In [1]:
```
import pandas as pd
import numpy as np
from scipy.sparse import hstack

# plots
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# processing
from sklearn.preprocessing import StandardScaler
from sklearn.compose import make_column_transformer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.pipeline import make_pipeline, Pipeline
from sklearn.model_selection import cross_val_score, GridSearchCV
from sklearn.feature_extraction.text import TfidfVectorizer, TfidfTransformer
from nltk import word_tokenize
from nltk.stem import WordNetLemmatizer

# modeling
from sklearn.linear_model import LogisticRegressionCV, LogisticRegression

# others
import warnings
warnings.filterwarnings('ignore')

# nlp
import string
from gensim import models
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
```

```
paramiko missing, opening SSH/SCP/SFTP paths will be disabled.  `pip install
paramiko` to suppress
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\costa\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\costa\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\costa\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[1]: True

Homework 4

In [2]:
```
df = pd.read_csv('data/reddit_200k_train.csv', encoding = 'latin-1', index_col
='Unnamed: 0')
test = pd.read_csv('data/reddit_200k_test.csv', encoding = 'latin-1', index_co
l='Unnamed: 0')

# subset the columns
df['removed'] = df.REMOVED
df = df[['body', 'removed']]

test['removed'] = test.REMOVED
test = test[['body', 'removed']]
```

In [3]:
```
df.head(2)
```

Out[3]:

	body	removed
1	I've always been taught it emerged from the ea...	False
2	As an ECE, my first feeling as "HEY THAT'S NOT...	True

Task 1: Bag of Words and Simple Features

1.1 Baseline Model

A simple logistic regression after vectorizing our text.

In [4]:
```
cv = CountVectorizer() # initializing the vectorizer
# fitting and transforming our text data - both train and test
X_train_base = cv.fit_transform(df.body)
X_test_base = cv.transform(test.body)

# mapping our targets
y_train = np.where(df.removed, 1, 0)
y_test = np.where(test.removed, 1, 0)
```

In [5]:
```
# training the model
lr = LogisticRegressionCV(cv=5, scoring='roc_auc', solver='sag', random_state=
42).fit(X_train_base, y_train)
baseline_train_score = lr.score(X_train_base, y_train) # score on training set
baseline_test_score = lr.score(X_test_base, y_test)    # score on test set
```

We check the performance on both the training and test set.

In [6]:
```
print('BASELINE MODEL MEAN PERFORMANCE (ROC-AUC):')
print('Training: {}'.format(np.round(np.mean(baseline_train_score), 2)))
print('Test: {}'.format(np.round(np.mean(baseline_test_score), 2)))
```

```
BASELINE MODEL MEAN PERFORMANCE (ROC-AUC):
Training: 0.75
Test: 0.75
```

In [7]:
```
bot10 = np.array(cv.get_feature_names())[np.argsort(lr.coef_[0])[:10]]; print(
'Bottom 10: {}'.format(bot10))
top10 = np.array(cv.get_feature_names())[np.argsort(lr.coef_[0])[::-1][:10]];
print('Top 10: {}'.format(top10))
```

```
Bottom 10: ['iâ' 'itâ' 'donâ' 'http' 'edit' 'does' 'www' 'link' 'org' 'com']
Top 10: ['fuck' 'comments' 'removed' 'shit' 'women' 'oh' 'weed' 'my' 'commen
t'
 'let']
```

1.2 Processing

1.2.1 Using lemmatization

We want to try using lemmatization with the count vectorizer, which will help reduce the number of features.

In [8]:
```
# initializing the vectorizer w/ lemmatization
class LemmaTokenizer(object):
    def __init__(self):
        self.wnl = WordNetLemmatizer()
    def __call__(self, doc):
        return [self.wnl.lemmatize(t) for t in word_tokenize(doc)]
lem = CountVectorizer(tokenizer = LemmaTokenizer())

# transforming the text data with our new vectorizer
X_train_lem = lem.fit_transform(df.body) # training
X_test_lem = lem.transform(test.body)    # test
```

In [9]:
```
# training the model
lr = LogisticRegressionCV(cv=5, scoring='roc_auc', solver='sag', random_state=
42).fit(X_train_lem, y_train)

