SENTIMENT ANALYSIS

Sentiment analysis is used to analyse what was said about a topic. Is the comment positive or negative? In this example we are using a data set of movie reviews. We need to classify each movie to positive/negative based on text given.

FEATURE EXTRACTION

The process of converting our text document to numerical form. I have used 3 types of vectorizers here. Vectorizers are evaluated based on the accuracies.

TF-IDF VECTORIZER

TF-IDF (Term frequency- Inverse Document Frequency) vectorizer considers the frequency of a word in a document and frequency between documents.

```
# training: tf-idf + logistic regression
# you should explore different representations and algorithms.
from sklearn.feature_extraction.text import TfidfVectorizer
max_feature_num = 1000
train_vectorizer = TfidfVectorizer(max_features=max_feature_num)
train_vecs = train_vectorizer.fit_transform(train_text)
test_vecs = TfidfVectorizer(max_features=max_feature_num,vocabulary=train_vectorizer.vocabulary_).fit_transform(test_text)

# train model
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(max_iter = 5000).fit(train_vecs, train_labels)

# test model
test_pred = clf.predict(test_vecs)
from sklearn.metrics import precision_recall_fscore_support,accuracy_score
tfidf_acc = accuracy_score(test_labels, test_pred)
pre, rec, f1, _ = precision_recall_fscore_support(test_labels, test_pred, average='macro')
print('acc', tfidf_acc)
print('precision', pre)
print('rec', rec)
print('f1', f1)
```

```
acc 0.8616
precision 0.8616043455692743
rec 0.8615913854621674
f1 0.8615956596398864
```

COUNT VECTORIZER

Count vectorizer considers every word in the document and converts to features.

```
from sklearn.feature_extraction.text import CountVectorizer
max_feature_num = 1000
count_train_vectorizer = CountVectorizer(max_features=max_feature_num)
train_vecs = count_train_vectorizer.fit_transform(train_text)
test_vecs = CountVectorizer(max_features=max_feature_num,vocabulary=train_vectorizer.vocabulary_).fit_transform(test_text)

# train model
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(max_iter = 5000).fit(train_vecs, train_labels)

# test model
test_pred = clf.predict(test_vecs)
from sklearn.metrics import precision_recall_fscore_support,accuracy_score
count_acc = accuracy_score(test_labels, test_pred)
pre, rec, f1, _ = precision_recall_fscore_support(test_labels, test_pred, average='macro')
print('acc', count_acc)
print('precision', pre)
print('rec', rec)
print('f1', f1)
```

```
acc 0.8648
precision 0.8648402808583475
rec 0.8647786364581833
f1 0.8647895293011492
```

HASHING VECTORIZER

Converts a document a collection to matrix of word occurrences

```
from sklearn.feature_extraction.text import HashingVectorizer
max_feature_num = 1000
hash_train_vectorizer = HashingVectorizer(n_features=max_feature_num)
train_vecs = hash_train_vectorizer.fit_transform(train_text)
test_vecs = HashingVectorizer(n_features=max_feature_num).fit_transform(test_text)

# train model
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(max_iter = 5000).fit(train_vecs, train_labels)

# test model
test_pred = clf.predict(test_vecs)
from sklearn.metrics import precision_recall_fscore_support,accuracy_score
hash_acc = accuracy_score(test_labels, test_pred)
pre, rec, f1, _ = precision_recall_fscore_support(test_labels, test_pred, average='macro')
print('acc', hash_acc)
print('precision', pre)
print('rec', rec)
print('f1', f1)
```

```
acc 0.82
precision 0.820026833196351
rec 0.8199795196723147
f1 0.8199872983037684
```

COMPARING THE RESULTS OF VECORIZERS

In [31]:

```
accuracy_df = pd.DataFrame()
accuracy_df['representation']= ['tfidf_vec','count_vec','hash_vec']
accuracy_df['accuracy']= [tfidf_acc,count_acc,hash_acc]
accuracy_df

```

Out[31]:

	representation	accuracy
0	tfidf_vec	0.8616
1	count_vec	0.8648
2	hash_vec	0.8200

Tfidf Vector and count vectorizer produced almost same result

We will be considering both count vectorizer and tfidf vectorizer for our next step.

Frequency distribution of words

TUNING HYPER PARAMETERS OF THE VECTORIZER

The frequency distribution of words in the data set is as follows

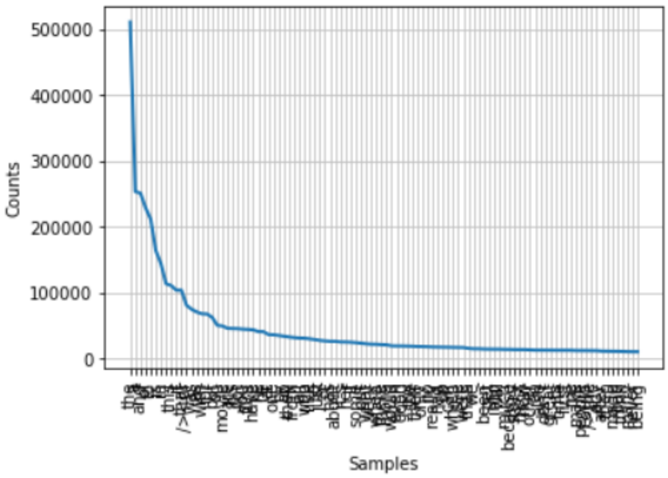
```
In [15]:
# frequency distribution of dataset
#making corpus containign all words of the dataset
import re
corpus = []
for sentence in all_text:
    for word in sentence.split():
        corpus.append(word.lower())
```

