Final Report

Name: Robb Alexander and Ryan Bailis
Class: CSCI349
Semester: 2021SP
Instructor: Brian King

In [27]:
```
import re
import string
from itertools import cycle
from collections import Counter
from IPython.core.display import display

import nltk
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from wordcloud import WordCloud
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import MWETokenizer
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import fpgrowth, association_rules
from sklearn.pipeline import Pipeline
from sklearn.linear_model import SGDClassifier
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.model_selection import train_test_split, cross_val_predict, GridSearchCV
from sklearn.metrics import make_scorer, fbeta_score, precision_recall_curve
from sklearn.metrics import confusion_matrix, classification_report, average_precision_score

import plotly.express as px
from plotly.offline import init_notebook_mode
init_notebook_mode(connected = True)
```

Introduction

This project is the final project for Brian King's CSCI349, Introduction to Data Mining, at Bucknell University. We are working on a classification project involving Hate Speech and Offensive Language.

The internet is growing at an insane pace that humans cannot monitor everything posted, but site have an obligation to prevent the spreading of hate and bullying through text. If we could categorize this and punish accordingly, it would free up lots of time for humans to just be reviewers.

We have a tweet, and it is assigned to a label of either `hatespeech`, `offensive language`, or `neither`. Our goal is to create a model that can learn from those human labeled tweets to extrapolate into a general formula to determine which class it falls under.

Data

Variables

count = number of CrowdFlower users who coded each tweet (min is 3, sometimes more users coded a tweet when judgments were determined to be unreliable by CF).

hate_speech = number of CF users who judged the tweet to be hate speech.

offensive_language = number of CF users who judged the tweet to be offensive.

neither = number of CF users who judged the tweet to be neither offensive nor non-offensive.

class = class label for majority of CF users. 0 - hate speech 1 - offensive language 2 - neither

Data Info

The data is from
"Repository for Thomas Davidson, Dana Warmsley, Michael Macy, and Ingmar Weber. 2017. "Automated Hate Speech Detection and the Problem of Offensive Language." ICWSM. You read the paper here."

Each row is a tweet that has been ran through a crowdsourcing platform to label how much of hatespeech it is. Each tweet was taken from Twitter itself (unknown consent from the original poster)

Motivation

Now we see that essentially the class and tweet column is the most important ones, and the other are auxiliary ones to reinforce the class column.py

We want to take the words in the tweet and see what it is classified as. This then can be modeled to new data online to flag what can be considered hatespeech.
Right now we will look into all the variables, but for modeling purposes in the future we will stick to just the two most important ones. text -> class.

Data

In [7]:
```
df = pd.read_csv("data/labeled_data.csv", index_col=0)
df
```

Out[7]:

	count	hate_speech	offensive_language	neither	class	tweet
0	3	0	0	3	2	!!! RT @mayasolovely: As a woman you shouldn't...
1	3	0	3	0	1	!!!!! RT @mleew17: boy dats cold...tyga dwn ba...
2	3	0	3	0	1	!!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...
3	3	0	2	1	1	!!!!!!!!! RT @C_G_Anderson: @viva_based she lo...
4	6	0	6	0	1	!!!!!!!!!!!! RT @ShenikaRoberts: The shit you...
...	...	...	...	...	...	...
25291	3	0	2	1	1	you's a muthaf***in lie “@LifeAsKing: @2...
25292	3	0	1	2	2	you've gone and broke the wrong heart baby, an...
25294	3	0	3	0	1	young buck wanna eat!!.. dat nigguh like I ain...
25295	6	0	6	0	1	youu got wild bitches tellin you lies
25296	3	0	0	3	2	~~Ruffled | Ntac Eileen Dahlia - Beautiful col...