# get the scores
lem_train_score = lr.score(X_train_lem, y_train)  # training set
lem_test_score = lr.score(X_test_lem, y_test)     # test set
```

Check the performance:

In [10]:
```
print('LEMMATIZATION MODEL MEAN PERFORMANCE (ROC-AUC):')
print('Training: {}'.format(np.round(np.mean(lem_train_score), 2)))
print('Test: {}'.format(np.round(np.mean(lem_test_score), 2)))
```

```
LEMMATIZATION MODEL MEAN PERFORMANCE (ROC-AUC):
Training: 0.72
Test: 0.72
```

In [11]:
```
bot10 = np.array(lem.get_feature_names())[np.argsort(lr.coef_[0])[:10]]; print
('Bottom 10: {}'.format(bot10))
top10 = np.array(lem.get_feature_names())[np.argsort(lr.coef_[0])[::-1][:10]];
print('Top 10: {}'.format(top10))
```

```
Bottom 10: ['?' ':' 'http' 'how' 'would' 'what' 'itâ\x80\x99s' 'there' 'in'
'doe']
Top 10: ['my' '!' 'comment' 'me' 'it\x92s' 'woman' '...' 'removed' 'fuck'
'>']
```

Lemmatization makes our train and test scores worse, and doesn't seem to really be working given the names of the features here.

1.2.2 Using tf-idf scaling w/ GS

In [12]:
```
tfidf = TfidfVectorizer()                                               # tfidf v
ectorizer
lr = LogisticRegression(solver='sag', random_state=42)                  # model
tfidf_pipeline = Pipeline([('preprocessing', tfidf), ('lr', lr)])       # pipelin
e - vectorizer and model
param_grid = {'lr__C': np.logspace(-3, 2, 6)}                           # param g
rid

# grid search
gs = GridSearchCV(tfidf_pipeline, param_grid, cv=5, scoring='roc_auc').fit(df.
body, y_train)
```

In [13]:
```
tfidf_train_score = gs.score(df.body, y_train) # training data
tfidf_test_score = gs.score(test.body, y_test) # test data
```

In [14]:
```
print('TFIDF MODEL MEAN PERFORMANCE (ROC-AUC):')
print('Training: {}'.format(np.round(np.mean(tfidf_train_score), 2)))
print('Test: {}'.format(np.round(np.mean(tfidf_test_score), 2)))
```

```
TFIDF MODEL MEAN PERFORMANCE (ROC-AUC):
Training: 0.84
Test: 0.78
```

In [15]:
```
bot10 = np.array(gs.best_estimator_.steps[0][1].get_feature_names())[np.argsor
t(gs.best_estimator_.steps[1][1].coef_[0])[:10]]
print('Bottom 10: {}'.format(bot10))
top10 = np.array(gs.best_estimator_.steps[0][1].get_feature_names())[np.argsor
t(gs.best_estimator_.steps[1][1].coef_[0])[::-1][:10]]
print('Top 10: {}'.format(top10))
```

```
Bottom 10: ['iâ' 'itâ' 'donâ' 'edit' '½ï' 'thatâ' 'doesnâ' 'youâ' 'didnâ' 'is
nâ']
Top 10: ['0001f914' '0001f602' 'fuck' 'mods' 'upvote' 'upvoted' 'censorship'
 'flair' 'fe0f' 'saffron']
```

Tf-idf scaling gives us results that are slightly better than our baseline model. The features are also now much more interesting, especially the 30 features with the highest positive coefficients. Indeed, we find many words related to very sensitive subjects ('feminists', 'liberals', 'hillary', 'trump'), as well as curse words. Moreover, '0001f602' and '0001f914' actually correspond to emojis (laughing crying emoji and thinking emoji respectively).

1.2.3 Using both lemmatization and tf-idf scaling w/ GS

In [17]:
```
tlem = TfidfVectorizer(tokenizer = LemmaTokenizer())                    # tfidf v
ectorizer w/ lemma
lr = LogisticRegression(solver='sag', random_state=42)                  # model
tlem_pipeline = Pipeline([('preprocessing', tlem), ('lr', lr)])         # pipelin
e - vectorizer and model
param_grid = {'lr__C': np.logspace(-3, 2, 6)}                           # param g
rid