The above lines of code creates a list of all the unique words in the dataset.

```
In [16]:
#getting the frequency distribution of words in corpus
import nltk
freqdist = nltk.FreqDist(corpus)
print(freqdist.most_common(10))
```

```
[('the', 510245), ('a', 253107), ('and', 250840), ('of', 228709), ('to', 211262), ('is', 163617), ('in', 143713), ('i', 11331
8), ('this', 110472), ('it', 103506)]
```

```
freqdist.plot(100)
```




```
<matplotlib.axes._subplots.AxesSubplot at 0x2221d369b88>
```

Inorder to find out the best parameters for the vectorizer we use the GridSerachCV and Pipeline tool in sklearn.

Choosing Ngrams

```python
from nltk.corpus import stopwords
#nltk.download('stopwords')
from nltk.stem import PorterStemmer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression

import re
def text_preprocess(text):
    replace_punct = re.compile("[,\"()\[\].;:!\'?]") #punctuation removal
    replace_tag = re.compile("(\-)|(\/)|(<br\s*\/>+)") #tags removal
    text = [replace_punct.sub("", sent.lower()) for sent in text]
    text = [replace_tag.sub(" ", line) for sent in text]
    return text
def tokenize(all_text):
    return all_text.split()
def norm_tokenize_stem(all_text):
    porter = PorterStemmer()
    for sent in all_text:
        final_text = [porter.stem(w) for w in sent.split()]
    return final_text
stop_words = stopwords.words('english')

vectorizer = TfidfVectorizer(strip_accents=None, lowercase=True)
clf = LogisticRegression(random_state=0,max_iter = 5000)
param_grid = [{'tfidf__ngram_range': [(1, 1),(1,2)],
               'tfidf__stop_words': [stop_words,None],
               'tfidf__tokenizer': [None,tokenize,norm_tokenize_stem],
               'tfidf__preprocessor': [None,text_preprocess],
               'tfidf__use_idf':[True,False],
               'tfidf__max_features':[5000],
               'tfidf__norm':['l2'],
               'clf__C': [0.1,1.0,10.0]}]

pipe_tfidf_lr = Pipeline([('tfidf', vectorizer),
                          ('clf', clf )])
grid_tfidf_lr = GridSearchCV(pipe_tfidf_lr, param_grid, scoring='accuracy', cv=5, verbose=1, n_jobs=-1)
```

In [5]:

```python
grid_tfidf_lr.fit(train_text,train_labels)
```

```
Fitting 5 folds for each of 144 candidates, totalling 720 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  2.7min
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed: 14.0min
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed: 28.8min
[Parallel(n_jobs=-1)]: Done 720 out of 720 | elapsed: 43.4min finished
```

Out[5]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('tfidf',
                                        TfidfVectorizer(analyzer='word',
                                                        binary=False,
                                                        decode_error='strict',
                                                        dtype=<class 'numpy.float64'>,
                                                        encoding='utf-8',
                                                        input='content',
                                                        lowercase=True,
                                                        max_df=1.0,
                                                        max_features=None,
                                                        min_df=1,
                                                        ngram_range=(1, 1),
                                                        norm='l2',
                                                        preprocessor=None,
                                                        smooth_idf=True,
                                                        stop_words=None,
                                                        strip_acc...
                                                'yourselves', 'he', 'him',
                                                'his', 'himself', 'she',
                                                "she's", 'her', 'hers',
                                                'herself', 'it', "it's", 'its',
                                                'itself', ...],
                                               None],
                          'tfidf__tokenizer': [None,
                                               <function tokenize at 0x0000021AF33DC678>,
                                               <function norm_tokenize_stem at 0x0000021AF33E2168>],
```

In [6]:

```python
print('Best parameters are: ' + str(grid_tfidf_lr.best_params_))
print('Best accuracy is : %.4f' % grid_tfidf_lr.best_score_)
```

