]:

```
fig, ax = plt.subplots(figsize=(6,6))
mat = confusion_matrix(y_test_lem, rf_test_preds_lem)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=['Not_Hate_Speech', 'Hate_Speech'], yticklabels=['Not_Hate_Speec
h', 'Hate_Speech'])
plt.xlabel('true label')
plt.ylabel('predicted label')
plt.title('Confusion Matrix for \n Random Forest with TFIDF Vectorizer')
#plt.savefig('../images/matrix.png')
plt.show()
```

Out[124]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc8aeef8850>
```

Out[124]:

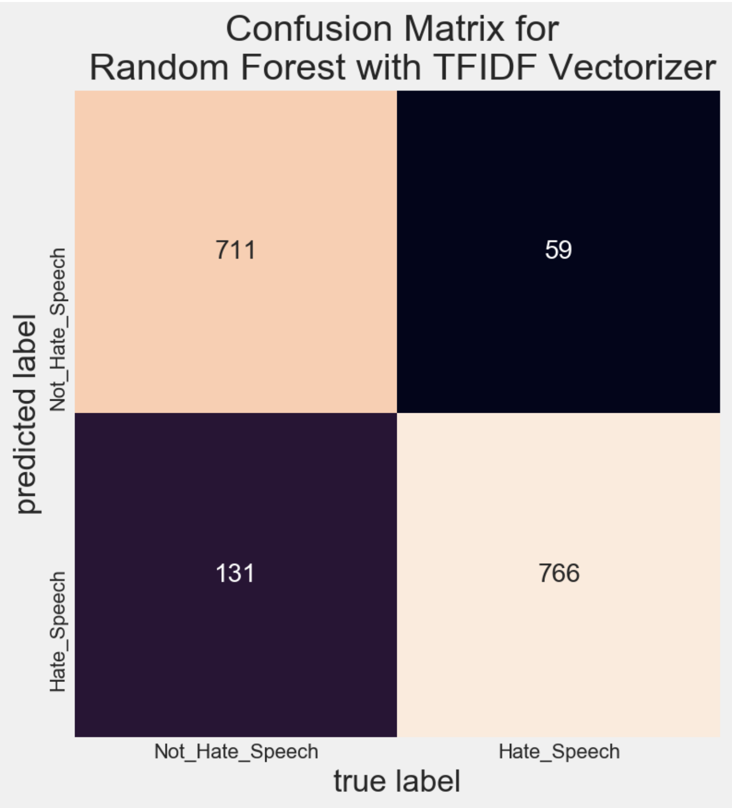
```
Text(0.5,12.6931,'true label')
```

Out[124]:

```
Text(29.9731,0.5,'predicted label')
```

Out[124]:

```
Text(0.5,1,'Confusion Matrix for \n Random Forest with TFIDF Vectorizer')
```



In [125]:

```
eval_metrics_dict['Random Forest Baseline'] = {'precision' : '{:.4}'.format(rf_precision
), 'recall': '{:.4}'.format(rf_recall), 'f1-score': '{:.4}'.format(rf_f1_score) }
```

Logistic Regression Baseline

In [126]:

```
logreg = LogisticRegression(random_state = 32)
```

```
logreg.fit(tfidf_data_train_lem, y_train_lem)
logreg_test_preds = logreg.predict(tfidf_data_test_lem)
```

Out[126]:

```
LogisticRegression(random_state=32)
```

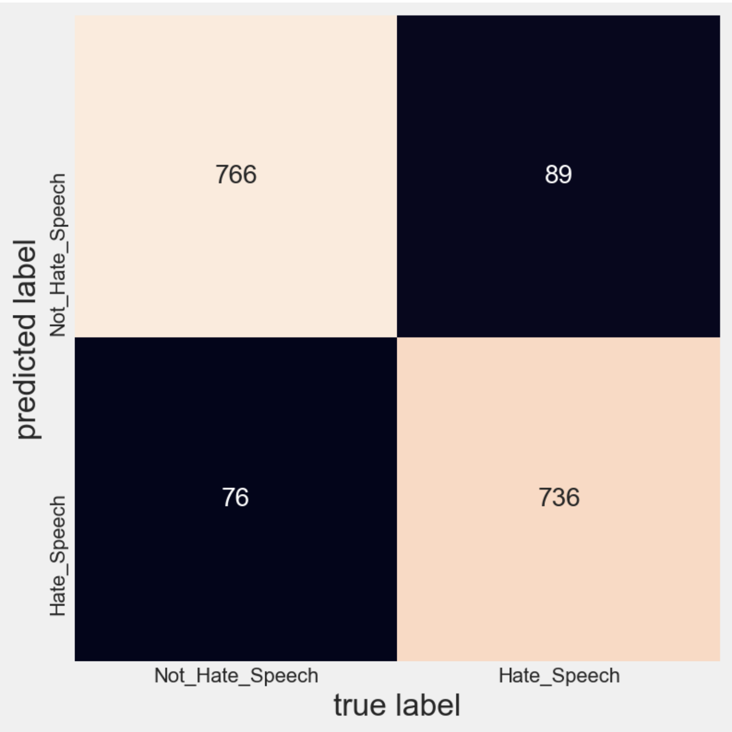
In [127]:

```
log_precision = precision_score(y_test_lem, logreg_test_preds)
log_recall = recall_score(y_test_lem, logreg_test_preds)
log_acc_score = accuracy_score(y_test_lem, logreg_test_preds)
log_f1_score = f1_score(y_test_lem, logreg_test_preds)
print('Random Forest with Lemmatization Features:')

print('Precision: {:.4}'.format(log_precision))
print('Recall: {:.4}'.format(log_recall))

print("Testing Accuracy: {:.4}".format(log_acc_score))
print("F1 Score: {:.4}".format(log_f1_score))
```

```
Random Forest with Lemmatization Features:
Precision: 0.9064
Recall: 0.8921
Testing Accuracy: 0.901
F1 Score: 0.8992
```

In [128]:

```
fig, ax = plt.subplots(figsize=(6,6))
mat = confusion_matrix(y_test_lem, logreg_test_preds)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=['Not_Hate_Speech', 'Hate_Speech'], yticklabels=['Not_Hate_Speec
h', 'Hate_Speech'])
plt.xlabel('true label')
plt.ylabel('predicted label');
```



In [129]:

```
eval_metrics_dict['Logistic Regression Baseline'] = {'precision' : '{:.4}'.format(log_pr
ecision), 'recall': '{:.4}'.format(log_recall), 'f1-score': '{:.4}'.format(log_f1_score)
}
```

Naive Bayes Baseline

In [131]:

In [131]:

```
nb = MultinomialNB()
nb.fit(tfidf_data_train_lem, y_train_lem)
nb_test_preds = nb.predict(tfidf_data_test_lem)
```

Out[131]:

```
MultinomialNB()
```

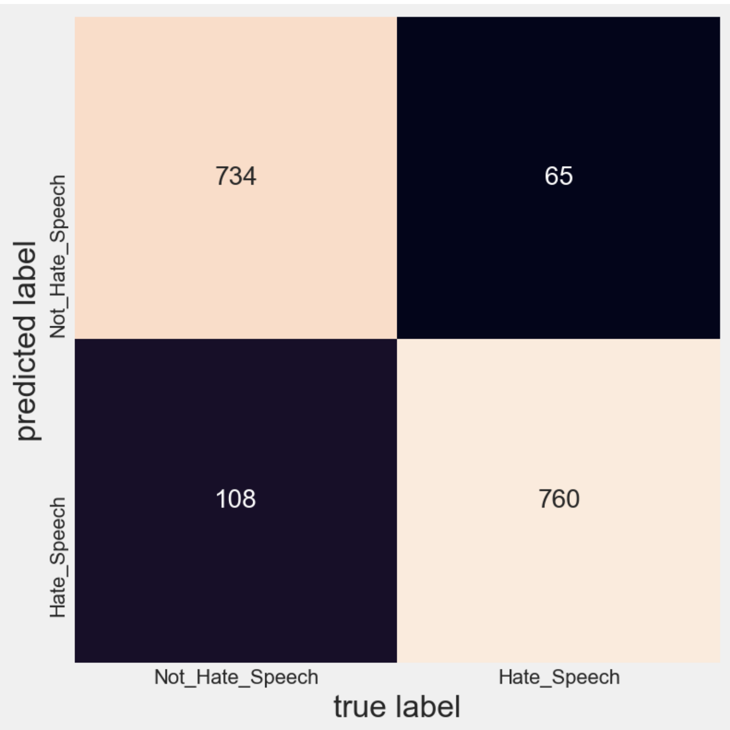
In [132]:

```
nb_precision = precision_score(y_test_lem, nb_test_preds)
nb_recall = recall_score(y_test_lem, nb_test_preds)
nb_acc_score = accuracy_score(y_test_lem, nb_test_preds)
nb_f1_score = f1_score(y_test_lem, nb_test_preds)
print('Random Forest with Lemmatization Features:')

print('Precision: {:.4}'.format(nb_precision))
print('Recall: {:.4}'.format(nb_recall))

print("Testing Accuracy: {:.4}".format(nb_acc_score))
print("F1 Score: {:.4}".format(nb_f1_score))
```

```
Random Forest with Lemmatization Features:
Precision: 0.8756
Recall: 0.9212
Testing Accuracy: 0.8962
F1 Score: 0.8978
```

In [133]:

```
fig, ax = plt.subplots(figsize=(6,6))
mat = confusion_matrix(y_test_lem, nb_test_preds)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=['Not_Hate_Speech', 'Hate_Speech'], yticklabels=['Not_Hate_Speec
h', 'Hate_Speech'])
plt.xlabel('true label')
plt.ylabel('predicted label');
```



In [134]:

```
eval_metrics_dict['Naive Bayes Baseline'] = {'precision' : '{:.4}'.format(nb_precision),
'recall': '{:.4}'.format(nb_recall), 'f1-score': '{:.4}'.format(nb_f1_score) }
```

In [135]:

```
baseline_results = pd.DataFrame(eval_metrics_dict).T
```

In [136]:

```
baseline_results
```

Out[136]:

	precision	recall	f1-score
Random Forest Baseline	0.854	0.9285	0.8897
Logistic Regression Baseline	0.9064	0.8921	0.8992
Naive Bayes Baseline	0.8756	0.9212	0.8978

As our major evaluation metrics will be Recall and F1 score, based on models perfomance - best results was achived with Random Forest Model

Tuning Model

In [139]:

```
from sklearn.model_selection import GridSearchCV
```

In [140]:

```
# Number of trees in random forrest
n_estimators = [int(x) for x in np.linspace(start = 50, stop = 200, num =5)]

# number of features to consider at each split
max_features = ['auto', 'sqrt']

# Max number of levels in tree
max_depth = [2,4]

# min number of samples required to splid the node
min_samples_split =[2,5]

# min number of samples required at each leaf node
min_samples_leaf =[1,2]

#Method of selecting samples for training each tree
# bootstrap =[True,False]
```

In [141]:

```
param_grid = {'n_estimators' : n_estimators,
              'max_features' : max_features,
              'max_depth' : max_depth,
              'min_samples_split' : min_samples_split,
              'min_samples_leaf' : min_samples_leaf}
              #'bootstrap' : bootstrap }
param_grid
```

Out[141]:

```
{'n_estimators': [50, 87, 125, 162, 200],
 'max_features': ['auto', 'sqrt'],
 'max_depth': [2, 4],
 'min_samples_split': [2, 5],
 'min_samples_leaf': [1, 2]}
```

In [142]:

```
rf_momdel = RandomForestClassifier()
rf_grid = GridSearchCV(estimator = rf_momdel, param_grid = param_grid, cv = 3, verbose =
3, n_jobs = 4, scoring = 'recall' )
```

In [143]:

```
rf_grid.fit(tfidf_data_train_lem, y_train_lem)
```

```
Fitting 3 folds for each of 80 candidates, totalling 240 fits
```

Out[143]:

```
GridSearchCV(cv=3, estimator=RandomForestClassifier(), n_jobs=4,
             param_grid={'max_depth': [2, 4], 'max_features': ['auto', 'sqrt'],
                         'min_samples_leaf': [1, 2],
                         'min_samples_split': [2, 5],
                         'n_estimators': [50, 87, 125, 162, 200]},
             scoring='recall', verbose=3)
```

In [144]:

```
rf_grid.best_params_
```

Out[144]:

```
{'max_depth': 2,
 'max_features': 'auto',
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 50}
```

In [149]:

```
rf_clf_tunned = RandomForestClassifier(n_estimators = 100, max_depth = 2, max_features =
'auto', min_samples_split=2)
rf_clf_tunned.fit(tfidf_data_train_lem, y_train_lem)
t_rf_test_preds_lem = rf_clf_tunned.predict(tfidf_data_test_lem)
```

Out[149]:

```
RandomForestClassifier(max_depth=2)
```

In [150]:

```
t_rf_precision = precision_score(y_test_lem, t_rf_test_preds_lem)
t_rf_recall = recall_score(y_test_lem, t_rf_test_preds_lem)
t_rf_acc_score = accuracy_score(y_test_lem, t_rf_test_preds_lem)
t_rf_f1_score = f1_score(y_test_lem, t_rf_test_preds_lem)
print('Random Forest with Hyper Parameters selected with GridSearch:')

print('Precision: {:.4}'.format(t_rf_precision))
print('Recall: {:.4}'.format(t_rf_recall))

print("Testing Accuracy: {:.4}".format(t_rf_acc_score))
print("F1 Score: {:.4}".format(t_rf_f1_score))
```

```
Random Forest with Hyper Parameters selected with GridSearch:
Precision: 0.7963
Recall: 0.7248
Testing Accuracy: 0.772
F1 Score: 0.7589
```

In [151]:

```
fig, ax = plt.subplots(figsize=(6,6))
mat = confusion_matrix(y_test_lem, t_rf_test_preds_lem)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=['Not_Hate_Speech', 'Hate_Speech'], yticklabels=['Not_Hate_Speec
h', 'Hate_Speech'])
plt.xlabel('true label')
plt.ylabel('predicted label')
plt.show()
```

Out[151]:

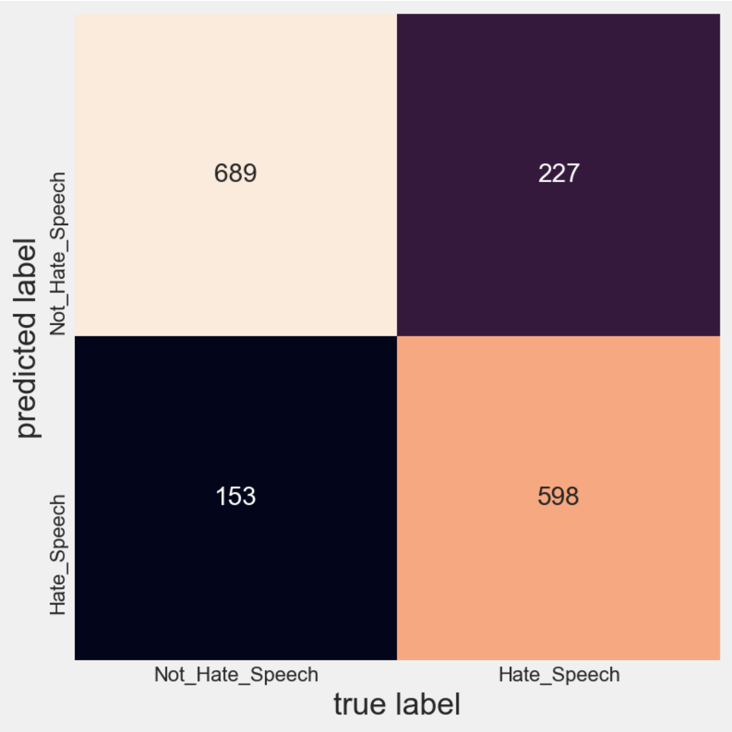
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc84cb5afd0>
```

Out[151]:

```
Text(0.5,12.6931,'true label')
```

Out[151]:

```
Text(29.9731,0.5,'predicted label')
```



In [148]:

```
t_rf_train_preds_lem = rf_clf_tunned.predict(tfidf_data_train_lem)

t_rf_precision_train = precision_score(y_train_lem, t_rf_train_preds_lem)
t_rf_recall_train = recall_score(y_train_lem, t_rf_train_preds_lem)
t_rf_acc_score_train = accuracy_score(y_train_lem, t_rf_train_preds_lem)
t_rf_f1_score_train = f1_score(y_train_lem, t_rf_train_preds_lem)
print('Random Forest with Hyper Parameters selected with GridSearch:')

print('Precision: {:.4}'.format(t_rf_precision_train))
print('Recall: {:.4}'.format(t_rf_recall_train))

print("Training Accuracy: {:.4}".format(t_rf_acc_score_train))
print("F1 Score: {:.4}".format(t_rf_f1_score_train))
```

```
Random Forest with Hyper Parameters selected with GridSearch:
Precision: 0.7388
Recall: 0.954
Training Accuracy: 0.8076
F1 Score: 0.8327
```

In [ ]:

```
```