# grid search
gs = GridSearchCV(tlem_pipeline, param_grid, cv=5, scoring='roc_auc', n_jobs=2
).fit(df.body, y_train)
```

In [18]:
```
tlem_train_score = gs.score(df.body, y_train) # training data
tlem_test_score = gs.score(test.body, y_test) # test data
```

In [19]:
```
print('TFIDF w/ LEMMA MODEL PERFORMANCE (ROC-AUC):')
print('Training: {}'.format(np.round(np.mean(tlem_train_score), 2)))
print('Test: {}'.format(np.round(np.mean(tlem_test_score), 2)))
```

```
TFIDF w/ LEMMA MODEL PERFORMANCE (ROC-AUC):
Training: 0.84
Test: 0.79
```

In [20]:
```
bot10 = np.array(gs.best_estimator_.steps[0][1].get_feature_names())[np.argsor
t(gs.best_estimator_.steps[1][1].coef_[0])[:10]]
print('Bottom 10: {}'.format(bot10))
top10 = np.array(gs.best_estimator_.steps[0][1].get_feature_names())[np.argsor
t(gs.best_estimator_.steps[1][1].coef_[0])[::-1][:10]]
print('Top 10: {}'.format(top10))
```

```
Bottom 10: ['itâ\x80\x99s' 'donâ\x80\x99t' 'iâ\x80\x99m' 'edit' 'http'
 'thatâ\x80\x99s' 'ï¿½ï¿½' 'doesnâ\x80\x99t' 'iâ\x80\x99ve'
 'didnâ\x80\x99t']
Top 10: ['it\x92s' 'don\x92t' 'i\x92m' 'that\x92s' 'can\x92t' 'i\x92ve'
 'doesn\x92t' 'didn\x92t' '>' '<']
```

Here, we can see that while lemmatization made the baseline model worse, it actually makes tf-idf scaling better. However, the features are not really interpretable here and they seem to consist mostly of stop words.

1.2.4 Using bi-grams, tri-grams and 4-grams

In [21]:
```
stopwords = stopwords.words('english')
for w in ['no', 'not', 'how', 'why', 'himself', 'yourself', 'you', 'me']:
    stopwords.remove(w)
```

In [22]:
```
gram = CountVectorizer(ngram_range=(2, 4), min_df=5, stop_words=stopwords)

X_train_chng = gram.fit_transform(df.body)
X_test_chng = gram.transform(test.body)
```

In [23]:
```
lr = LogisticRegressionCV(cv=5, scoring='roc_auc', solver='sag', n_jobs=2, ran
dom_state=42).fit(X_train_chng, y_train)
```

In [24]:
```
chng_train_score = lr.score(X_train_chng, y_train)
chng_test_score = lr.score(X_test_chng, y_test)
```

In [25]:
```
print('Model with n-grams achieves a mean of {} ROC-AUC on our training data.'
.format(np.round(np.mean(chng_train_score), 2)))
print('Model with n-grams achieves a mean of {} ROC-AUC on our test data.'.for
mat(np.round(np.mean(chng_test_score), 2)))
```

```
Model with n-grams achieves a mean of 0.83 ROC-AUC on our training data.
Model with n-grams achieves a mean of 0.71 ROC-AUC on our test data.
```

In [26]:
```
bot10 = np.array(gram.get_feature_names())[np.argsort(lr.coef_[0])[:10]]; prin
t(bot10)
top10 = np.array(gram.get_feature_names())[np.argsort(lr.coef_[0])[::-1][:10
]]; print(top10)
```

```
['itâ not' 'thanks ama' 'iâ not' 'donâ know' 'donâ think' 'you donâ'
 'anyone know' 'abstract gt' 'link paper' 'you recommend']
['comment section' 'comments removed' 'fuck you' 'fat shaming'
 'big pharma' 'comments deleted' 'social justice' 'happened comments'
 'affirmative action' 'commit suicide']
```

As we can see, the test score of n-grams is not so good, despite its training score being pretty high. However, we some other interesting features: some, such as "comments removed", may actually indicate a leak in the data. Others, like "rick morty", are... interesting!

Using all of it: lemmatization, tf-idf scaling, n-grams w/ GS

In [27]:
```
allv = CountVectorizer(ngram_range=(2, 4), min_df=5, stop_words='english', tok
enizer=LemmaTokenizer()) # tfidf vectorizer w/ lemma
lr = LogisticRegression(solver='sag', random_state=42)
# model
allv_pipeline = Pipeline([('preprocessing', allv), ('lr', lr)])
# pipeline - vectorizer and model
param_grid = {'lr__C': np.logspace(-3, 2, 6)}
# param grid