```
Best parameters are: {'clf__C': 1.0, 'tfidf__max_features': 5000, 'tfidf__ngram_range': (1, 2), 'tfidf__norm': 'l2', 'tfidf__pr
eprocessor': None, 'tfidf__stop_words': ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you'v
e", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers',
'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'thi
s', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'havin
g', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'b
y', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'fro
m', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'wher
e', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'ow
n', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'l
l', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "ha
dn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "need
n't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"], 't
fidf__tokenizer': None, 'tfidf__use_idf': True}
Best accuracy is : 0.8845
```

1.The preprocess I used here will remove the hashtags and punctuations

2.Tokenizer used will split the word with space

3. Normalization used is Porter stemming.

The above results shows that:

1. Best value for C(logistic regression parameter) = 1.0
2. Best ngram_range = (1,2)
3. Preprocessing have no effect on getting best accuracy
4. Set of stopwords which improves the accuracy is `['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]`
5. Tokenization have no effect on improving accuracy

   In the following steps, we will be using the best parameters that we obtained from previous result for evaluating our vectorizers.

   Apply the best parameters in the TF-IDF vectorizer and Count Vectorizer:

TF-IDF vectorizer

In [16]:

```
# training: tf-idf + logistic regression
# you should explore different representations and algorithms.
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
max_feature_num = 5000
n_range = (1,2)
preprocess = None
stp_words = ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'you
nm = 'l2'
smoothIDF = True
token = None
useIDF = True
train_vectorizer = TfidfVectorizer(preprocessor=preprocess, tokenizer=token, stop_words=stp_words,ngram_range=n_range, max_f
train_vecs = train_vectorizer.fit_transform(train_text)
test_vecs = TfidfVectorizer(preprocessor=preprocess, tokenizer=token, stop_words=stp_words,ngram_range=n_range, max_features

# train model
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(C=1.0,max_iter = 5000).fit(train_vecs, train_labels)

# test model
test_pred = clf.predict(test_vecs)
from sklearn.metrics import precision_recall_fscore_support,accuracy_score
custom_tfidf_acc = accuracy_score(test_labels, test_pred)
pre, rec, f1, _ = precision_recall_fscore_support(test_labels, test_pred, average='macro')
print('acc', custom_tfidf_acc)
print('precision', pre)
print('rec', rec)
print('f1', f1)
```

```
acc 0.8868125
precision 0.8869615346072897
rec 0.8868125
f1 0.8868016006744337
```

Count vectorizer

In [17]:

```
# training: tf-idf + logistic regression
# you should explore different representations and algorithms.
from sklearn.feature_extraction.text import CountVectorizer
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
max_feature_num = 5000
n_range = (1,2)
preprocess = None
stp_words = ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'you
token = None
train_vectorizer = CountVectorizer(preprocessor=preprocess, tokenizer=token, stop_words=stp_words,ngram_range=n_range, max_f
test_vecs = CountVectorizer(preprocessor=preprocess, tokenizer=token, stop_words=stp_words,ngram_range=n_range, max_features

# train model
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(C=1.0,max_iter = 5000).fit(train_vecs, train_labels)

# test model
test_pred = clf.predict(test_vecs)
from sklearn.metrics import precision_recall_fscore_support,accuracy_score
custom_count_acc = accuracy_score(test_labels, test_pred)
pre, rec, f1, _ = precision_recall_fscore_support(test_labels, test_pred, average='macro')
print('acc', custom_count_acc)
print('precision', pre)
print('rec', rec)
print('f1', f1)
```

```
acc 0.6410625
precision 0.6503367982235213
rec 0.6410625
f1 0.6354400615324234
```

:

```
accuracy_df = pd.DataFrame()
accuracy_df['representation']= ['tfidf_vec','count_vec']
accuracy_df['accuracy']= [custom_tfidf_acc,custom_count_acc]
accuracy_df

```

:

	representation	accuracy
0	tfidf_vec	0.886813
1	count_vec	0.641062

The best params from grid search increased the accuracy of our sentiment classifier. But the accuracy of the count vectorizer has decreased.

Choosing maximum feature numbers

Trying different values for max_feature_num:

```
from nltk.corpus import stopwords
#nltk.download('stopwords')
from nltk.stem import PorterStemmer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression

import re
def text_preprocess(text):
    replace_punct = re.compile("[,\"()\[\].;:!\'?]") #punctuation removal
    replace_tag = re.compile("(\-)|(\/)|(<br\s*\/>+)") #tags removal
    text = [replace_punct.sub("", sent.lower()) for sent in text]
    text = [replace_tag.sub(" ", line) for sent in text]
    return text
def tokenize(all_text):
    return all_text.split()
def norm_tokenize_stem(all_text):
    porter = PorterStemmer()
    for sent in all_text:
        final_text = [porter.stem(w) for w in sent.split()]
    return final_text
stop_words = stopwords.words('english')

vectorizer = TfidfVectorizer(strip_accents=None, lowercase=True)
clf = LogisticRegression(random_state=0,max_iter = 5000)
param_grid = [{'tfidf__ngram_range': [(1,2)],
               'tfidf__stop_words': [stop_words,None],
               'tfidf__tokenizer': [None],
               'tfidf__preprocessor': [None],
               'tfidf__use_idf':[True],
               'tfidf__max_features':[5000,7000,9000,10000,12000,15000],
               'tfidf__norm':['l2'],
               'tfidf__smooth_idf':[True],
               'clf__C': [1.0]}]

pipe_tfidf_lr = Pipeline([('tfidf', vectorizer),
                          ('clf', clf )])
grid_tfidf_lr = GridSearchCV(pipe_tfidf_lr, param_grid, scoring='accuracy', cv=5, verbose=1, n_jobs=-1)
```


```
from nltk.corpus import stopwords
#nltk.download('stopwords')
from nltk.stem import PorterStemmer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression

import re
def text_preprocess(text):
    replace_punct = re.compile("[,\"()\[\].;:!\'?]") #punctuation removal
    replace_tag = re.compile("(\-)|(\/)|(<br\s*\/>+)") #tags removal
    text = [replace_punct.sub("", sent.lower()) for sent in text]
    text = [replace_tag.sub(" ", line) for sent in text]
    return text
def tokenize(all_text):
    return all_text.split()
def norm_tokenize_stem(all_text):
    porter = PorterStemmer()
    for sent in all_text:
        final_text = [porter.stem(w) for w in sent.split()]
    return final_text
stop_words = stopwords.words('english')

vectorizer = TfidfVectorizer(strip_accents=None, lowercase=True)
clf = LogisticRegression(random_state=0,max_iter = 5000)
param_grid = [{'tfidf__ngram_range': [(1,2)],
               'tfidf__stop_words': [stop_words,None],
               'tfidf__tokenizer': [None],
               'tfidf__preprocessor': [None],
               'tfidf__use_idf':[True],
               'tfidf__max_features':[5000,7000,9000,10000,12000,15000],
               'tfidf__norm':['l2'],
               'tfidf__smooth_idf':[True],
               'clf__C': [1.0]}]

pipe_tfidf_lr = Pipeline([('tfidf', vectorizer),
                          ('clf', clf )])
grid_tfidf_lr = GridSearchCV(pipe_tfidf_lr, param_grid, scoring='accuracy', cv=5, verbose=1, n_jobs=-1)
```


```
In [96]:
print('Best parameter set: ' + str(grid_tfidf_lr.best_params_))
print('Best accuracy: %.3f' % grid_tfidf_lr.best_score_)
```

```
Best parameter set: {'clf__C': 1.0, 'tfidf__max_features': 15000, 'tfidf__ngram_range': (1, 2), 'tfidf__norm': 'l2', 'tfidf__pr
eprocessor': None, 'tfidf__smooth_idf': True, 'tfidf__stop_words': None, 'tfidf__tokenizer': None, 'tfidf__use_idf': True}
Best accuracy: 0.889
```

The number of features which gives the maximum accuracy is 15000.Train the data set in combination with max_features = 15000 and the other params we choose earlier gives:

```
In [19]:
# training: tf-idf + logistic regression
# you should explore different representations and algorithms.
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
max_feature_num = 15000
n_range = (1,2)
preprocess = None
stp_words = ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'you
nm = 'l2'
smoothIDF = True
token = None
useIDF = True
train_vectorizer = TfidfVectorizer(preprocessor=preprocess, tokenizer=token, stop_words=stp_words,ngram_range=n_range, max_f
train_vecs = train_vectorizer.fit_transform(train_text)
test_vecs = TfidfVectorizer(preprocessor=preprocess, tokenizer=token, stop_words=stp_words,ngram_range=n_range, max_features

# train model
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(C=1.0,max_iter = 5000).fit(train_vecs, train_labels)

# test model
test_pred = clf.predict(test_vecs)
from sklearn.metrics import precision_recall_fscore_support,accuracy_score
custom_tfidf_acc = accuracy_score(test_labels, test_pred)
pre, rec, f1, _ = precision_recall_fscore_support(test_labels, test_pred, average='macro')
print('acc', custom_tfidf_acc)
print('precision', pre)
print('rec', rec)
print('f1', f1)
```

```
acc 0.8901875
precision 0.8903340277346301
rec 0.8901874999999999
f1 0.8901771933869965
```

The accuracy now has improved to 0.890175

Tuning max_df

```
    return final_text
stop_words = stopwords.words('english')
stp_words = ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'you

vectorizer = TfidfVectorizer(strip_accents=None, lowercase=True)
clf = LogisticRegression(random_state=0,max_iter = 5000)
param_grid = [{'tfidf__ngram_range': [(1,2)],
               'tfidf__max_df':[0.25, 0.5, 0.75, 1.0],
               'tfidf__stop_words': [stp_words,None],
               'tfidf__tokenizer': [None],
               'tfidf__preprocessor': [None],
               'tfidf__use_idf':[True],
               'tfidf__max_features':[15000],
               'tfidf__norm':['l2'],
               'tfidf__smooth_idf':[True],
               'clf__C': [1.0]}]

pipe_tfidf_lr = Pipeline([('tfidf', vectorizer),
                     ('clf', clf )])
grid_tfidf_lr = GridSearchCV(pipe_tfidf_lr, param_grid, scoring='accuracy', cv=5, verbose=1, n_jobs=-1)
```

In [21]:

```
grid_tfidf_lr.fit(train_text,train_labels)
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:  4.3min finished
```

Out[21]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('tfidf',
                                        TfidfVectorizer(analyzer='word',
                                                        binary=False,
                                                        decode_error='strict',
                                                        dtype=<class 'numpy.float64'>,
                                                        encoding='utf-8',
                                                        input='content',
                                                        lowercase=True,
                                                        max_df=1.0,
                                                        max_features=None,
                                                        min_df=1,
                                                        ngram_range=(1, 1),
                                                        norm='l2',
                                                        preprocessor=None,
                                                        smooth_idf=True,
                                                        stop_words=None,
                                                        strip_acc...
                                                 'we', 'our', 'ours',
                                                 'ourselves', 'you', "you're",
                                                 "you've", "you'll", "you'd",
                                                 'your', 'yours', 'yourself',
                                                 'yourselves', 'he', 'him',
                                                 'his', 'himself', 'she',
                                                 "she's", 'her', 'hers',
                                                 'herself', 'it', "it's", 'its',
                                                 'itself', ...],
                                                None],
                         'tfidf__tokenizer': [None],
```

In [22]:

```
print('Best parameter set: ' + str(grid_tfidf_lr.best_params_))
print('Best accuracy: %.3f' % grid_tfidf_lr.best_score_)
```

```
Best parameter set: {'clf__C': 1.0, 'tfidf__max_df': 0.25, 'tfidf__max_features': 15000, 'tfidf__ngram_range': (1, 2), 'tfidf__
norm': 'l2', 'tfidf__preprocessor': None, 'tfidf__smooth_idf': True, 'tfidf__stop_words': None, 'tfidf__tokenizer': None, 'tfid
f__use_idf': True}
Best accuracy: 0.893
```

In [28]:

```
# training: tf-idf + logistic regression
# you should explore different representations and algorithms.
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
max_feature_num = 15000
n_range = (1,2)
preprocess = None

smoothIDF = True
token = None
useIDF = True
maxdf = 0.25
train_vectorizer = TfidfVectorizer(preprocessor=preprocess, tokenizer=token, stop_words=stp_words,ngram_range=n_range,max_df
train_vecs = train_vectorizer.fit_transform(train_text)
test_vecs = TfidfVectorizer(preprocessor=preprocess, tokenizer=token, stop_words=stp_words,ngram_range=n_range,max_df = maxd

# train model
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(C=1.0,max_iter = 5000).fit(train_vecs, train_labels)
stp_words = None
nm = 'l2'
# test model
test_pred = clf.predict(test_vecs)
from sklearn.metrics import precision_recall_fscore_support,accuracy_score
custom_tfidf_acc = accuracy_score(test_labels, test_pred)
pre, rec, f1, _ = precision_recall_fscore_support(test_labels, test_pred, average='macro')
print('acc', custom_tfidf_acc)
print('precision', pre)
print('rec', rec)
print('f1', f1)
```

```
acc 0.8895625
precision 0.8898336841476866
rec 0.8895625
f1 0.8895432904563844
```

Accuracy has now decreased to .8895

Tuning logistic regression params

```
    for sent in all_text:
        final_text = [porter.stem(w) for w in sent.split()]
    return final_text
stop_words = stopwords.words('english')
stp_words = ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'you

vectorizer = TfidfVectorizer(strip_accents=None, lowercase=True)
clf = LogisticRegression(random_state=0,max_iter = 5000)
param_grid = [{'tfidf__ngram_range': [(1,2)],
               'tfidf__max_df':[0.25],
               'tfidf__stop_words': [stp_words,None],
               'tfidf__tokenizer': [None],
               'tfidf__preprocessor': [None],
               'tfidf__use_idf':[True],
               'tfidf__max_features':[15000],
               'tfidf__norm':['l2'],
               'tfidf__smooth_idf':[True],
               'clf__C': [0.1,0.5,1.0,10.0,100.0]}]

pipe_tfidf_lr = Pipeline([('tfidf', vectorizer),
                          ('clf', clf )])
grid_tfidf_lr = GridSearchCV(pipe_tfidf_lr, param_grid, scoring='accuracy', cv=5, verbose=1, n_jobs=-1)
```

In [43]:

```
grid_tfidf_lr.fit(train_text,train_labels)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  5.8min
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed:  8.1min finished
```

Out[43]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('tfidf',
                                        TfidfVectorizer(analyzer='word',
                                                        binary=False,
                                                        decode_error='strict',
                                                        dtype=<class 'numpy.float64'>,
                                                        encoding='utf-8',
                                                        input='content',
                                                        lowercase=True,
                                                        max_df=1.0,
                                                        max_features=None,
                                                        min_df=1,
                                                        ngram_range=(1, 1),
                                                        norm='l2',
                                                        preprocessor=None,
                                                        smooth_idf=True,
                                                        stop_words=None,
                                                        strip_acc...
                                              'we', 'our', 'ours',
                                              'ourselves', 'you', "you're",
                                              "you've", "you'll", "you'd",
                                              'your', 'yours', 'yourself',
                                              'yourselves', 'he', 'him',
                                              'his', 'himself', 'she',
                                              "she's", 'her', 'hers',
                                              'herself', 'it', "it's", 'its',
```

```
print('Best parameter set: ' + str(grid_tfidf_lr.best_params_))
print('Best accuracy: %.3f' % grid_tfidf_lr.best_score_)
```

```
Best parameter set: {'clf__C': 1.0, 'tfidf__max_df': 0.25, 'tfidf__max_features': 15000, 'tfidf__ngram_range': (1, 2), 'tfidf__norm': 'l2', 'tfidf__preprocessor': None, 'tfidf__smooth_idf': True, 'tfidf__stop_words': None, 'tfidf__tokenizer': None, 'tfidf__use_idf': True}
Best accuracy: 0.898
```

The best value of C is 1.00

Tuning the penalty:

```
    return final_text
stop_words = stopwords.words('english')
stp_words = ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'you

vectorizer = TfidfVectorizer(strip_accents=None, lowercase=True)
clf = LogisticRegression(random_state=0,max_iter = 5000)
param_grid = [{'tfidf__ngram_range': [(1,2)],
               'tfidf__max_df':[0.25],
               'tfidf__stop_words': [stp_words,None],
               'tfidf__tokenizer': [None],
               'tfidf__preprocessor': [None],
               'tfidf__use_idf':[True],
               'tfidf__max_features':[15000],
               'tfidf__norm':['l2'],
               'tfidf__smooth_idf':[True],
               'clf__penalty':['l1','l2'],
               'clf__C': [1.0],
               'clf__solver':['liblinear']}]

pipe_tfidf_lr = Pipeline([('tfidf', vectorizer),
                          ('clf', clf )])
grid_tfidf_lr = GridSearchCV(pipe_tfidf_lr, param_grid, scoring='accuracy', cv=5, verbose=1, n_jobs=-1)
```

In [48]:

```
grid_tfidf_lr.fit(train_text,train_labels)
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  20 out of  20 | elapsed:  3.2min finished
```

Out[48]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('tfidf',
                                        TfidfVectorizer(analyzer='word',
                                                        binary=False,
                                                        decode_error='strict',
                                                        dtype=<class 'numpy.float64'>,
                                                        encoding='utf-8',
                                                        input='content',
                                                        lowercase=True,
                                                        max_df=1.0,
                                                        max_features=None,
                                                        min_df=1,
                                                        ngram_range=(1, 1),
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=1)
```

In [49]:

```
print('Best parameter set: ' + str(grid_tfidf_lr.best_params_))
print('Best accuracy: %.3f' % grid_tfidf_lr.best_score_)
```

```
Best parameter set: {'clf__C': 1.0, 'clf__penalty': 'l2', 'clf__solver': 'liblinear', 'tfidf__max_df': 0.25, 'tfidf__max_featur
es': 15000, 'tfidf__ngram_range': (1, 2), 'tfidf__norm': 'l2', 'tfidf__preprocessor': None, 'tfidf__smooth_idf': True, 'tfidf__
stop_words': None, 'tfidf__tokenizer': None, 'tfidf__use_idf': True}
Best accuracy: 0.898
```

In [50]:

```python
# training: tf-idf + logistic regression
# you should explore different representations and algorithms.
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
max_feature_num = 15000
n_range = (1,2)
preprocess = None
stp_words = None

smoothIDF = True
token = None
useIDF = True
maxdf = 0.25
train_vectorizer = TfidfVectorizer(preprocessor=preprocess, tokenizer=token, stop_words=stp_words,ngram_range=n_range,max_df
train_vecs = train_vectorizer.fit_transform(train_text)
test_vecs = TfidfVectorizer(preprocessor=preprocess, tokenizer=token, stop_words=stp_words,ngram_range=n_range,max_df = maxd

# train model
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(penalty = 'l2',C=1.0,solver  = 'liblinear',max_iter = 5000).fit(train_vecs, train_labels)
stp_words = None
nm = 'l2'
# test model
test_pred = clf.predict(test_vecs)
from sklearn.metrics import precision_recall_fscore_support,accuracy_score
custom_tfidf_acc = accuracy_score(test_labels, test_pred)
pre, rec, f1, _ = precision_recall_fscore_support(test_labels, test_pred, average='macro')
print('acc', custom_tfidf_acc)
print('precision', pre)
print('rec', rec)
print('f1', f1)
```

```
acc 0.8994
precision 0.8994578972578153
rec 0.8993767900286405
f1 0.8993911101984972
```

The solver used here is liblinear and penalty is ‘l2’(Ridge regression)

Comparing all best params we obtain from above:

50:50 Train/test split with test_size 0.3(balanced classes)

```python

from sklearn.model_selection import train_test_split

train_text, test_text, train_labels, test_labels = train_test_split(all_text, all_labels, test_size=0.3, random_state=0, str

```

Here the parameter stratify make the number of pos and neg classes equal.

```python
accuracy_list = [acc,best_ngram_acc, best_num_features_acc, best_maxdf_acc,best_penalty_acc]
best_params = ['acc','best_ngram_acc', 'best_num_features_acc', 'best_maxdf_acc','best_penalty_acc']
accuracy_df = pd.DataFrame()
accuracy_df['best_params']= ['acc','best_ngram_acc', 'best_num_features_acc', 'best_maxdf_acc','best_penalty_acc']
accuracy_df['accuracy']= accuracy_list
accuracy_df.sort_values('accuracy', ascending=False)
```

	best_params	accuracy
3	best_maxdf_acc	0.900750
4	best_penalty_acc	0.900750
2	best_num_features_acc	0.894000
1	best_ngram_acc	0.888500
0	acc	0.866583

50:50 train/test split with test_size 0.4(balanced classes)

```
from sklearn.model_selection import train_test_split

train_text, test_text, train_labels, test_labels = train_test_split(all_text, all_labels, test_size=0.4, random_state=0, str
```

```
accuracy_list = [acc,best_ngram_acc, best_num_features_acc, best_maxdf_acc,best_penalty_acc]
best_params = ['acc','best_ngram_acc', 'best_num_features_acc', 'best_maxdf_acc','best_penalty_acc']
accuracy_df = pd.DataFrame()
accuracy_df['best_params']= ['acc','best_ngram_acc', 'best_num_features_acc', 'best_maxdf_acc','best_penalty_acc']
accuracy_df['accuracy']= accuracy_list
accuracy_df.sort_values('accuracy', ascending=False)
```

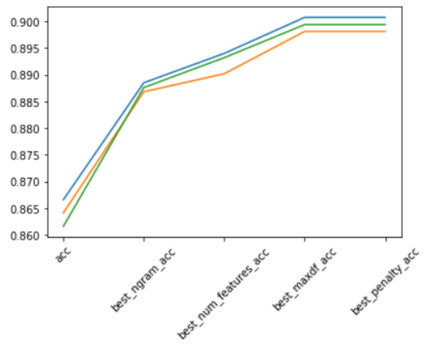
	best_params	accuracy
3	best_maxdf_acc	0.898125
4	best_penalty_acc	0.898125
2	best_num_features_acc	0.890188
1	best_ngram_acc	0.886813
0	acc	0.864125

Train/test split already given:

```
train_text = all_text[:35000]
train_labels = all_labels[:35000]
test_text = all_text[35000:]
test_labels = all_labels[35000:]
```

Comparing different train/test split ratios

```
In [33]:
accuracy_list_03 = [0.8665833333333334, 0.8885, 0.894, 0.90075, 0.90075]
accuracy_list_04 = [0.864125, 0.8868125, 0.8901875, 0.898125, 0.898125]
accuracy_initial = [0.8616, 0.8876, 0.8932, 0.8994, 0.8994]
best_params = ['acc','best_ngram_acc', 'best_num_features_acc', 'best_maxdf_acc','best_penalty_acc']
plt.figure()
plt.plot(best_params,accuracy_list_03,label = 'test_size = 0.3',zorder = 1)
plt.plot(best_params,accuracy_list_04,label = 'test_size = 0.4',zorder = 2)
plt.plot(best_params,accuracy_initial,label = 'train/test initial',zorder = 10)
plt.xticks(rotation = 45)
plt.show()
```



test_size = 0.3 shows maximum accuracy

SGD based sentiment analysis: (Feature based method)

SGD classifiers are classifiers with with a stochastic gradient descent (SGD) learning for gradient loss.

```
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
max_feature_num = 15000
n_range = (1,2)
preprocess = None
stp_words = None

smoothIDF = True
token = None
useIDF = True
maxdf = 0.25
train_vectorizer = TfidfVectorizer(preprocessor=preprocess, tokenizer=token, stop_words=stp_words,ngram_range=n_range,max_df
train_vecs = train_vectorizer.fit_transform(train_text)
test_vecs = TfidfVectorizer(preprocessor=preprocess, tokenizer=token, stop_words=stp_words,ngram_range=n_range,max_df = maxd

# train model
from sklearn.linear_model import SGDClassifier
clf = SGDClassifier(loss='hinge', penalty='l2', alpha=1e-3, random_state=0, max_iter=100,learning_rate='optimal',tol=None).f
stp_words = None
nm = 'l2'
# test model
test_pred = clf.predict(test_vecs)
from sklearn.metrics import precision_recall_fscore_support,accuracy_score
best_penalty_acc = accuracy_score(test_labels, test_pred)
pre, rec, f1, _ = precision_recall_fscore_support(test_labels, test_pred, average='macro')
print('acc', best_penalty_acc)
print('precision', pre)
print('rec', rec)
print('f1', f1)
```

```
acc 0.8679166666666667
precision 0.8713750035434138
rec 0.8679166666666667
f1 0.8676084508658104
```

Decision Tree Based Sentiment analysis:

```
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
max_feature_num = 15000
n_range = (1,2)
preprocess = None
stp_words = None

smoothIDF = True
token = None
useIDF = True
maxdf = 0.25
train_vectorizer = TfidfVectorizer(preprocessor=preprocess, tokenizer=token, stop_words=stp_words,ngram_range=n_range,max_df
train_vecs = train_vectorizer.fit_transform(train_text)
test_vecs = TfidfVectorizer(preprocessor=preprocess, tokenizer=token, stop_words=stp_words,ngram_range=n_range,max_df = maxd

# train model
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(criterion='entropy', random_state=0).fit(train_vecs, train_labels)
stp_words = None
nm = 'l2'
# test model
test_pred = clf.predict(test_vecs)
from sklearn.metrics import precision_recall_fscore_support,accuracy_score
dec_tree_accuracy = accuracy_score(test_labels, test_pred)
pre, rec, f1, _ = precision_recall_fscore_support(test_labels, test_pred, average='macro')
print('acc', dec_tree_accuracy)
print('precision', pre)
print('rec', rec)
print('f1', f1)
```

```
acc 0.7248333333333333
precision 0.7248335581668914
rec 0.7248333333333334
f1 0.7248332645416495
```

Naïve Bayes based sentiment analysis:

```
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
max_feature_num = 15000
n_range = (1,2)
preprocess = None
stp_words = None

smoothIDF = True
token = None
useIDF = True
maxdf = 0.25
train_vectorizer = TfidfVectorizer(preprocessor=preprocess, tokenizer=token, stop_words=stp_words,ngram_range=n_range,max_df
train_vecs = train_vectorizer.fit_transform(train_text)
test_vecs = TfidfVectorizer(preprocessor=preprocess, tokenizer=token, stop_words=stp_words,ngram_range=n_range,max_df = maxd

# train model
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(train_vecs, train_labels)
stp_words = None
nm = 'l2'
# test model
test_pred = clf.predict(test_vecs)
from sklearn.metrics import precision_recall_fscore_support,accuracy_score
MNB_accuracy = accuracy_score(test_labels, test_pred)
pre, rec, f1, _ = precision_recall_fscore_support(test_labels, test_pred, average='macro')
print('acc', MNB_accuracy)
print('precision', pre)
print('rec', rec)
print('f1', f1)
```

```
acc 0.8681666666666666
precision 0.8682532470967619
rec 0.8681666666666668
f1 0.8681589173408082
```

SVC based sentiment analysis:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
max_feature_num = 15000
n_range = (1,2)
preprocess = None
stp_words = None

smoothIDF = True
token = None
useIDF = True
maxdf = 0.25
train_vectorizer = TfidfVectorizer(preprocessor=preprocess, tokenizer=token, stop_words=stp_words,ngram_range=n_range,max_df
train_vecs = train_vectorizer.fit_transform(train_text)
test_vecs = TfidfVectorizer(preprocessor=preprocess, tokenizer=token, stop_words=stp_words,ngram_range=n_range,max_df = maxd

# train model
from sklearn.svm import SVC
clf = SVC(kernel='linear').fit(train_vecs, train_labels)
stp_words = None
nm = 'l2'
# test model
test_pred = clf.predict(test_vecs)
from sklearn.metrics import precision_recall_fscore_support,accuracy_score
SVC_accuracy = accuracy_score(test_labels, test_pred)
pre, rec, f1, _ = precision_recall_fscore_support(test_labels, test_pred, average='macro')
print('acc', SVC_accuracy)
print('precision', pre)
print('rec', rec)
print('f1', f1)
```

```
acc 0.8975
precision 0.8975481033205017
rec 0.8975
f1 0.8974968992812034
```

Comparing the accuracy of different algorithms

```
accuracy_df = pd.DataFrame()
accuracy_df['model']= ['best_lr_accuracy','sgd_accuracy','dec_tree_accuracy','SVC_accuracy','MNB_accuracy']
accuracy_df['accuracy']= [best_lr,sgd_accuracy,dec_tree_accuracy,SVC_accuracy,MNB_accuracy]
accuracy_df.sort_values('accuracy',ascending = False)
accuracy_df
```

	model	accuracy
0	best_lr_accuracy	0.900750
1	sgd_accuracy	0.867917
2	dec_tree_accuracy	0.724833
3	SVC_accuracy	0.897500
4	MNB_accuracy	0.868167

Logistic Regression performs better than all other algorithms

So the final model I chose is TF-IDF vectorization with Logistic regression , 50:50 train test split(balanced classes),max_features = 15000,max_df = 0.25,and C= 1.00, penalty = ‘l2’ and solver = ‘liblinear’

#MarkingId:25309