24783 rows × 6 columns

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 24783 entries, 0 to 25296
Data columns (total 6 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   count               24783 non-null  int64 
 1   hate_speech         24783 non-null  int64 
 2   offensive_language  24783 non-null  int64 
 3   neither             24783 non-null  int64 
 4   class               24783 non-null  int64 
 5   tweet               24783 non-null  object
dtypes: int64(5), object(1)
memory usage: 1.3+ MB
```

Research

Links

Automated Hate Speech Detection and the Problem of Offensive Language

PDF

Quotes

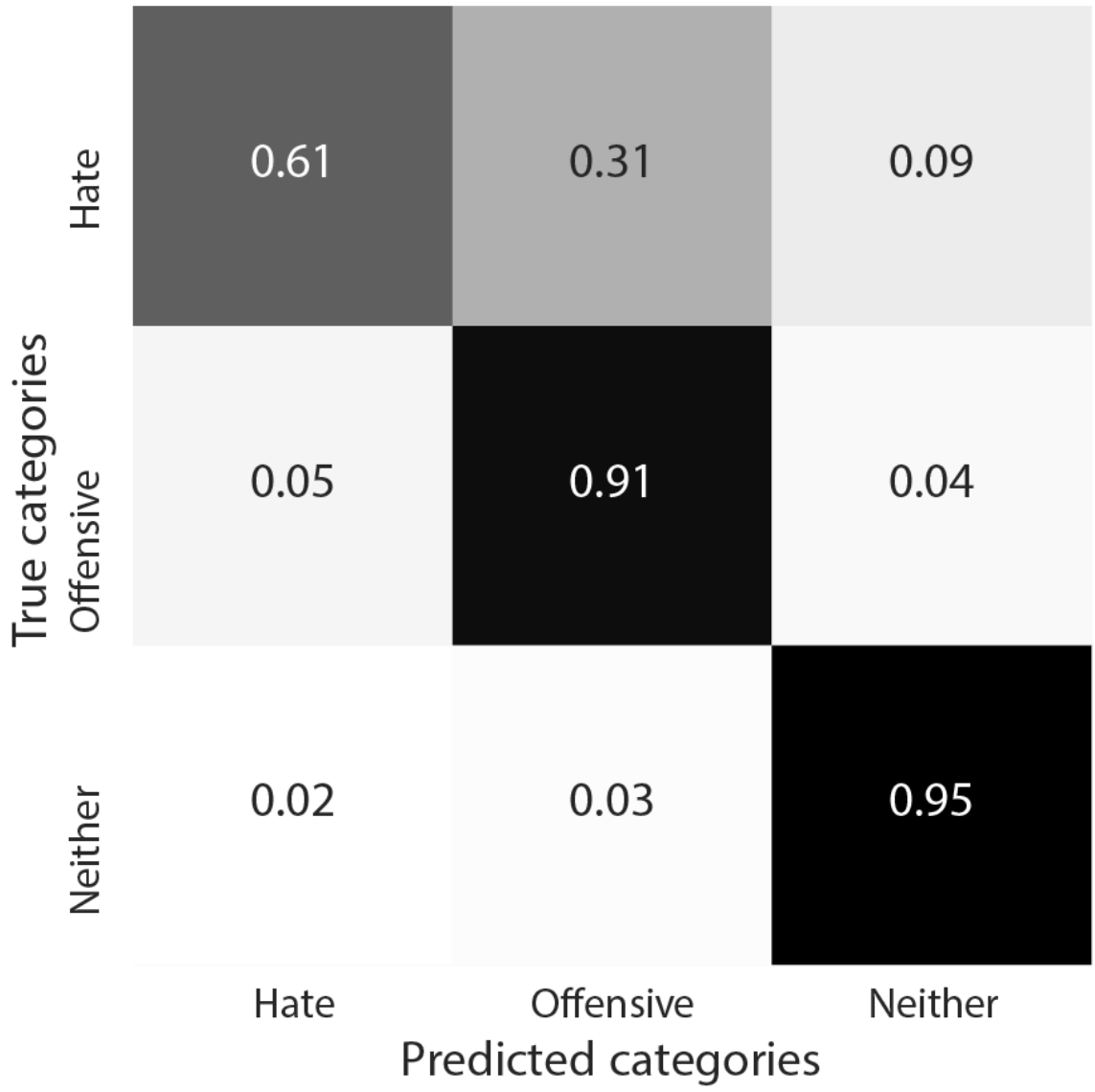
"Only 5% of tweets were coded as hate speech by the majority of coders and only 1.3% were coded unanimously, demonstrating the imprecision of the Hatebase lexicon"

"We decided to use a logistic regression with L2 regularization for the final model as it more readily allows us to examine the predicted probabilities of class membership"

Results

In [30]:
```
from IPython import display
display.Image("results.png")
```

Out[30]:



After finishing our own modeling we read the research paper and noticed that the research paper had a similar confusion matrix as us besides the neither category. The performance of hatespeech misclassified with offensive is a key point in both our results.

Data Prep

Apply Functions for cleaning

In [15]:
```
def remove_punctuation(tweet):
    return tweet.translate(str.maketrans(string.punctuation, ' '*len(string.punctuation)))

def remove_link(tweet):
    return re.sub(r'http\S+', ' ', tweet)

def remove_at(tweet):
    return re.sub(r'@\S+', ' ', tweet)

def remove_symbols_emoji(tweet):
    return re.sub(r'&.+?;', ' ', tweet)

def remove_retweet(tweet):
    return re.sub(r'RT', ' ', tweet)
```

Read Data

In [9]:
```
df["class"] = pd.Categorical(df["class"])
df.iloc[:,0:4] = df.iloc[:,0:4].apply(pd.to_numeric, downcast="unsigned")


df_old = df.copy()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 24783 entries, 0 to 25296
Data columns (total 6 columns):
 #   Column              Non-Null Count  Dtype   
---  ------              --------------  -----   
 0   count               24783 non-null  uint8   
 1   hate_speech         24783 non-null  uint8   
 2   offensive_language  24783 non-null  uint8   
 3   neither             24783 non-null  uint8   
 4   class               24783 non-null  category
 5   tweet               24783 non-null  object  
dtypes: category(1), object(1), uint8(4)
memory usage: 508.4+ KB
```

Apply cleaning to each tweet

In [16]:
```
df["tweet"] = df["tweet"].apply(remove_link)
df["tweet"] = df["tweet"].apply(remove_at)
df["tweet"] = df["tweet"].apply(remove_symbols_emoji)
df["tweet"] = df["tweet"].apply(remove_retweet)
df["tweet"] = df["tweet"].apply(remove_punctuation)
df["tweet"] = df["tweet"].apply(str.lower)
```

Split tweets and remove common english words, then lemmatize (remove plurals etc.)

In [17]:
```
sw = set(map(remove_punctuation, stopwords.words('english')))
bad_lemmas = set(['as', 'ass', 'was'])
wnl = WordNetLemmatizer()
combiner = MWETokenizer([('wan','na'), ('gon','na'),('got','ta')])

df["tweet"] = df["tweet"].apply(nltk.word_tokenize)
df["tweet"] = df["tweet"].map(lambda x: [wnl.lemmatize(i) if i not in bad_lemmas else i for i
df["tweet"] = df["tweet"].map(lambda x: [i for i in x if i not in sw])
df["tweet"] = df["tweet"].apply(combiner.tokenize)
df["tweet"] = df["tweet"].map(lambda x: ' '.join(x))
```

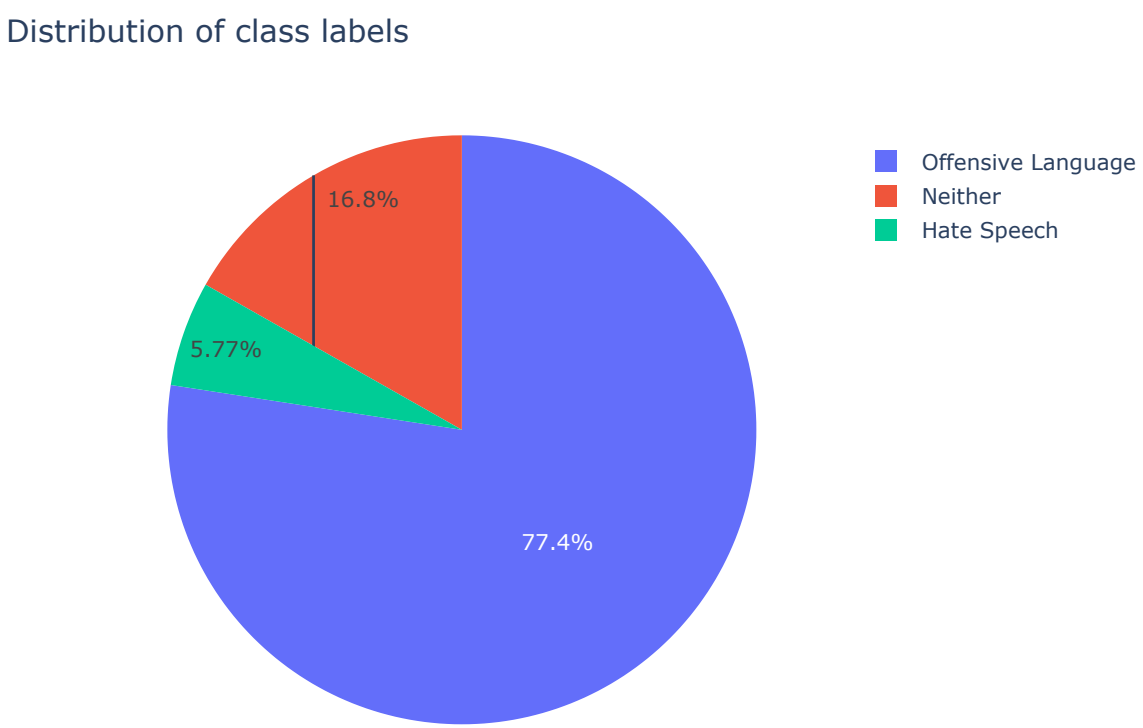
Rename categories

In [18]:
```
class_map = { 0 : "Hate Speech",
              1 : "Offensive Language",
              2 : "Neither" }

df["class"].cat.rename_categories(class_map, inplace=True)
```

Visualization

This pie graph shows the imbalnace of class assignments, so we need to account for that during modeling somehow.

In [25]:
```
px.pie(df["class"].value_counts(), values='class', names=df["class"].value_counts().index, ti
```

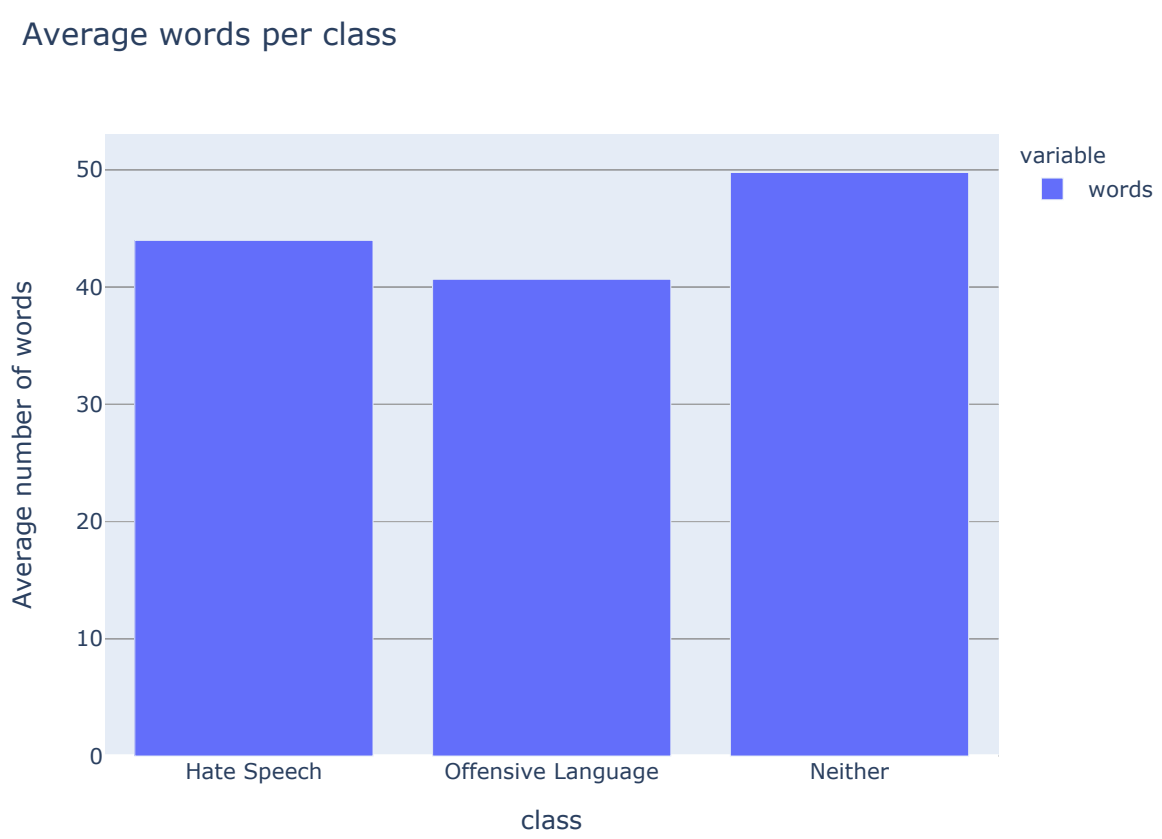


Percent of unanimous hate speech labeled.

In [20]:
```
round(len(np.where(df_old["hate_speech"] == df_old["count"] )[0]) / len(df_old[df_old["class"
```

Out[20]: 18.39

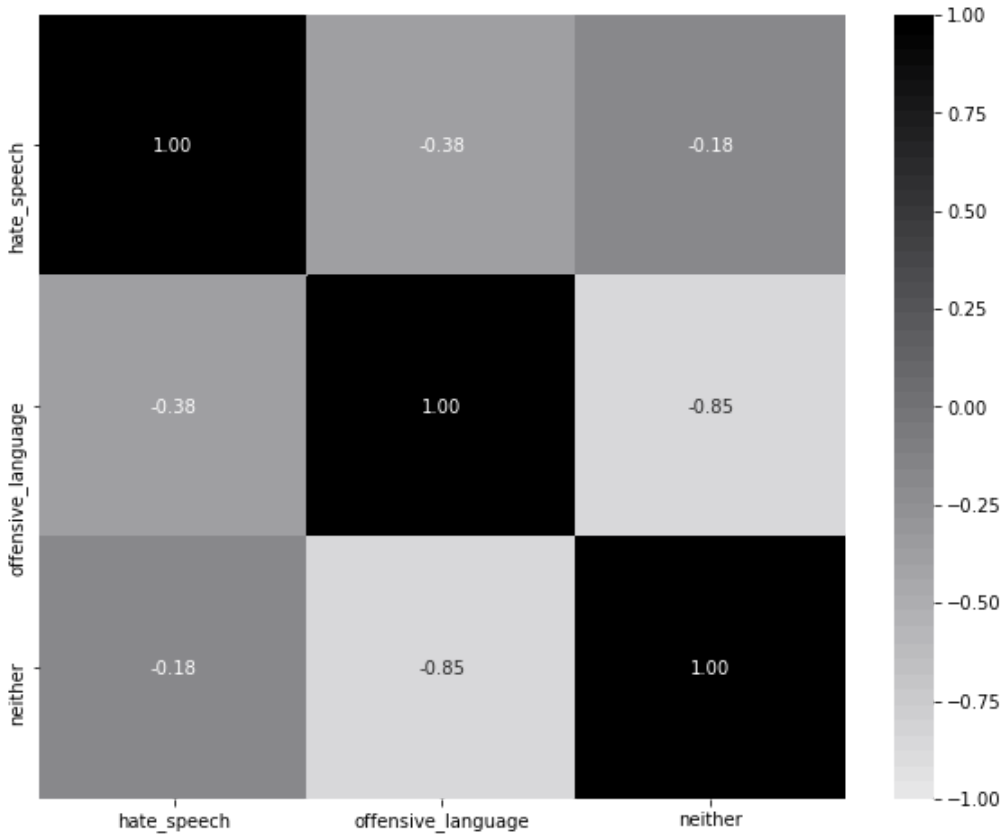
In [26]:
```
df["words"] = df["tweet"].str.len()

fig = px.bar(df[["class","tweet","words"]].groupby(by='class')["words"].mean(),labels={'value
fig.update_yaxes(range=[0, 53])
fig.show()

df.drop("words", axis=1, inplace=True)
```



Correlation between the labeled counts

In [39]:
```
plt.figure(figsize=(10,8))
color = sns.cubehelix_palette(50, hue=0.05, rot=0, light=0.9, dark=0)

sns.heatmap(df.iloc[:,1:4].div(df["count"], axis=0).corr(),
            center=0, vmin=-1, vmax=1, cmap=color,
            annot=True, fmt=".2f")
```

Out[39]: <AxesSubplot:>



Word Clouds of each class's common words (!!!TW: hate speech!!!)

In [40]:
```
ol = df[df["class"] == "Offensive Language"].tweet.apply(nltk.word_tokenize)
hs = df[df["class"] == "Hate Speech"].tweet.apply(nltk.word_tokenize)
n = df[df["class"] == "Neither"].tweet.apply(nltk.word_tokenize)


def get_word_freq(tweets):
    wnl = WordNetLemmatizer()
    bad_lemmas = set(['ass', 'was'])

    tweets = tweets.map(lambda x: [wnl.lemmatize(i) if i not in bad_lemmas else i for i in x]
    word_freq = Counter(x for xs in tweets for x in set(xs))
    sw = stopwords.words('english')
    sw = set(map(remove_punctuation, sw))

    for k in sw:
        word_freq.pop(k, None)

    wc = WordCloud(background_color="white", max_words=50, max_font_size=50, relative_scaling
    plt.figure()
    plt.imshow(wc)
    plt.axis("off")
    plt.show()
    return word_freq
```

Offensive Language

In [41]:
```
wf_ol = get_word_freq(ol)
```

Hate Speech

In [42]:
```
wf_hs = get_word_freq(hs)
```

Neither

In [43]:
```
wf_n = get_word_freq(n)
```

Model

Model Discussion

The model that we chose in the end was the Stochastic Gradient Descent. The Convoluted Neural Network didn't work as well as we expected, albeit the average macro-f1 score increased, but the hatespeech recognition was reduced. We thought the gloVe word embedding would at least increase the neither category accuracy, but it did help for some semantic examples such as 'I love my mom'. But in the end the CNN put too much focus on offensive language and detracted from the goal of categorizing hatespeech.

We also used a ComplementNB after researching a way to use naive bayes on imbalanced datasets. Due to the nature of naive bayes, the imbalance will destroy the whole way it works, thus we learned that it said ComplementNB solved many of those problems. The produced results were much better than our original simple naive bayes, but we wanted something more complex.

We also chose to use an ensemble classifier, ExtraTreesClassifier, mainly to prevent over-fitting, while also adding more layers thus utilizing the power of trees and forest type classifiers. This had a much higher wall time with almost identical performance from the SGD.

Final Model

In [51]:
```
tweet = df["tweet"].values
y = df["class"].values

pipe = Pipeline([('vect', CountVectorizer()),
                 ('tfidf', TfidfTransformer()),
                 ('clf', SGDClassifier(loss="modified_huber", alpha=0.00045, tol=0.001, max_i
                 ])

y_pred = cross_val_predict(pipe, tweet, y, cv = 10)
```

Hyper parameters

We ran a grid search with the SGD using the `loss`, `alpha`, `penalty`, `max_iter`, and `tol` parameters.

In [47]:
```
param_grid = {
    'clf__loss' : ["hinge","modified_huber"],
    'clf__alpha' : [round(i, 5) for i in np.linspace(0.00005,0.0005, 10)],
    'clf__penalty' : ["l1", "l2"],
    'clf__tol' : [round(i, 5) for i in np.linspace(0.0005,0.005, 10)],
    'clf__max_iter' : [round(i) for i in np.linspace(100,2000, 10)]
}
```

We used the fbeta scoring with the `beta` value of 0.3 in order to optimize the precision of the hate speech class.

In [111…
```
f2_score = make_scorer(fbeta_score, beta=0.2, average="macro", pos_label=1)

grid = GridSearchCV(pipe, param_grid, cv=10, scoring=f2_score, n_jobs=4, verbose=0)
grid_result = grid.fit(tweet, y)
```

In [112…]
```
df_grid_results = pd.DataFrame(grid_result.cv_results_)
df_grid_results = df_grid_results.sort_values(by="mean_test_score", ascending=False)[:5]

for i, row in df_grid_results.iterrows():
    print(row['params'])
    print(row['mean_test_score'])
    print("-------")
```

```
{'clf__alpha': 0.00045, 'clf__loss': 'modified_huber', 'clf__max_iter': 1578, 'clf__penalty':
'l2', 'clf__tol': 0.001}
0.7713001350305142
-------
{'clf__alpha': 0.00025, 'clf__loss': 'modified_huber', 'clf__max_iter': 1789, 'clf__penalty':
'l2', 'clf__tol': 0.0045}
0.7706986873921461
-------
{'clf__alpha': 0.00025, 'clf__loss': 'modified_huber', 'clf__max_iter': 1156, 'clf__penalty':
'l2', 'clf__tol': 0.001}
0.7706604668285255
-------
{'clf__alpha': 0.00045, 'clf__loss': 'modified_huber', 'clf__max_iter': 2000, 'clf__penalty':
'l2', 'clf__tol': 0.003}
0.7705527872139408
-------
{'clf__alpha': 0.00045, 'clf__loss': 'modified_huber', 'clf__max_iter': 1578, 'clf__penalty':
'l2', 'clf__tol': 0.004}
0.7705381344538134
-------
```

Model Performance

Classification report

In the end even by not optimizing for the accuracy or f1_macro, we still had a 70%, which was on par with the CNN using semantic embeddings. But here we achieved a 57%vs42% hate speech precision, which is closer to the 61vs31% the researchers had, in relation to the CNN.

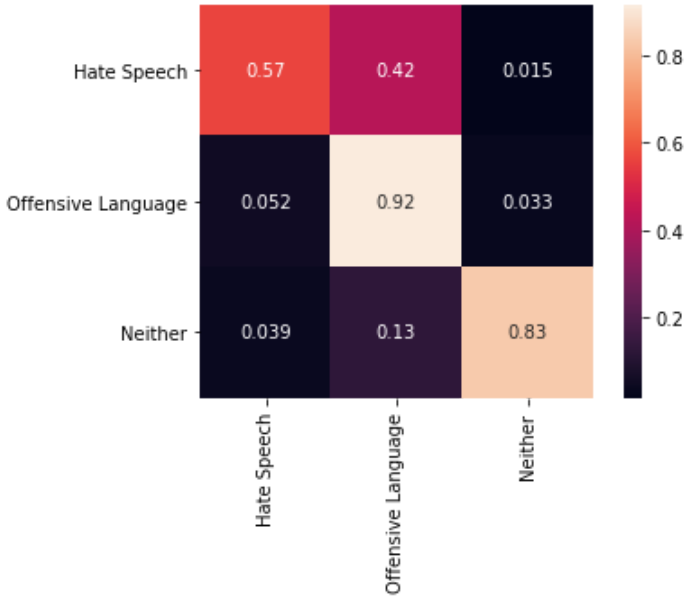
In [52]:
```
print(classification_report(y, y_pred, digits=3))
```

```
                    precision    recall  f1-score   support

       Hate Speech      0.565     0.157     0.246      1430
           Neither      0.835     0.837     0.836      4163
Offensive Language      0.915     0.964     0.939     19190

          accuracy                          0.896     24783
         macro avg      0.772     0.653     0.674     24783
      weighted avg      0.881     0.896     0.882     24783
```

Confusion Matrix

In [53]:
```
sns.heatmap(confusion_matrix(y, y_pred, normalize="pred", labels=list(class_map.values())).T,
```

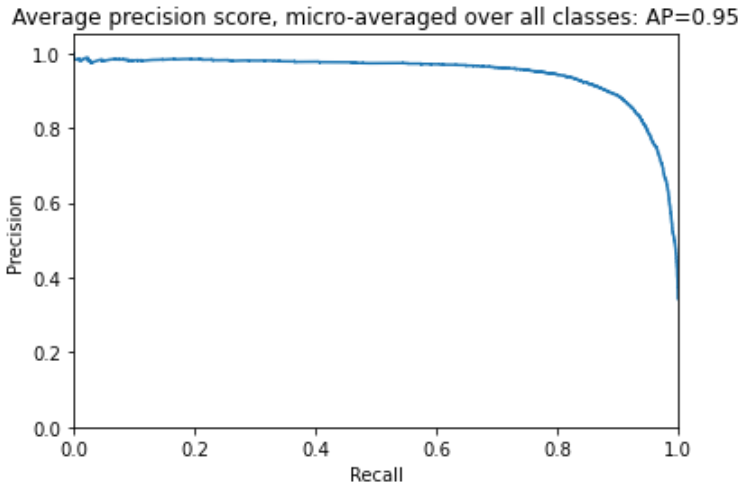
Out[53]: <AxesSubplot:>



Multiclass Prescision-Recall Curve

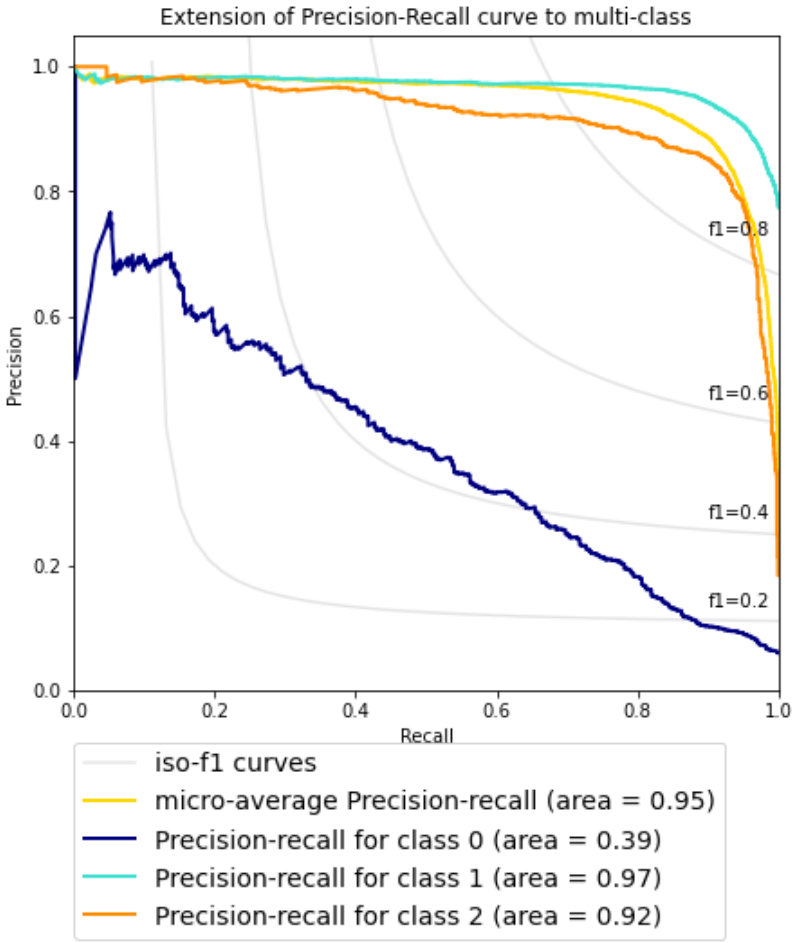
In [45]:
```
Y = label_binarize(y, classes=["Hate Speech", "Offensive Language", "Neither"])
n_classes = Y.shape[1]
X_train, X_test, Y_train, Y_test = train_test_split(tweet, Y, test_size=.3)

classifier = OneVsRestClassifier(pipe)
classifier.fit(X_train, Y_train)
y_score = classifier.decision_function(X_test)

precision = {}
recall = {}
average_precision = {}
for i in range(n_classes):
    precision[i], recall[i], _ = precision_recall_curve(Y_test[:, i], y_score[:, i])
    average_precision[i] = average_precision_score(Y_test[:, i], y_score[:, i])
precision["micro"], recall["micro"], _ = precision_recall_curve(Y_test.ravel(), y_score.ravel
average_precision["micro"] = average_precision_score(Y_test, y_score, average="micro")
plt.figure()
plt.step(recall['micro'], precision['micro'], where='post')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title('Average precision score, micro-averaged over all classes: AP={0:0.2f}'.format(aver
```

Out[45]: Text(0.5, 1.0, 'Average precision score, micro-averaged over all classes: AP=0.95')



In [47]:

```
colors = cycle(['navy', 'turquoise', 'darkorange', 'cornflowerblue', 'teal'])
plt.figure(figsize=(7, 8))
f_scores = np.linspace(0.2, 0.8, num=4)
lines = []
labels = []
for f_score in f_scores:
    x = np.linspace(0.01, 1)
    y = f_score * x / (2 * x - f_score)
    l, = plt.plot(x[y >= 0], y[y >= 0], color='gray', alpha=0.2)
    plt.annotate('f1={0:0.1f}'.format(f_score), xy=(0.9, y[45] + 0.02))
lines.append(l)
labels.append('iso-f1 curves')
l, = plt.plot(recall["micro"], precision["micro"], color='gold', lw=2)
lines.append(l)
labels.append('micro-average Precision-recall (area = {0:0.2f})'''.format(average_precision["
for i, color in zip(range(n_classes), colors):
    l, = plt.plot(recall[i], precision[i], color=color, lw=2)
    lines.append(l)
    labels.append('Precision-recall for class {0} (area = {1:0.2f})'
                  ''.format(i, average_precision[i]))
fig = plt.gcf()
fig.subplots_adjust(bottom=0.25)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Extension of Precision-Recall curve to multi-class')
plt.legend(lines, labels, loc=(0, -.38), prop=dict(size=14))
plt.show()
```



As you can see the class 0 (hate speech) p-r curve is struggling for the most part, while the other two are much higher up, with class 2 (neither) being slightly below class 1 (offensive language)

Discussion

We tried a plethora of classifiers, and as text classification was one of the topics we didn't go over class, it gave us a chance to struggle with the whole process and overcome it. A large challenge that we faced during our process was the gridsearch for the CNN, where we ported everything over to our communal PC, which has more processing power than our laptops. This was an issue because of the implementation of the packages and issues with n_jobs/multiprocessing. We looked through a few research papers to even try to solve the issue with the data in the first place and find a way to adjust how the counting/vectorizing of the text itself could increase performance.

Our choice to use nltk, and the power of nlp did increase the performance without much side effects, so we continued to use that model for all the classification. Every tweet is cleaned and also lemmantized. We learned the use of the Pipeline which is a great tool in sklearn, we used an ensemble method just to have some variety, we changed our initial MultinomialNB to ComplementNB after reading a research paper, and we upgraded tensorflow to have access to newer methods such as the powerful TextVectorization for the CNN.

The most interesting part of this project is how close our intuitive approach got to the actual research paper. All of this besides the neither category, where they scored in the high 90s. We hypothesize that this is from the extra parameters they passed into the classifier on top of the text, e.g. sentiment, vocab, and readability, lengths, @s, link counts, syllables. We think this would help the neither category as sentiment is helpful to differentiate the positive meaning phrases.

Association rules (related words)

We also tried to find word associations in the hate speech and offensive language classes. This was done in hopes to see if there were any groupings that could be misinterpreted

In [56]:
```
te = TransactionEncoder()
te_ary = te.fit_transform(list(map(nltk.word_tokenize,df[df['class'] == "Hate Speech"]["tweet
df_enc = pd.DataFrame(te_ary, columns=te.columns_)
df_apri = fpgrowth(df_enc, min_support=0.004, use_colnames=True)
len(df_apri)
```

Out[56]: 554

In [57]:
```
rules = association_rules(df_apri, metric="confidence", min_threshold=0.7)
sorted_rules = rules[rules.lift > 1.5].sort_values(by="lift", ascending=False)
sorted_rules
```

Out[57]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
19	(half)	(breed)	0.004895	0.004895	0.004196	0.857143	175.102041	0.004172	6.965734
18	(breed)	(half)	0.004895	0.004895	0.004196	0.857143	175.102041	0.004172	6.965734
12	(tom)	(uncle)	0.006294	0.007692	0.005594	0.888889	115.555556	0.005546	8.930769
11	(uncle)	(tom)	0.007692	0.006294	0.005594	0.727273	115.555556	0.005546	3.643590
14	(birthday)	(happy)	0.006294	0.009790	0.005594	0.888889	90.793651	0.005533	8.911888
9	(2, 3)	(1)	0.004895	0.011189	0.004196	0.857143	76.607143	0.004141	6.921678
8	(1, 3)	(2)	0.004196	0.018182	0.004196	1.000000	55.000000	0.004120	inf
16	(oh, ass)	(u)	0.004196	0.072028	0.004196	1.000000	13.883495	0.003894	inf
5	(white, full)	(trash)	0.005594	0.074126	0.005594	1.000000	13.490566	0.005180	inf
15	(trailer)	(trash)	0.004196	0.074126	0.004196	1.000000	13.490566	0.003885	inf
4	(trash, full)	(white)	0.005594	0.083217	0.005594	1.000000	12.016807	0.005129	inf
17	(oh, u)	(ass)	0.004895	0.087413	0.004196	0.857143	9.805714	0.003768	6.388112
2	(look, bitch)	(like)	0.006294	0.112587	0.004895	0.777778	6.908213	0.004187	3.993357
13	(word)	(nigger)	0.005594	0.111888	0.004196	0.750000	6.703125	0.003570	3.552448
10	(hoe, got)	(nigga)	0.006294	0.133566	0.005594	0.888889	6.655032	0.004754	7.797902
1	(hoe, get)	(nigga)	0.004895	0.133566	0.004196	0.857143	6.417352	0.003542	6.065035
3	(pussy, shit)	(nigga)	0.005594	0.133566	0.004196	0.750000	5.615183	0.003449	3.465734
0	(hoe, fuck)	(nigga)	0.006993	0.133566	0.004895	0.700000	5.240838	0.003961	2.888112
7	(nigga, ass, u)	(bitch)	0.004895	0.167133	0.004196	0.857143	5.128512	0.003378	5.830070
6	(nigga, u)	(bitch)	0.011888	0.167133	0.008392	0.705882	4.223480	0.006405	2.831748

In [78]:
```
te_ary = te.fit_transform(list(map(nltk.word_tokenize,df[df['class'] == "Offensive Language"]
df_enc = pd.DataFrame(te_ary, columns=te.columns_)
df_apri = fpgrowth(df_enc, min_support=0.00235, use_colnames=True)
len(df_apri)
```

Out[78]: 943

In [79]:
```
rules = association_rules(df_apri, metric="confidence", min_threshold=0.4)
sorted_rules = rules[rules.lift > 1.2].sort_values(by="confidence", ascending=False)
sorted_rules.head(10)
```

Out[79]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
247	(karma)	(bitch)	0.002553	0.546274	0.002501	0.979592	1.793224	0.001106	22.232569
1	(bad, like)	(bitch)	0.004429	0.546274	0.003908	0.882353	1.615220	0.001489	3.856670
2	(got, bad)	(bitch)	0.002866	0.546274	0.002449	0.854545	1.564316	0.000884	3.119366
0	(bad)	(bitch)	0.026159	0.546274	0.022303	0.852590	1.560736	0.008013	3.077978
200	(basic)	(bitch)	0.004221	0.546274	0.003596	0.851852	1.559385	0.001290	3.062650
92	(son)	(bitch)	0.005211	0.546274	0.004117	0.790000	1.446160	0.001270	2.160600
32	(hate, like)	(bitch)	0.004169	0.546274	0.003283	0.787500	1.441584	0.001006	2.135181
159	(lil)	(bitch)	0.012142	0.546274	0.009484	0.781116	1.429897	0.002851	2.072905
144	(nigga, yo)	(bitch)	0.003439	0.546274	0.002606	0.757576	1.386805	0.000727	1.871619
172	(little)	(bitch)	0.013705	0.546274	0.010318	0.752852	1.378157	0.002831	1.835845

In [77]:
```
sorted_rules[sorted_rules["consequents"].apply(lambda x: 'like' not in x and ('bitch' not in
```

Out[77]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
165	(loyal)	(hoe)	0.006826	0.202814	0.005107	0.748092	3.688561	0.003722	3.164587
156	(eating)	(pussy)	0.004013	0.109849	0.002762	0.688312	6.265987	0.002321	2.855902
230	(eat)	(pussy)	0.011516	0.109849	0.006566	0.570136	5.190183	0.005301	2.070773
74	(suck)	(dick)	0.005680	0.016832	0.002606	0.458716	27.253103	0.002510	1.816362

Conclusion

We used a SGDClassifier along with a GridSearch to optimize the classification of Twitter tweets between three human labeled possibilities: hatespeech, offensive language, and neither.

We put in most effort into two classifiers: SGD and CNN. Even with all the complex additions and embeddings to CNN, it didn't overperform the SGD classifier in terms of hate speech precision.

In future research opportunities, if the data could be classified under more categories of hatespeech along with the crowdsourced labelers being more attentive, it could make determining hatespeech more pinpointed. People will always get around an automatic filter no matter how complex it is, since humans can manipulate language much better than any computer can comprehend. Classfiers like these would then only be used as an aid to scour the vast amounts of new text from users for humans to verify.

With our SGDClassifier we built, we have a f1_score of 67.4% and a 56.5% precision score for hate speech. And during our modeling process when we combined hate speech and offensive language we have found a f1_score of 91.4% and a 97.1% precision score for offensive with 85.6% for unoffensive.