# grid search
gs = GridSearchCV(allv_pipeline, param_grid, cv=5, scoring='roc_auc', n_jobs=2
).fit(df.body, y_train)
```

In [28]:
```
allv_train_score = gs.score(df.body, y_train) # training data
allv_test_score = gs.score(test.body, y_test) # test data
```

In [29]:
```
print('TFIDF w/ LEMMA MODEL PERFORMANCE (ROC-AUC):')
print('Training: {}'.format(np.round(np.mean(allv_train_score), 2)))
print('Test: {}'.format(np.round(np.mean(allv_test_score), 2)))
```

```
TFIDF w/ LEMMA MODEL PERFORMANCE (ROC-AUC):
Training: 0.65
Test: 0.65
```

In [30]:
```
bot10 = np.array(gs.best_estimator_.steps[0][1].get_feature_names())[np.argsor
t(gs.best_estimator_.steps[1][1].coef_[0])[:10]]
print('Bottom 10: {}'.format(bot10))
top10 = np.array(gs.best_estimator_.steps[0][1].get_feature_names())[np.argsor
t(gs.best_estimator_.steps[1][1].coef_[0])[::-1][:10]]
print('Top 10: {}'.format(top10))
```

```
Bottom 10: ['http :' '& gt ;' '& gt' 'gt ;' ') .' '] (' '( http' '( http :'
 '] ( http' '] ( http :']
Top 10: ['> <' '! !' 'comment removed' '. it\x92s' 'flair post' 'removed ?'
 '! ! !' '. i\x92m' 'comment removed ?' ', it\x92s']
```

Combining everything actually seems to yield the worst scores so far, which is somewhat surprising. The Lemma Tokenizer probably doesn't really work as we'd intend it to.

1.3 Other features

We'll engineer the following features:

- Length: document size (# of characters)
- Capitalization: percentage of capital characters
- Punctuations: boolean indicating whether the post contained punctuations or not

In [31]:
```
df.head(2)
```

Out[31]:

	body	removed
1	I've always been taught it emerged from the ea...	False
2	As an ECE, my first feeling as "HEY THAT'S NOT...	True

Length:

In [32]:
```
df['length'] = df.body.str.len()
test['length'] = test.body.str.len()
```

Upper Case Characters:

In [33]:
```
df['all_cap'] = np.where(df.body.str.isupper(), 1, 0)
test['all_cap'] = np.where(test.body.str.isupper(), 1, 0)
```

Punctuations:

In [34]:
```
df['punctuation'] = np.where(df.body.str.contains('!'), 1, 0)
test['punctuation'] = np.where(test.body.str.contains('!'), 1, 0)
```

Scaling:

In [35]:
```
scaler = StandardScaler().fit((df['length']).values.reshape(-1,1))
df['length'] = scaler.transform(df['length'].values.reshape(-1,1))
test['length'] = scaler.transform(test['length'].values.reshape(-1,1))
```

In [36]:
```
tlem = TfidfVectorizer(tokenizer = LemmaTokenizer())  # vectorizer w/ lemma

X_tlem = tlem.fit_transform(df.body)                  # transform train
X_test_tlem = tlem.transform(test.body)               # transform test

# combining the text data with the other features
X_tlem = hstack((X_tlem, df[['length', 'all_cap', 'punctuation']].values))
X_test_tlem = hstack((X_test_tlem, test[['length', 'all_cap', 'punctuation']].
values))

# training the model
lr = LogisticRegressionCV(cv=5, scoring='roc_auc', solver='sag', n_jobs=2, ran
dom_state=42).fit(X_tlem, y_train)
tlem_score = lr.score(X_tlem, y_train)
tlem_test_score = lr.score(X_test_tlem, y_test)
```

In [37]:
```
print('TFIDF w/ LEMMA and EXTRA FEATURES PERFORMANCE (ROC-AUC):')
print('Training: {}'.format(np.round(np.mean(tlem_score), 2)))
print('Test: {}'.format(np.round(np.mean(tlem_test_score), 2)))
```

```
TFIDF w/ LEMMA and EXTRA FEATURES PERFORMANCE (ROC-AUC):
Training: 0.83
Test: 0.79
```

Task 2: Word Vectors

In [38]:
```
w = models.KeyedVectors.load_word2vec_format('V:/word_vectors/GoogleNews-vecto
rs-negative300.bin', binary=True)
```

Vectorizing our text body and the test set.

In [39]:
```
vect_w2v = CountVectorizer(vocabulary=w.index2word)
vect_w2v.fit(df.body)

docs = vect_w2v.inverse_transform(vect_w2v.transform(df.body))
X_train_body = []
for doc in docs:
    if len(doc) > 0:
        X_train_body.append(np.mean(w[doc], axis=0))
    else:
        X_train_body.append(np.zeros(300))
X_train_body = np.vstack(X_train_body)
```

In [40]:
```
# repeating the above for the test set
docs_test = vect_w2v.inverse_transform(vect_w2v.transform(test.body))
X_test_body = []
for doc in docs_test:
    if len(doc) > 0:
        X_test_body.append(np.mean(w[doc], axis=0))
    else:
        X_test_body.append(np.zeros(300))
X_test_body = np.vstack(X_test_body)
```

Testing the model.

In [41]:
```
lr = LogisticRegressionCV(cv=5, scoring='roc_auc', solver='sag', n_jobs=2, ran
dom_state=42).fit(X_train_body, y_train)

w2v_train_score = lr.score(X_train_body, y_train)
w2v_test_score = lr.score(X_test_body, y_test)
```

In [42]:
```
print('Model w/ W2V achieves a mean of {} ROC-AUC on our training data.'.forma
t(np.round(np.mean(w2v_train_score), 2)))
print('Model w/ W2V achieves a mean of {} ROC-AUC on our test data.'.format(np
.round(np.mean(w2v_test_score), 2)))
```

```
Model w/ W2V achieves a mean of 0.73 ROC-AUC on our training data.
Model w/ W2V achieves a mean of 0.73 ROC-AUC on our test data.
```

What if we incorporate the other features? Including one that indicates that there were no vocab words.

In [43]:
```
docs_series = pd.Series(docs)
df['v_length'] = docs_series.apply(lambda x: len(x)) # finds the document leng
th
df['v_empty'] = np.where(df.v_length == 0.0, 1, 0)   # maps empty docs to 1 an
d others to 0

# repeat the above for test
docs_series = pd.Series(docs_test)
test['v_length'] = docs_series.apply(lambda x: len(x)) # finds the document le
ngth
test['v_empty'] = np.where(test.v_length == 0.0, 1, 0)   # maps empty docs to
 1 and others to 0
```

In [44]:
```
X_train_body2 = np.concatenate((X_train_body, df[['length', 'all_cap', 'v_empt
y', 'punctuation']].values), axis=1)
X_test_body2 = np.concatenate((X_test_body, test[['length', 'all_cap', 'v_empt
y', 'punctuation']].values), axis=1)
```

In [45]:
```
lr = LogisticRegressionCV(cv=5, scoring='roc_auc', solver='sag', n_jobs=2, ran
dom_state=42).fit(X_train_body2, y_train)

w2v_train_score2 = lr.score(X_train_body2, y_train)
w2v_test_score2 = lr.score(X_test_body2, y_test)
```

In [46]:
```
print('Second model w/ W2V achieves a mean of {} ROC-AUC on our training dat
a.'.format(np.round(np.mean(w2v_train_score), 2)))
print('Second model w/ W2V achieves a mean of {} ROC-AUC on our test data.'.fo
rmat(np.round(np.mean(w2v_test_score), 2)))
```

```
Second model w/ W2V achieves a mean of 0.73 ROC-AUC on our training data.
Second model w/ W2V achieves a mean of 0.73 ROC-AUC on our test data.
```

In [ ]: