Import necessary dependencies

In [1]:
```
import pandas
from matplotlib import pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
import numpy
from sklearn.feature_selection import chi2
from PIL import Image
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
from collections import Counter
import re
import sqlite3
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import string
```

Read in the data

In [9]:
```
train_data = pandas.read_csv("./data/train.csv", header=None)
train_data.head()
```

Out[9]:

	0	1	2
0	3	Wall St. Bears Claw Back Into the Black (Reuters)	Reuters - Short-sellers, Wall Street's dwindli...
1	3	Carlyle Looks Toward Commercial Aerospace (Reu...	Reuters - Private investment firm Carlyle Grou...
2	3	Oil and Economy Cloud Stocks' Outlook (Reuters)	Reuters - Soaring crude prices plus worries\ab...
3	3	Iraq Halts Oil Exports from Main Southern Pipe...	Reuters - Authorities have halted oil export\f...
4	3	Oil prices soar to all-time record, posing new...	AFP - Tearaway world oil prices, toppling reco...

In [10]:
```
test_data = pandas.read_csv("./data/test.csv", header=None)
test_data.head()
```

Out[10]:

	0	1	2
0	3	Fears for T N pension after talks	Unions representing workers at Turner Newall...
1	4	The Race is On: Second Private Team Sets Launc...	SPACE.com - TORONTO, Canada -- A second\team o...
2	4	Ky. Company Wins Grant to Study Peptides (AP)	AP - A company founded by a chemistry research...
3	4	Prediction Unit Helps Forecast Wildfires (AP)	AP - It's barely dawn when Mike Fitzpatrick st...
4	4	Calif. Aims to Limit Farm-Related Smog (AP)	AP - Southern California's smog-fighting agenc...

Save necessary data in database, first open connection.

In [12]:
```
db = sqlite3.connect('newsclassifier.db')
cat_list = pandas.read_csv('./data/classes.txt', header=None)
cat_list.head()
cat_list.to_sql("category_list", db, if_exists='replace')
```

Data Cleaning

In [11]:
```
train_data.columns = ['category', 'headline', 'content']
train_data.head()
```

Out[11]:

	category	headline	content
0	3	Wall St. Bears Claw Back Into the Black (Reuters)	Reuters - Short-sellers, Wall Street's dwindli...
1	3	Carlyle Looks Toward Commercial Aerospace (Reu...	Reuters - Private investment firm Carlyle Grou...
2	3	Oil and Economy Cloud Stocks' Outlook (Reuters)	Reuters - Soaring crude prices plus worries\ab...
3	3	Iraq Halts Oil Exports from Main Southern Pipe...	Reuters - Authorities have halted oil export\f...
4	3	Oil prices soar to all-time record, posing new...	AFP - Tearaway world oil prices, toppling reco...

In [12]:
```
test_data.columns = ['category', 'headline', 'content']
test_data.head()
```

Out[12]:

	category	headline	content
0	3	Fears for T N pension after talks	Unions representing workers at Turner Newall...
1	4	The Race is On: Second Private Team Sets Launc...	SPACE.com - TORONTO, Canada -- A second\team o...
2	4	Ky. Company Wins Grant to Study Peptides (AP)	AP - A company founded by a chemistry research...
3	4	Prediction Unit Helps Forecast Wildfires (AP)	AP - It's barely dawn when Mike Fitzpatrick st...
4	4	Calif. Aims to Limit Farm-Related Smog (AP)	AP - Southern California's smog-fighting agenc...

Sample 1000 rows

In [17]:
```
train_data_sample = train_data.sample(n = 5000, replace = False, random_state = 123)
train_data_sample.head()
```

Out[17]:

	category	headline	content
30870	2	NHL on Ice, Maybe for Whole 2004-05 Season (AP)	AP - No shots, no saves, no goals. The Nationa...
7738	2	Rowers to be punished for criticism of teammate	ROWER Sally Robbins #39;s teammates are expect...
25351	2	Changing Directions	Over at USA Today -- Slogan: "All the News Tha...
74309	4	Cassini snapshots murky moon Titan	The Cassini probe got the first close-up photo...
88347	1	Farewell Yasser Arafat	GAZA CITY, - The world will bid farewell to Ab...

In [19]:
```
test_data_sample = test_data.sample(n = 5000, replace = False, random_state = 123)
test_data_sample.head()
```

Out[19]:

	category	headline	content
646	1	Panama pardons Castro 'plotters'	Four men accused of planning to kill Cuba's Fi...
2616	4	Elephant DNA Could Help Stem Ivory Trade (AP)	AP - Analyzing the DNA of elephants may help t...
2300	1	Job-Loss Panic Rises in Western Europe (AP)	AP - Stephane Zervos first suspected his job w...
4764	1	Remark on Homosexuality Delays Seating of Euro...	The European Union #39;s normally yawn-inducin...
3617	3	Linux: Paris weighs a shift to open-source camp	PARIS The open-source computer system known as...

Clean HTML code & news sources from headline

In [20]:
```
import string
import re

def clean(x):
    # strip HTML and sources of the format eg. "&lt and (Reuters)"
    x = re.sub(r'(&[A-Za-z]+)|\(.*\)', '', x)
    # split into words
    tokens = word_tokenize(x)
    # convert to lower case
    tokens = [w.lower() for w in tokens]
    # # remove punctuation from each word
    # table = str.maketrans(string.punctuation, ' ')
    # stripped = [w.translate(table) for w in tokens]
    # remove remaining tokens that are not alphabetic
    words = [word for word in tokens if word.isalpha()]
    # filter out stop words
    stop_words = set(stopwords.words('english'))
    words = [w for w in words if not w in stop_words]
    # re-create document from words
    doc = ' '.join(words)
    return str(doc)

for i, row in train_data_sample.iterrows():
    train_data_sample.at[i, "headline_cleaned"] = clean(row.headline)

for i, row in test_data_sample.iterrows():
    test_data_sample.at[i, "headline_cleaned"] = clean(row.headline)
```

Clean news sources from content

In [21]:
```
# Function to clean out the dates
def clean_dates(x):
    x = re.sub(r'[0-9 ]*(January|February|March|April|May|June|July|August|September|October|November|December|JANUARY|FEBRUARY|MARCH
    return x

# Clean out the dates for training data
for i, row in train_data.iterrows():
    train_data.at[i, "content_cleaned"] = clean_dates(row.content)

# Clean out the dates for testing data
for i, row in test_data.iterrows():
    test_data.at[i, "content_cleaned"] = clean_dates(row.content)
```

In [70]:
```
# Generate the sources list
def extract_sources(x):
    sources = []
    for sentence in x.content:
        trimmed = sentence[:35]
        temp = re.search(r'^[A-Za-z0-9\/,. ]*\(*[A-Za-z.]+\)* -', trimmed)
        if temp is not None:
            sources.append(temp.group())
    sources = numpy.array(sources)
    sources = numpy.unique(sources)
    return sources

# Call function to generate the sources list and save it
sources = extract_sources(train_data.content_cleaned)
sources = numpy.append(sources, extract_sources(test_data.content_cleaned))
sources = numpy.unique(sources)

# Save the sources to a file
numpy.savetxt("./data/news_sources_from_both.csv", sources, \
             header='list', delimiter=',', fmt='%s')
```

In [51]:
```
sources_data = pandas.read_csv("./data/news_sources_clean_v1.csv")

def remove_sources(x, sources):
    x = str(x)
    for i, source in sources.iterrows():
        if source.list in x:
            x = x.replace(source.list, ' ')
    return x

for i, row in train_data.iterrows():
    train_data.at[i, "content_nosources"] = remove_sources(row.content_cleaned, sources_df)

for i, row in test_data.iterrows():
    test_data.at[i, "content_nosources"] = remove_sources(row.content_cleaned, sources_df)

for i, row in train_data.iterrows():
    train_data.at[i, "content_cleaned"] = clean(row.content_nosources)
train_data.head()

for i, row in test_data.iterrows():
    test_data.at[i, "content_cleaned"] = clean(row.content_nosources)
test_data.head()
```

```
National Hockey League
USA Today
Foreign Ministry said
Wall Street Journal
IBM
Interior Minister Dominique

---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-51-1557a9100804> in <module>
     10
     11 for i, row in train_data_sample.iterrows():
---> 12     train_data_sample.at[i, "content_cleaned"] = remove_sources(row.content)

<ipython-input-51-1557a9100804> in remove_sources(x)
      4     x = str(x)
      5     for i, source in sources_data.iterrows():
----> 6         if source.list in x:
      7             print(source.list)
      8             x = x.replace(source.list, ' ')

~\.conda\envs\env1\lib\site-packages\pandas\core\generic.py in __getattr__(self, name)
   5271         else:
   5272             if self._info_axis._can_hold_identifiers_and_holds_name(name):
-> 5273                 return self[name]
   5274             return object.__getattribute__(self, name)
   5275

~\.conda\envs\env1\lib\site-packages\pandas\core\series.py in __getitem__(self, key)
    869         key = com.apply_if_callable(key, self)
    870         try:
--> 871             result = self.index.get_value(self, key)
    872
    873             if not is_scalar(result):

~\.conda\envs\env1\lib\site-packages\pandas\core\indexes\base.py in get_value(self, series, key)
   4400         k = com.values_from_object(key)
   4401
-> 4402         k = self._convert_scalar_indexer(k, kind="getitem")
   4403         try:
   4404             return self._engine.get_value(s, k, tz=getattr(series.dtype, "tz", None))

~\.conda\envs\env1\lib\site-packages\pandas\core\indexes\base.py in _convert_scalar_indexer(self, key, kind)
   2857             return self._validate_indexer("positional", key, kind)
   2858
-> 2859         if len(self) and not isinstance(self, ABCMultiIndex):
   2860
   2861             # we can raise here if we are definitive that this

KeyboardInterrupt:
```

Save to Database

In [ ]:
```
db = sqlite3.connect('./data/newsclassifier.db')
cat_list = pandas.read_csv('./data/classes.txt', header=None)
cat_list.head()
cat_list.to_sql("category_list", db, if_exists='replace')
train_data.to_sql('train_data', db, if_exists='replace')
test_data.to_sql('test_data', db, if_exists='replace')
train_data_sample.to_sql('train_data_sample', db, if_exists='replace')
test_data_sample.to_sql('test_data_sample', db, if_exists='replace')
db.commit()
db.close()
```

Make a CountVector (Bag of words)

In [7]:
```
# create a CountVectorizer from raw data, with options to clean it
cv = CountVectorizer(min_df = 2, lowercase = True, token_pattern=r'(?u)\b[A-Za-z]{2,}\b',
                         strip_accents = 'ascii', ngram_range = (1, 1),
                         stop_words = 'english')
cv_matrix = cv.fit_transform(train_data_sample.headline).toarray()

# get all unique words in the corpus
vocab = cv.get_feature_names()

# produce a dataframe including the feature names
cv_matrix_df = pandas.DataFrame(cv_matrix, columns=vocab)
```

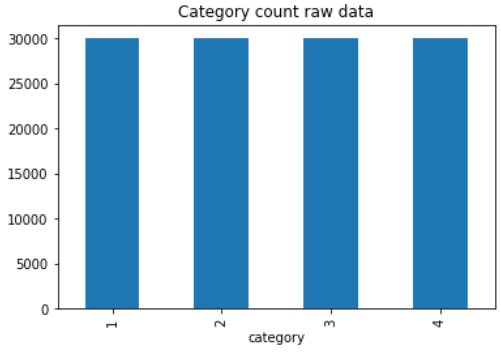
Out[7]:
```
'======================================================================'
```

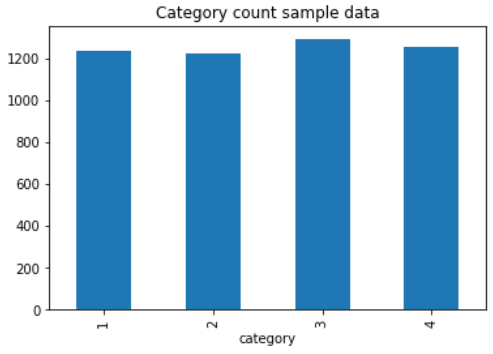
Save the bag of words

In [ ]:
```
cv_matrix_df.to_sql('headline_bagofwords', db, if_exists='replace')
db.commit()
db.close()
```

Data Exploration

In [18]:
```
# bar plot of the count of unique things in each category
train_data.groupby('category').headline.count().plot.bar(ylim = 0)
plt.title("Category count raw data")
plt.show()
train_data_sample.groupby('category').headline.count().plot.bar(ylim = 0)
plt.title("Category count sample data")
plt.show()
```





The number of unique documents in each category

In [9]:
```
print(pandas.DataFrame(train_data_sample.groupby(['category']).count()))
```

```
          headline  content  content_cleaned
category
1              236      236              236
2              251      251              251
3              258      258              258
4              255      255              255
```

The count of observations and features

In [10]:
```
print("There are {} observations and {} features in this dataset. \n".\
    format(cv_matrix_df.shape[0],cv_matrix_df.shape[1]))
```

```
There are 1000 observations and 893 features in this dataset.
```

A description of the categories

In [11]:
```
categories = train_data_sample.groupby("category")
categories.describe().head()
```

Out[11]:

	headline				content				content_cleaned	
	count	unique	top	freq	count	unique	top	freq	top	freq
category										
1	236	236	Bush's Convention Tops Kerry's in Primetime Po...	1	236	235	TAIPEI (Reuters) - The pro-independence party...	2	TAIPEI (Reuters) - The pro-independence party...	2
2	251	251	Edwards banned from Games	1	251	251	ISTANBUL, Turkey -- Striker Andriy Shevchenko ...	1	NEW YORK (Reuters) - Lamar Odom supplemented ...	1
3	258	258	Consumer Sentiment Improves in November	1	258	258	The Congress-led UPA government decided on Wed...	1	The Congress-led UPA government decided on Wed...	1
4	255	255	Arguments conclude in evolution sticker trial	1	255	255	com September 14, 2004, 4:00 AM PT. With the e...	1	AP - The on Thursday filed the first case in ...	1

WordCloud/TagCloud of the top words in the headlines

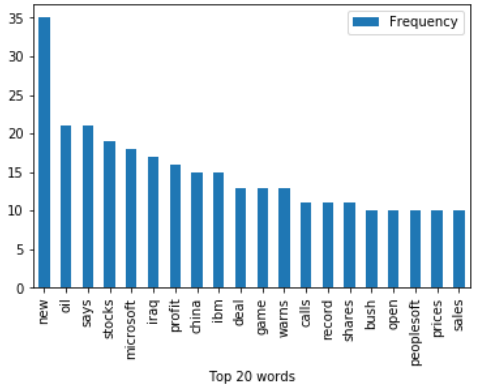
In [12]:
```
# prepare the dictionary to be used in wordcloud
word_count_dict = {}
for word in vocab:
    word_count_dict[word] = int(sum(cv_matrix_df.loc[:, word]))
```

In [13]:
```
# generate a word cloud image with top 100 words and 80% horizontal:
wordcloud = WordCloud(max_words=100, prefer_horizontal=0.8, background_color='white').\
            generate_from_frequencies(word_count_dict)

# display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

Plots of the data

Bar plot of the top word counts

In [18]:
```
from collections import Counter

counter = Counter(word_count_dict)

freq_df = pandas.DataFrame.from_records(counter.most_common(20),
                                        columns=['Top 20 words', 'Frequency'])
freq_df.plot(kind='bar', x='Top 20 words');
```



In [1]:
```
# To add a new cell, type '# %%'
# To add a new markdown cell, type '# %% [markdown]'
# %% [markdown]
# # Feature Engineering, Baseline Model and Feature Selection
```

Import necessary dependencies

In [2]:
```
import pandas
from matplotlib import pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
import numpy
from sklearn.feature_selection import chi2
from PIL import Image
from collections import Counter
import re
import sqlite3
from sklearn import decomposition, ensemble
import nltk
from keras.preprocessing import text
from keras.utils import np_utils
from keras.preprocessing import sequence
import pydot
import seaborn as sns
from sklearn.metrics import precision_recall_curve # The average precision score in multi-label settings
from sklearn.metrics import average_precision_score
from sklearn import svm # Support Vector Machine
from sklearn.preprocessing import label_binarize # Split category encoding eg. y=[1,2,3] into y1=[0,1], y2=[0,1], y3=
[0,1]
from sklearn.model_selection import train_test_split # Built-in train test splitter
from sklearn.multiclass import OneVsRestClassifier # We use OneVsRestClassifier for multi-label prediction
from itertools import cycle
from sklearn.feature_selection import SelectPercentile, f_classif
```

```
Using TensorFlow backend.
```

Load in the data from the database

In [3]:
```
dbconn = sqlite3.connect('./data/cleanedtraintest_v2.db')
train_data_df = pandas.read_sql_query('SELECT * FROM train_data', dbconn)
test_data_df = pandas.read_sql_query('SELECT * FROM test_data', dbconn)
dbconn.commit()
dbconn.close()
```

Check the if the data was loaded correctly

In [4]:
```
train_data_df.head()
```

Out[4]:

	index	category	headline	content	headline_cleaned	content_cleaned	content_nosources
0	0	3	Wall St. Bears Claw Back Into the Black (Reuters)	Reuters - Short-sellers, Wall Street's dwindli...	wall bears claw back black	wall street seeing green	Short-sellers, Wall Street's dwindling\band ...
1	1	3	Carlyle Looks Toward Commercial Aerospace (Reu...	Reuters - Private investment firm Carlyle Grou...	carlyle looks toward commercial aerospace	private investment firm carlyle group reputati...	Private investment firm Carlyle Group,\which...
2	2	3	Oil and Economy Cloud Stocks' Outlook (Reuters)	Reuters - Soaring crude prices plus worries\ab...	oil economy cloud stocks outlook	soaring crude prices plus economy outlook earn...	Soaring crude prices plus worries\about the ...
3	3	3	Iraq Halts Oil Exports from Main Southern Pipe...	Reuters - Authorities have halted oil export\f...	iraq halts oil exports main southern pipeline	authorities halted oil main pipeline southern ...	Authorities have halted oil export\flows fro...
4	4	3	Oil prices soar to all-time record, posing new...	AFP - Tearaway world oil prices, toppling reco...	oil prices soar record posing new menace us ec...	tearaway world oil prices toppling records str...	Tearaway world oil prices, toppling records ...

In [5]:
```
train_data_df.drop('index', axis=1, inplace=True)
train_data_df.head()
```

Out[5]:

	category	headline	content	headline_cleaned	content_cleaned	content_nosources
0	3	Wall St. Bears Claw Back Into the Black (Reuters)	Reuters - Short-sellers, Wall Street's dwindli...	wall bears claw back black	wall street seeing green	Short-sellers, Wall Street's dwindling\band ...
1	3	Carlyle Looks Toward Commercial Aerospace (Reu...	Reuters - Private investment firm Carlyle Grou...	carlyle looks toward commercial aerospace	private investment firm carlyle group reputati...	Private investment firm Carlyle Group,\which...
2	3	Oil and Economy Cloud Stocks' Outlook (Reuters)	Reuters - Soaring crude prices plus worries\ab...	oil economy cloud stocks outlook	soaring crude prices plus economy outlook earn...	Soaring crude prices plus worries\about the ...
3	3	Iraq Halts Oil Exports from Main Southern Pipe...	Reuters - Authorities have halted oil export\f...	iraq halts oil exports main southern pipeline	authorities halted oil main pipeline southern ...	Authorities have halted oil export\flows fro...
4	3	Oil prices soar to all-time record, posing new...	AFP - Tearaway world oil prices, toppling reco...	oil prices soar record posing new menace us ec...	tearaway world oil prices toppling records str...	Tearaway world oil prices, toppling records ...

In [6]:
```
test_data_df.head()
```

Out[6]:

	index	category	headline	content	headline_cleaned	content_cleaned	content_nosources
0	0	3	Fears for T N pension after talks	Unions representing workers at Turner Newall...	fears n pension talks	unions representing workers turner newall say ...	Unions representing workers at Turner Newall...
1	1	4	The Race is On: Second Private Team Sets Launc...	SPACE.com - TORONTO, Canada -- A second\team o...	race second private team sets launch date huma...	toronto canada rocketeers competing million an...	TORONTO, Canada -- A second\team of rocketee...
2	2	4	Ky. Company Wins Grant to Study Peptides (AP)	AP - A company founded by a chemistry research...	company wins grant study peptides	company founded chemistry researcher universit...	A company founded by a chemistry researcher ...
3	3	4	Prediction Unit Helps Forecast Wildfires (AP)	AP - It's barely dawn when Mike Fitzpatrick st...	prediction unit helps forecast wildfires	barely dawn mike fitzpatrick starts shift blur...	It's barely dawn when Mike Fitzpatrick start...
4	4	4	Calif. Aims to Limit Farm-Related Smog (AP)	AP - Southern California's smog-fighting agenc...	calif aims limit smog	southern california agency went emissions bovi...	Southern California's smog-fighting agency w...

In [7]:
```
test_data_df.drop('index', axis=1, inplace=True)
test_data_df.head()
```

Out[7]:

	category	headline	content	headline_cleaned	content_cleaned	content_nosources
0	3	Fears for T N pension after talks	Unions representing workers at Turner Newall...	fears n pension talks	unions representing workers turner newall say ...	Unions representing workers at Turner Newall...
1	4	The Race is On: Second Private Team Sets Launc...	SPACE.com - TORONTO, Canada -- A second\team o...	race second private team sets launch date huma...	toronto canada rocketeers competing million an...	TORONTO, Canada -- A second\team of rocketee...
2	4	Ky. Company Wins Grant to Study Peptides (AP)	AP - A company founded by a chemistry research...	company wins grant study peptides	company founded chemistry researcher universit...	A company founded by a chemistry researcher ...
3	4	Prediction Unit Helps Forecast Wildfires (AP)	AP - It's barely dawn when Mike Fitzpatrick st...	prediction unit helps forecast wildfires	barely dawn mike fitzpatrick starts shift blur...	It's barely dawn when Mike Fitzpatrick start...
4	4	Calif. Aims to Limit Farm-Related Smog (AP)	AP - Southern California's smog-fighting agenc...	calif aims limit smog	southern california agency went emissions bovi...	Southern California's smog-fighting agency w...

Sample 4000 rows

In [8]:
```
train_data_sample = train_data_df.sample(n = 4000, replace = False, random_state = 123)
train_data_sample.head()
```

Out[8]:

	category	headline	content	headline_cleaned	content_cleaned	content_nosources
30870	2	NHL on Ice, Maybe for Whole 2004-05 Season (AP)	AP - No shots, no saves, no goals. The Nationa...	nhl ice maybe whole season	shots saves goals national hockey league locke...	No shots, no saves, no goals. The National H...
7738	2	Rowers to be punished for criticism of teammate	ROWER Sally Robbins #39;s teammates are expect...	rowers punished criticism teammate	rower sally robbins teammates expected face di...	ROWER Sally Robbins #39;s teammates are expect...
25351	2	Changing Directions	Over at USA Today -- Slogan: "All the News Tha...	changing directions	slogan news fit print four paragraphs less got...	Over at - Slogan: "All the News That's Fit to...
74309	4	Cassini snapshots murky moon Titan	The Cassini probe got the first close-up photo...	cassini snapshots murky moon titan	cassini probe got first photos saturn murky mo...	The Cassini probe got the first close-up photo...
88347	1	Farewell Yasser Arafat	GAZA CITY, 12 November 2004 - The world will b...	farewell yasser arafat	gaza city world bid farewell abu ammar yasser ...	GAZA CITY, - The world will bid farewell to Ab...

In [9]:
```
test_data_sample = test_data_df.sample(n = 4000, replace = False, random_state = 123)
test_data_sample.head()
```

Out[9]:

	category	headline	content	headline_cleaned	content_cleaned	content_nosources
646	1	Panama pardons Castro 'plotters'	Four men accused of planning to kill Cuba's Fi...	panama pardons castro	four men accused planning kill cuba fidel cast...	Four men accused of planning to kill Cuba's Fi...
2616	4	Elephant DNA Could Help Stem Ivory Trade (AP)	AP - Analyzing the DNA of elephants may help t...	elephant dna could help stem ivory trade	analyzing dna elephants may help trace origins...	Analyzing the DNA of elephants may help trac...
2300	1	Job-Loss Panic Rises in Western Europe (AP)	AP - Stephane Zervos first suspected his job w...	panic rises western europe	stephane zervos first suspected job threatened...	Stephane Zervos first suspected his job was ...
4764	1	Remark on Homosexuality Delays Seating of Euro...	The European Union #39;s normally yawn-inducin...	remark homosexuality delays seating european p...	european union normally institutions raised ey...	The European Union #39;s normally yawn-inducin...
3617	3	Linux: Paris weighs a shift to open-source camp	PARIS The open-source computer system known as...	linux paris weighs shift camp	paris computer system known linux tough battle...	PARIS The open-source computer system known as...

Train & Test data where x is the predictor features, y is the predicted feature

In [10]:
```
n_classes = 4

x_train = train_data_sample.content_cleaned
y_train = label_binarize(train_data_sample.category, classes=[1, 2, 3, 4])

x_test = test_data_sample.content_cleaned
y_test = label_binarize(test_data_sample.category, classes=[1, 2, 3, 4])
```

Let's make a Bag of Words

In [11]:
```
# Use countvectorizer to get a vector of words
cv = CountVectorizer(min_df = 2, lowercase = True,
                     token_pattern=r'\b[A-Za-z]{2,}\b', ngram_range = (1, 1))
x_train_cv = cv.fit_transform(x_train)
x_test_cv = cv.transform(x_test)

selector = SelectPercentile(f_classif, percentile=10)
selector.fit(x_train_cv, train_data_sample.category)
x_train_cv_10p = selector.transform(x_train_cv).toarray()
x_test_cv_10p = selector.transform(x_test_cv).toarray()

# get all unique words in the corpus
bow_vocab = cv.get_feature_names()

columns = numpy.asarray(bow_vocab)
support = numpy.asarray(selector.get_support())
bow_vocab_10p = columns[support]

x_train_cv = x_train_cv.toarray()
x_test_cv = x_test_cv.toarray()

# produce a dataframe including the feature names
x_train_bagofwords = pandas.DataFrame(x_train_cv, columns=bow_vocab)
x_test_bagofwords = pandas.DataFrame(x_test_cv, columns=bow_vocab)
x_train_bagofwords_10p = pandas.DataFrame(x_train_cv_10p, columns=bow_vocab_10p)
x_test_bagofwords_10p = pandas.DataFrame(x_test_cv_10p, columns=bow_vocab_10p)
x_train_bagofwords.head()
```

Out[11]:

	aaron	ab	abandon	abandoned	abandons	abbas	abc	abducted	abduction	abductions	...	zaragoza	zdnet	zealand	zee	zero	zimbabwe
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0

5 rows × 6873 columns

In [12]:
```
x_test_bagofwords_10p.head()
```

Out[12]:

	abducted	abu	access	according	accounting	accused	administration	adrian	afghan	afghanistan	...	writes	xp	yahoo	yankees	yards	ya
0	0	0	0	0	0	1	0	0	0	0	...	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	

5 rows × 687 columns

We have bag of words already, let's make a Bag of N-Grams

In [13]:
```
# Use countvectorizer to get a vector of ngrams
cv = CountVectorizer(min_df = 2, lowercase = True,
                     token_pattern=r'\b[A-Za-z]{2,}\b', ngram_range = (2, 3))
x_train_cv = cv.fit_transform(x_train)
x_test_cv = cv.transform(x_test)

# get all unique words in the corpus
ngram_vocab = cv.get_feature_names()

selector = SelectPercentile(f_classif, percentile=10)
selector.fit(x_train_cv, train_data_sample.category)
x_train_cv_10p = selector.transform(x_train_cv).toarray()
x_test_cv_10p = selector.transform(x_test_cv).toarray()

columns = numpy.asarray(ngram_vocab)
support = numpy.asarray(selector.get_support())
ngram_vocab_10p = columns[support]

x_train_cv = x_train_cv.toarray()
x_test_cv = x_test_cv.toarray()

# produce a dataframe including the feature names
x_train_bagofngrams = pandas.DataFrame(x_train_cv, columns=ngram_vocab)
x_test_bagofngrams = pandas.DataFrame(x_test_cv, columns=ngram_vocab)
x_train_bagofngrams_10p = pandas.DataFrame(x_train_cv_10p, columns=ngram_vocab_10p)
x_test_bagofngrams_10p = pandas.DataFrame(x_test_cv_10p, columns=ngram_vocab_10p)
x_train_bagofngrams.head()
```

Out[13]:

	ab billion	abducted militants	abductions foreigners	abductions foreigners iraq	aboard international	aboard international space	abu ghraib	abu ghraib prison	abu musab	ac milan	...	yukos said	yukos said would	zdnet survey	zdnet survey professionals	z b
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	

5 rows × 5929 columns

In [14]:
```
# Use countvectorizer to get a vector of chars
cv = CountVectorizer(analyzer='char', min_df = 2, ngram_range = (2, 3),
                     token_pattern=r'\b[A-Za-z]{2,}\b')
x_train_cv = cv.fit_transform(x_train)
x_test_cv = cv.transform(x_test)

# get all unique words in the corpus
cv_char_vocab = cv.get_feature_names()

selector = SelectPercentile(f_classif, percentile=10)
selector.fit(x_train_cv, train_data_sample.category)
x_train_cv_10p = selector.transform(x_train_cv).toarray()
x_test_cv_10p = selector.transform(x_test_cv).toarray()

columns = numpy.asarray(cv_char_vocab)
support = numpy.asarray(selector.get_support())
cv_char_vocab_10p = columns[support]

x_train_cv = x_train_cv.toarray()
x_test_cv = x_test_cv.toarray()

# produce a dataframe including the feature names
x_train_cv_char = pandas.DataFrame(x_train_cv, columns = cv_char_vocab)
x_test_cv_char = pandas.DataFrame(x_test_cv, columns=cv_char_vocab)
x_train_cv_char_10p = pandas.DataFrame(x_train_cv_10p, columns = cv_char_vocab_10p)
x_test_cv_char_10p = pandas.DataFrame(x_test_cv_10p, columns=cv_char_vocab_10p)
x_train_cv_char.head()
```

Out[14]:

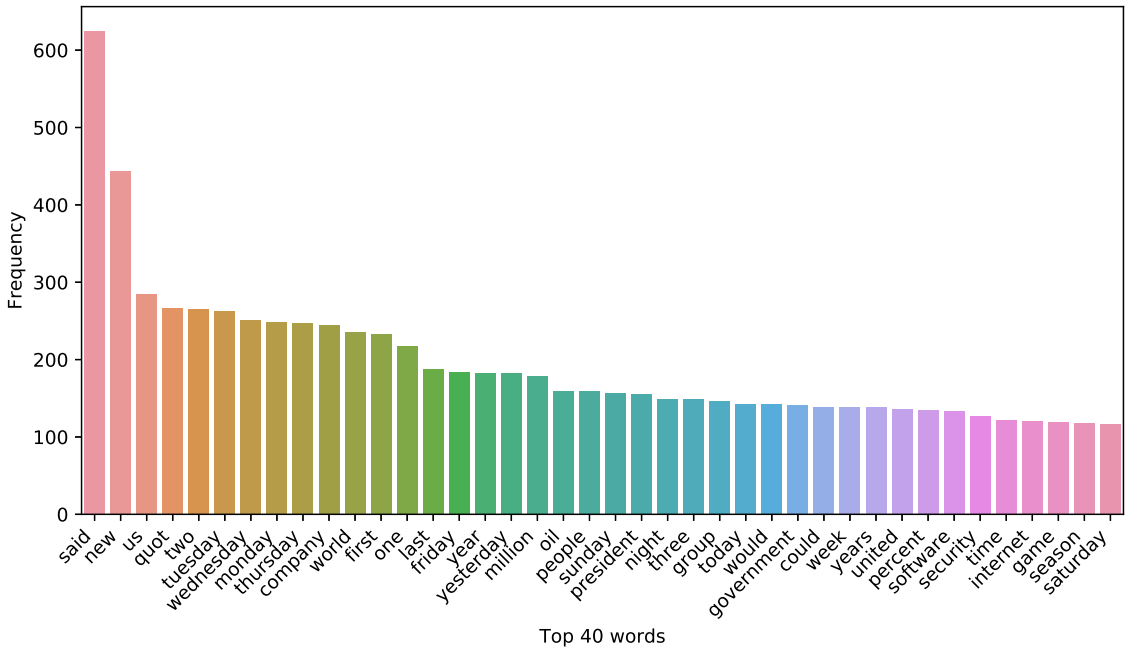
	a	aa	ab	ac	ad	ae	af	ag	ah	ai	...	zur	zv	zvo	zy	zy	zz	zz	zza	zzi	zzl
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	1	0	0	0	0	0
4	4	0	1	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 5834 columns

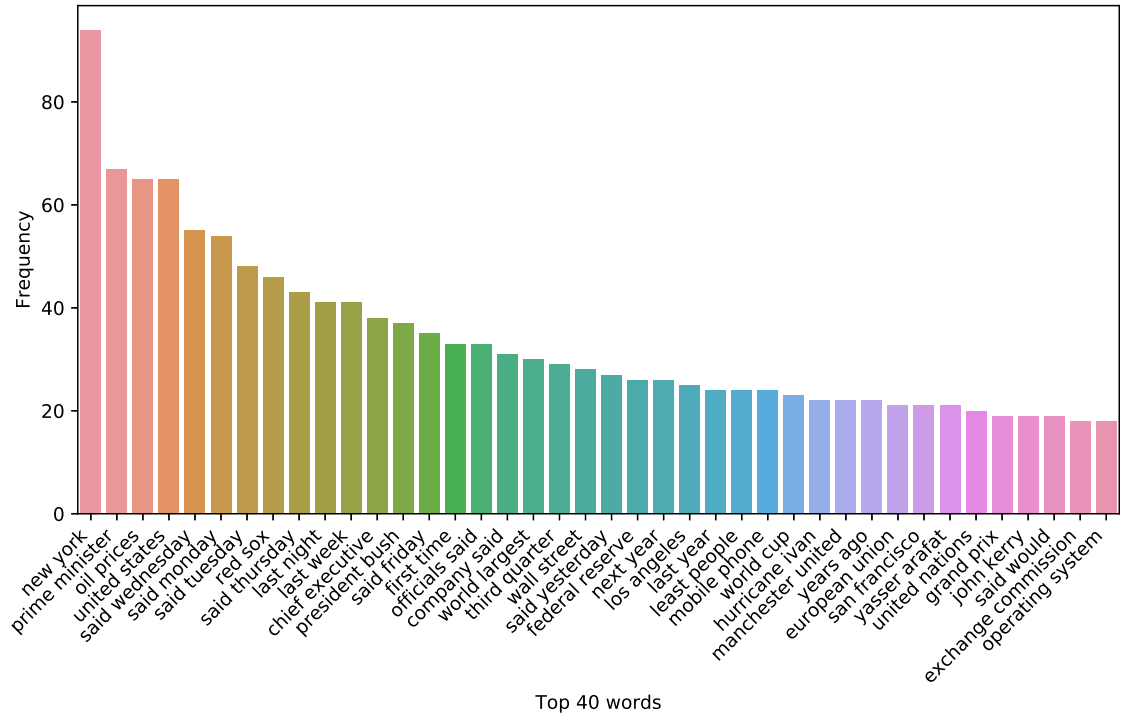
Let's explore the data we got through plots and tables

In [15]:
```
def words_barchart(df, df_label):
    word_count_dict = {}
    for word in df_label:
        word_count_dict[word] = int(sum(df.loc[:, word]))

    counter = Counter(word_count_dict)

    freq_df = pandas.DataFrame.from_records(counter.most_common(40),
                                            columns=['Top 40 words', 'Frequency'])

    plt.figure(figsize=(10,5))
    chart = sns.barplot(
        data=freq_df,
        x='Top 40 words',
        y='Frequency'
    )

    chart.set_xticklabels(
        chart.get_xticklabels(),
        rotation=45,
        horizontalalignment='right',
        fontweight='light'
    )
```

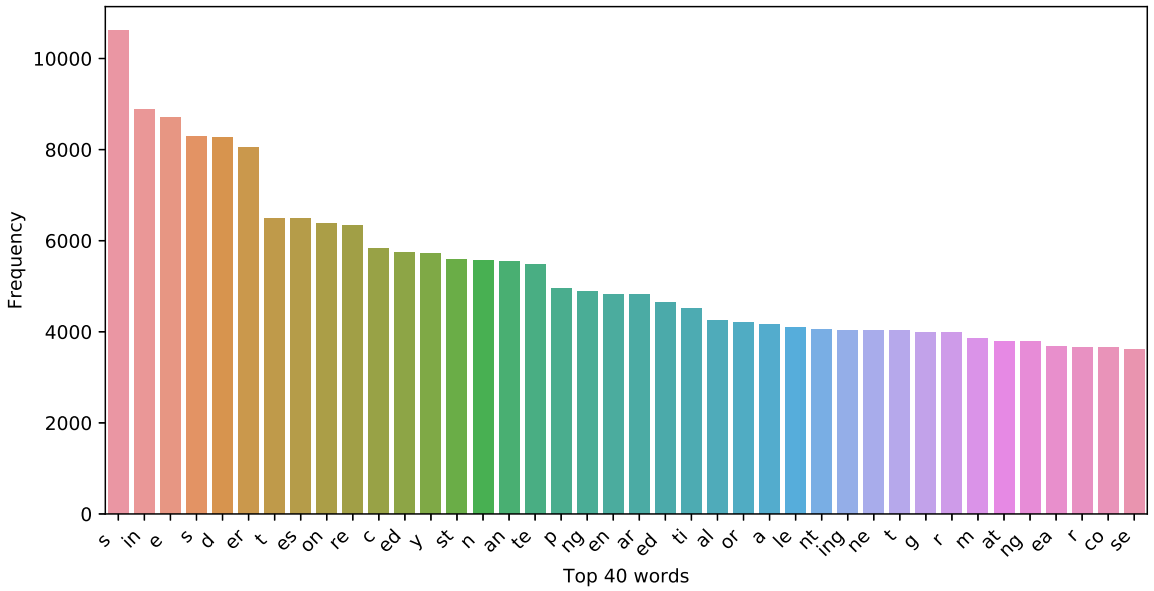
In [16]:
```
words_barchart(x_train_bagofwords, bow_vocab)
```



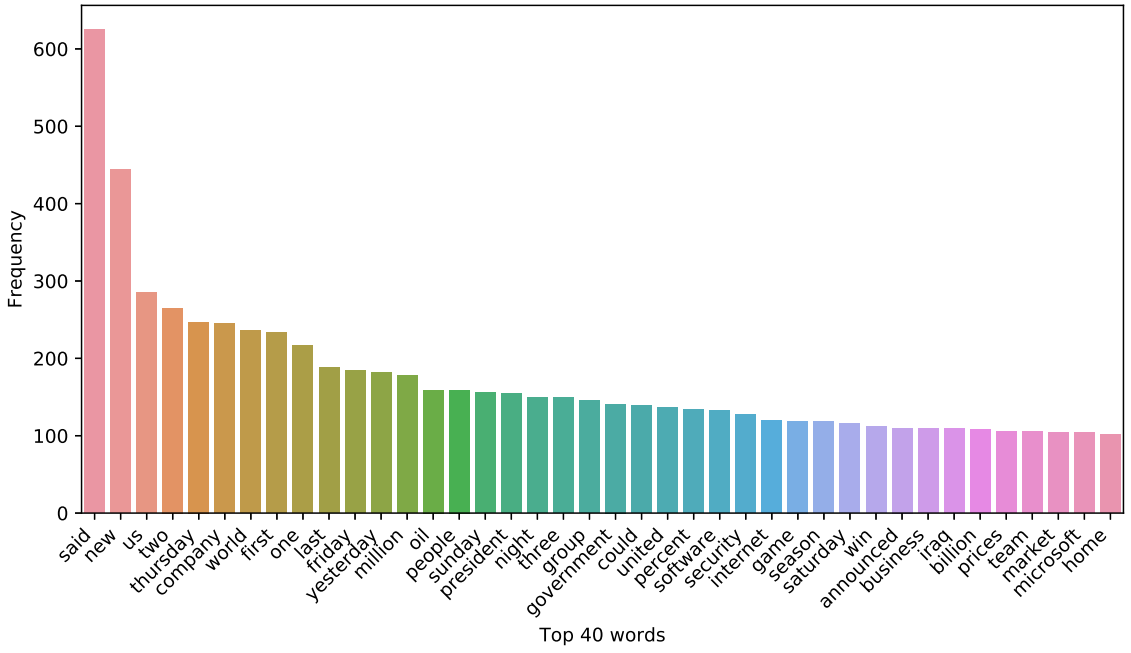
In [17]:
```
words_barchart(x_train_bagofngrams, ngram_vocab)
```



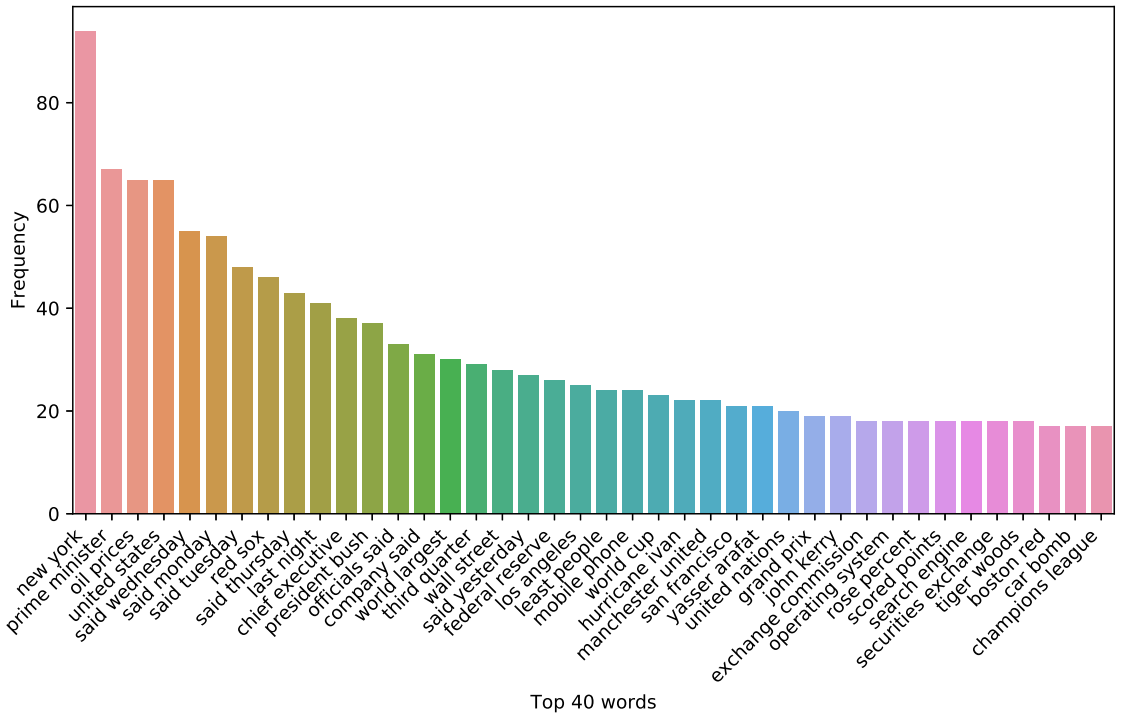
In [18]:
```
words_barchart(x_train_cv_char, cv_char_vocab)
```



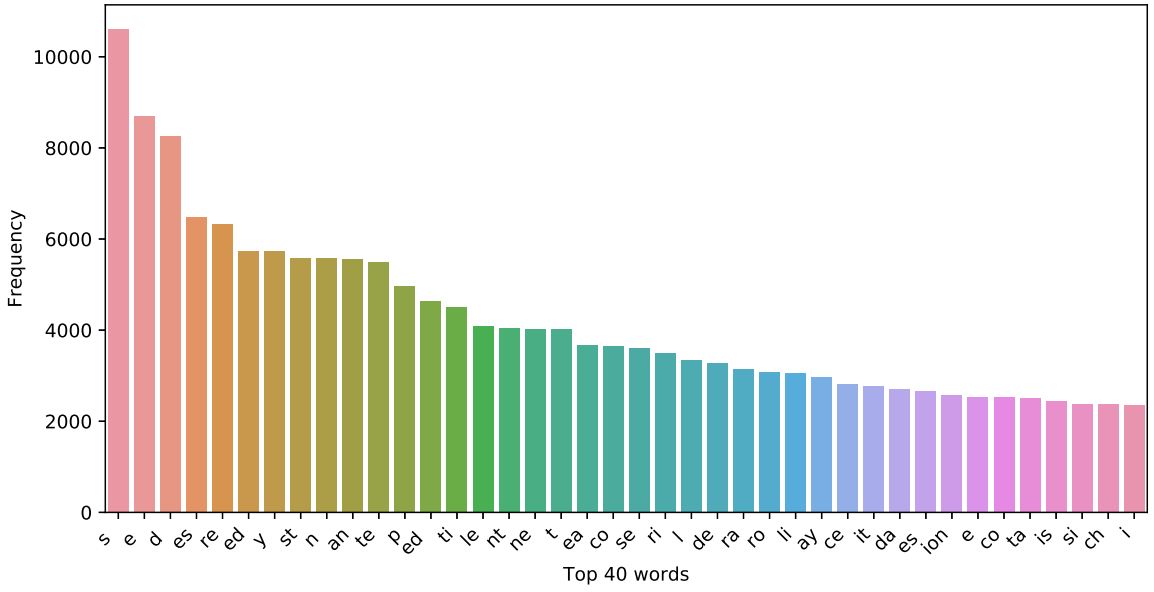
In [19]:
```
words_barchart(x_train_bagofwords_10p, bow_vocab_10p)
```



In [20]:
```
words_barchart(x_train_bagofngrams_10p, ngram_vocab_10p)
```



In [21]:
```
words_barchart(x_train_cv_char_10p, cv_char_vocab_10p)
```



TF/IDF

Unigram TF/IDF

In [22]:
```
# Use TF/IDF vectorizer to get a vector of unigrams
tfidf_vect = TfidfVectorizer(sublinear_tf = True, min_df = 2, ngram_range = (1, 1),
                             use_idf = True, token_pattern=r'\b[A-Za-z]{2,}\b')
x_train_tfidf_unigram = tfidf_vect.fit_transform(x_train).toarray()
x_test_tfidf_unigram = tfidf_vect.transform(x_test).toarray()

# get all unique words in the corpus
vocab = tfidf_vect.get_feature_names()

# produce a dataframe including the feature names
x_train_tfidf_unigram = pandas.DataFrame(numpy.round(x_train_tfidf_unigram, 2), columns = vocab)
x_test_tfidf_unigram = pandas.DataFrame(numpy.round(x_test_tfidf_unigram, 2), columns = vocab)
x_train_tfidf_unigram.head()
```

Out[22]:

	aaron	ab	abandon	abandoned	abandons	abbas	abc	abducted	abduction	abductions	...	zaragoza	zdnet	zealand	zee	zero	zimbabwe
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 6873 columns

N-Gram TF/IDF

Use TF/IDF vectorizer to get a vector of n-grams

In [23]:
```
tfidf_vect = TfidfVectorizer(sublinear_tf = True, min_df = 2, ngram_range = (2, 3),
                             use_idf = True, token_pattern=r'\b[A-Za-z]{2,}\b')
x_train_tfidf_ngram = tfidf_vect.fit_transform(x_train).toarray()
x_test_tfidf_ngram = tfidf_vect.transform(x_test).toarray()
# get all unique words in the corpus
vocab = tfidf_vect.get_feature_names()

# produce a dataframe including the feature names
x_train_tfidf_ngram = pandas.DataFrame(numpy.round(x_train_tfidf_ngram, 2), columns = vocab)
x_test_tfidf_ngram = pandas.DataFrame(numpy.round(x_test_tfidf_ngram, 2), columns = vocab)
x_train_tfidf_ngram.head()
```

Out[23]:

	ab billion	abducted militants	abductions foreigners	abductions foreigners iraq	aboard international	aboard international space	abu ghraib	abu ghraib prison	abu musab	ac milan	...	yukos said	yukos said would	zdnet survey	zdnet survey professionals	z b
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	

5 rows × 5929 columns

Character TF/IDF

Use TF/IDF vectorizer to get a vector of chars

In [24]:
```
tfidf_vect = TfidfVectorizer(analyzer = 'char', sublinear_tf = True, min_df = 2,
                             ngram_range = (2, 3), use_idf = True,
                             token_pattern=r'\b[A-Za-z]{2,}\b')
x_train_tfidf_char = tfidf_vect.fit_transform(x_train).toarray()
x_test_tfidf_char = tfidf_vect.transform(x_test).toarray()

# get all unique words in the corpus
char_vocab = tfidf_vect.get_feature_names()

# produce a dataframe including the feature names
x_train_tfidf_char = pandas.DataFrame(numpy.round(x_train_tfidf_char, 2), columns = char_vocab)
x_test_tfidf_char = pandas.DataFrame(numpy.round(x_test_tfidf_char, 2), columns = char_vocab)
x_train_tfidf_char.head()
```

Out[24]:

	a	aa	ab	ac	ad	ae	af	ag	ah	ai	...	zur	zv	zvo	zy	zy	zz	zz	zza	zzi	zzl
0	0.00	0.0	0.00	0.00	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.00	0.00	0.0	0.0	0.0	0.0	0.0
1	0.03	0.0	0.00	0.07	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.00	0.00	0.0	0.0	0.0	0.0	0.0
2	0.03	0.0	0.00	0.00	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.00	0.00	0.0	0.0	0.0	0.0	0.0
3	0.00	0.0	0.00	0.00	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.13	0.14	0.0	0.0	0.0	0.0	0.0
4	0.06	0.0	0.07	0.00	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.00	0.00	0.0	0.0	0.0	0.0	0.0

5 rows × 5834 columns

Using gensim to build Word2Vec

In [25]:
```
from gensim.models import word2vec

# tokenize sentences in corpus
wpt = nltk.WordPunctTokenizer()
tokenized_corpus_train = [wpt.tokenize(document) for document in x_train]
tokenized_corpus_test = [wpt.tokenize(document) for document in x_test]

# Set values for various parameters
feature_size = 4000    # Word vector dimensionality
window_context = 20          # Context window size
workers = 12
min_word_count = 5   # Minimum word count
sample = 1e-3   # Downsample setting for frequent words

w2v_model_train = word2vec.Word2Vec(tokenized_corpus_train, size=feature_size,
                          window=window_context, min_count=min_word_count,
                          sample=sample, iter=50)
w2v_model_test = word2vec.Word2Vec(tokenized_corpus_test, size=feature_size,
                          window=window_context, min_count=min_word_count,
                          sample=sample, iter=50)
```

Functions to get document level embeddings

The idea is to distill a word vector of 'n' features into a single point and use that at a document level

In [27]:
```
def average_word_vectors(words, model, vocabulary, num_features):

 feature_vector = numpy.zeros((num_features,),dtype="float64")
 nwords = 0.

 for word in words:
    if word in vocabulary:
      nwords = nwords + 1.
      feature_vector = numpy.add(feature_vector, model[word])

 if nwords:
    feature_vector = numpy.divide(feature_vector, nwords)

 return feature_vector


def averaged_word_vectorizer(corpus, model, num_features):
 vocabulary = set(model.wv.index2word)
 features = [average_word_vectors(tokenized_sentence, model, vocabulary, num_features)
             for tokenized_sentence in corpus]
 return numpy.array(features)
```

Obtain document level embeddings

In [28]:
```
w2v_feature_array_train = averaged_word_vectorizer(corpus=tokenized_corpus_train, model=w2v_model_train,
                                                   num_features=feature_size)
w2v_feature_array_test = averaged_word_vectorizer(corpus=tokenized_corpus_test, model=w2v_model_test,
                                                  num_features=feature_size)
x_train_w2v = pandas.DataFrame(w2v_feature_array_train)
x_test_w2v = pandas.DataFrame(w2v_feature_array_test)
```

In [29]:
```
x_train_w2v.head()
```

Out[29]:

	0	1	2	3	4	5	6	7	8	9	...	3990	3991	3992	
0	-0.047792	-0.010078	-0.047501	0.065156	0.040363	0.053578	-0.078255	-0.072195	-0.048381	0.004039	...	0.018117	0.001505	0.024522	(
1	-0.027499	-0.007007	-0.011028	0.007280	0.062373	-0.033475	-0.064167	-0.063685	-0.071467	-0.020646	...	-0.004596	0.036393	-0.014121	-
2	-0.033255	-0.047467	-0.039281	0.084740	0.031803	-0.025014	-0.062031	-0.059123	-0.036834	-0.017421	...	0.015355	-0.048778	0.069986	(
3	0.037945	0.031579	-0.041542	-0.051591	0.080797	-0.051425	0.027580	-0.023383	0.061208	-0.056261	...	0.007027	0.010646	-0.012574	(
4	0.064920	0.018165	-0.029505	0.007345	0.056818	-0.126387	-0.068517	-0.032791	-0.055429	0.016335	...	-0.033108	-0.024404	0.020513	-(

5 rows × 4000 columns

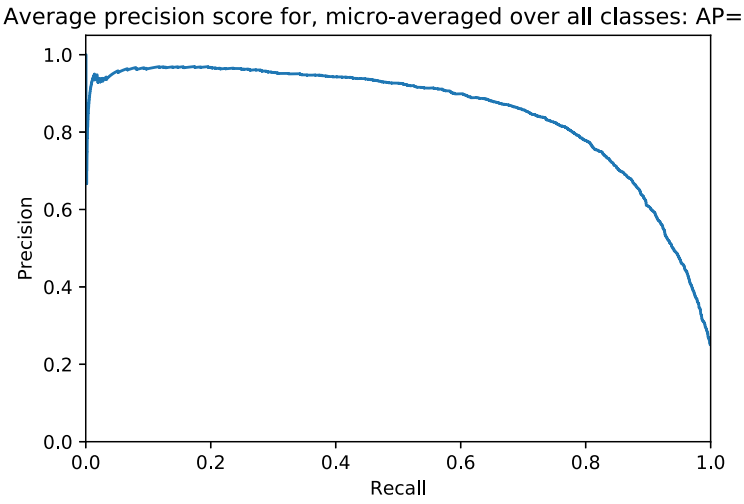
Perform SVM as a baseline model and evaluate it.

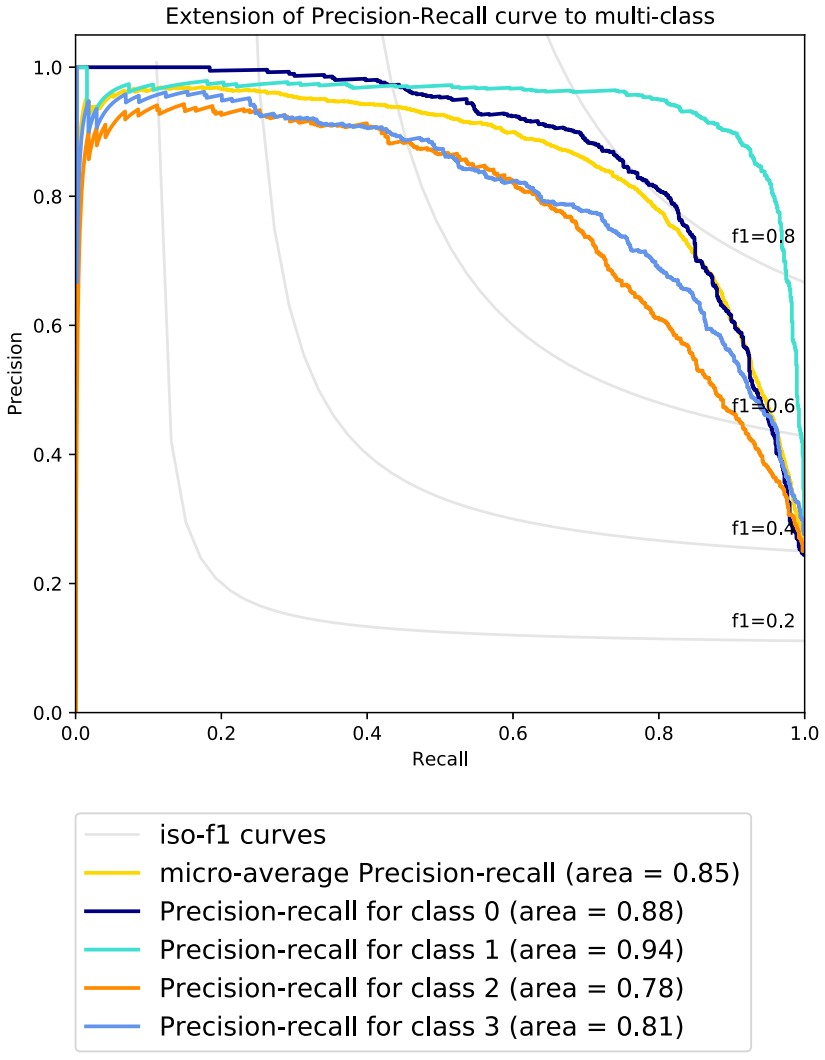
In [30]:
```
# SVM classifier and plot superfunction
def run_svm(x_train, y_train, x_test, emb):
    str(emb)
    classifier = OneVsRestClassifier(svm.LinearSVC(random_state=1))
    classifier.fit(x_train, y_train)
    y_score = classifier.decision_function(x_test)


    # The average precision score in multi-label settings
    # For each class
    precision = dict()
    recall = dict()
    average_precision = dict()
    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])

    # A "micro-average": quantifying score on all classes jointly
    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")
    print(emb)
    print('Average precision score, micro-averaged over all classes: {0:0.2f}'
        .format(average_precision["micro"]))

    # Plot the micro-averaged Precision-Recall curve
    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 for, micro-averaged over all classes: AP={0:0.2f}'
        .format(average_precision["micro"]))

    # Plot Precision-Recall curve for each class and iso-f1 curves
    # setup plot details
    colors = cycle(['navy', 'turquoise', 'darkorange', 'cornflowerblue', 'teal'])

    plt.figure(figsize=(7, 8))
    f_scores = numpy.linspace(0.2, 0.8, num=4)
    lines = []
    labels = []
    for f_score in f_scores:
        x = numpy.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["micro"]))

    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, -.5), prop=dict(size=14))

    plt.show()
```

SVM for Bag of Words

In [31]:
```
run_svm(x_train_bagofwords, y_train, x_test_bagofwords, 'Bag of Words')
```

```
Bag of Words
Average precision score, micro-averaged over all classes: 0.85
```

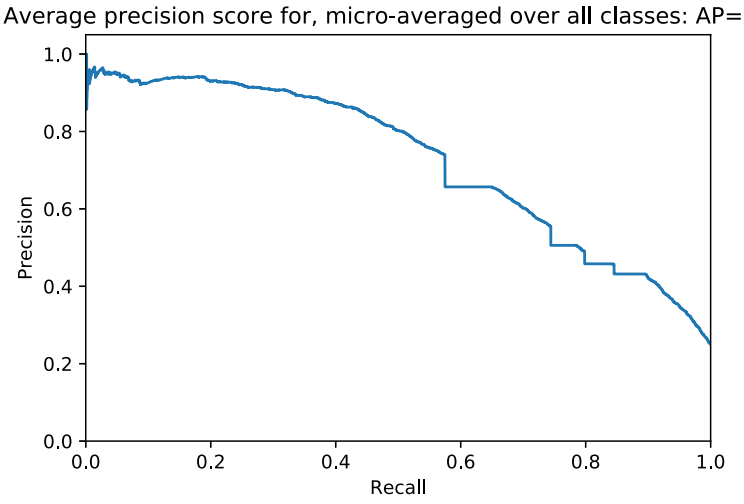


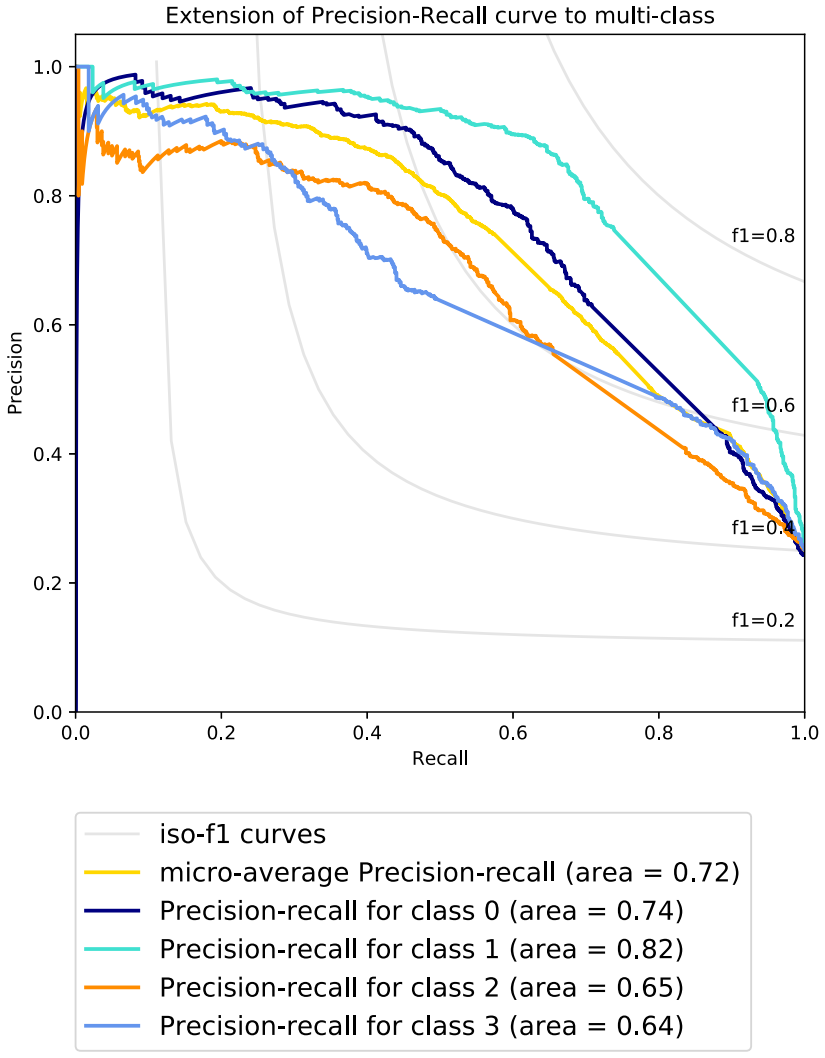


SVM for Bag of N-grams

In [32]:
```
run_svm(x_train_bagofngrams, y_train, x_test_bagofngrams, 'Bag of N-Grams')
```

```
Bag of N-Grams
Average precision score, micro-averaged over all classes: 0.72
```

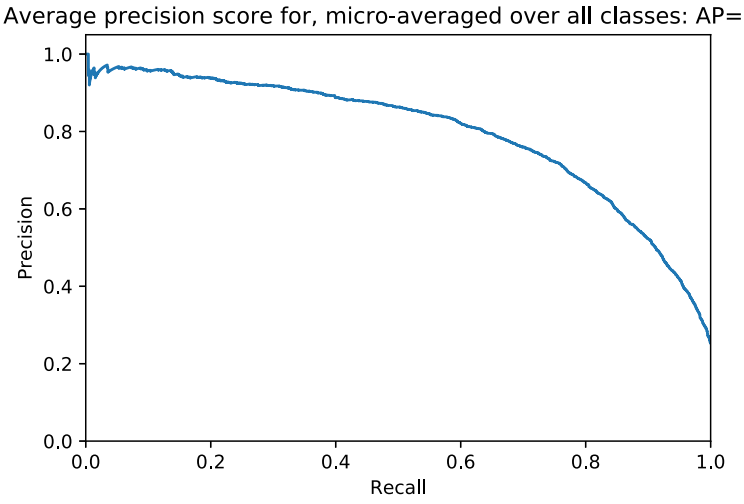


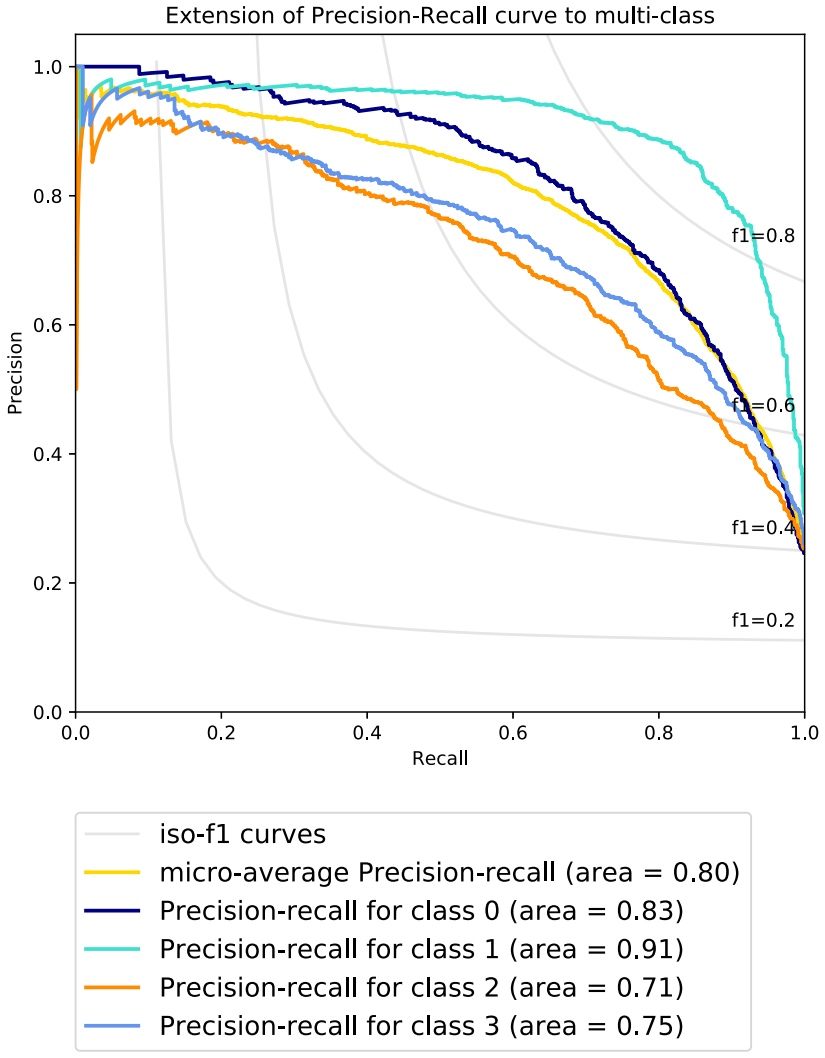


SVM for Bag of Chars

In [33]:
```
run_svm(x_train_cv_char, y_train, x_test_cv_char, 'Bag of Chars')
```

```
Bag of Chars
Average precision score, micro-averaged over all classes: 0.80
```

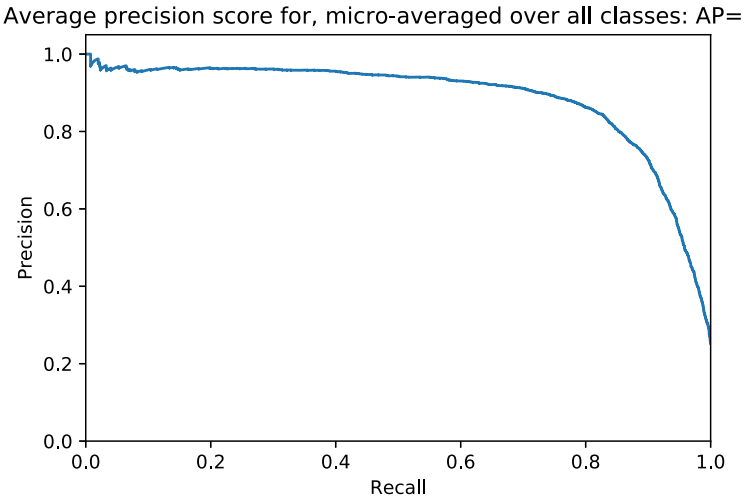


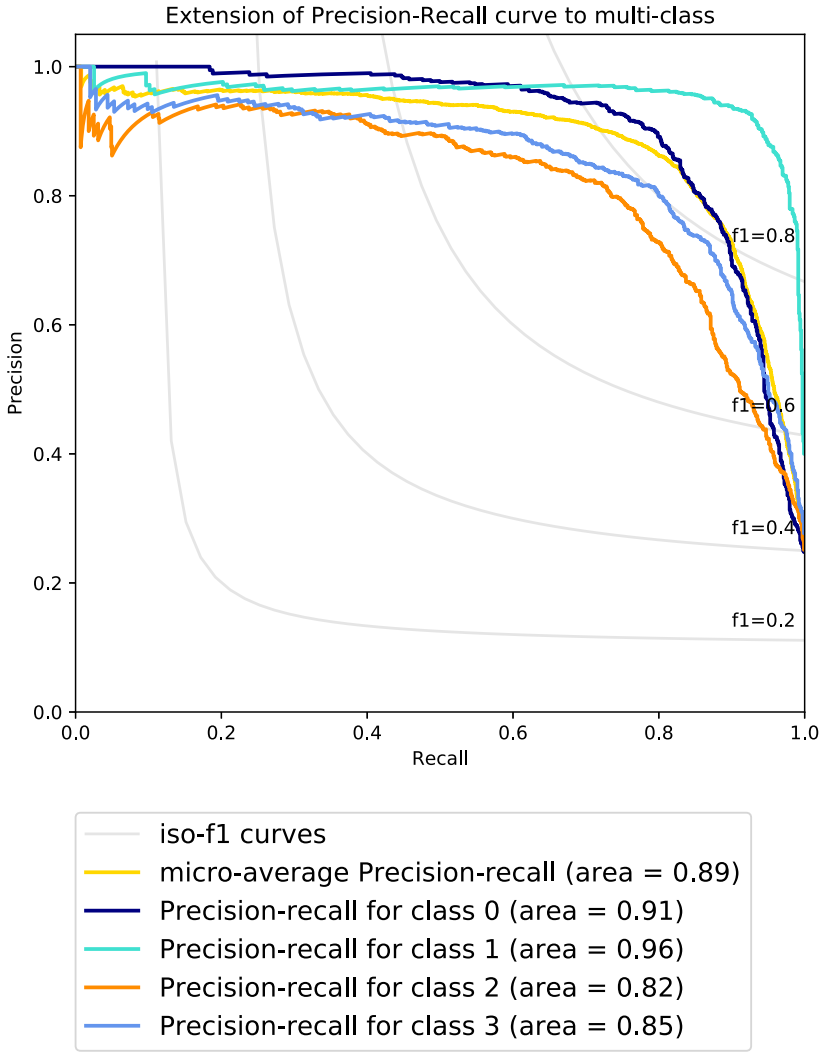


SVM for TF/IDF Unigram

In [34]:
```
run_svm(x_train_tfidf_unigram, y_train, x_test_tfidf_unigram, 'TF/IDF Unigram')
```

```
TF/IDF Unigram
Average precision score, micro-averaged over all classes: 0.89
```

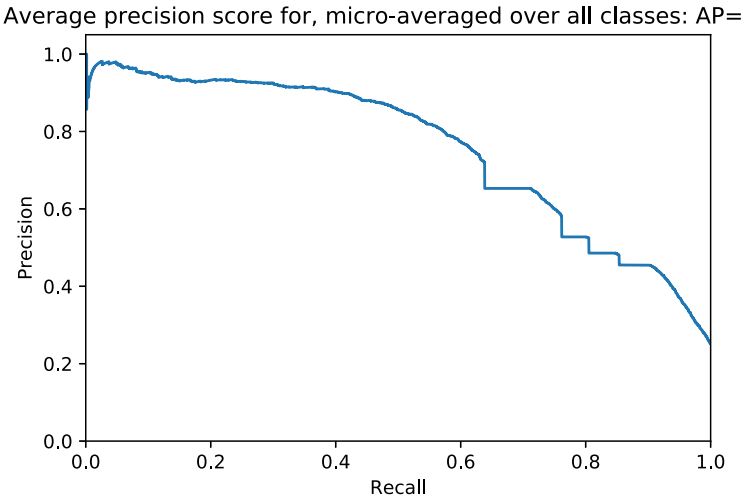


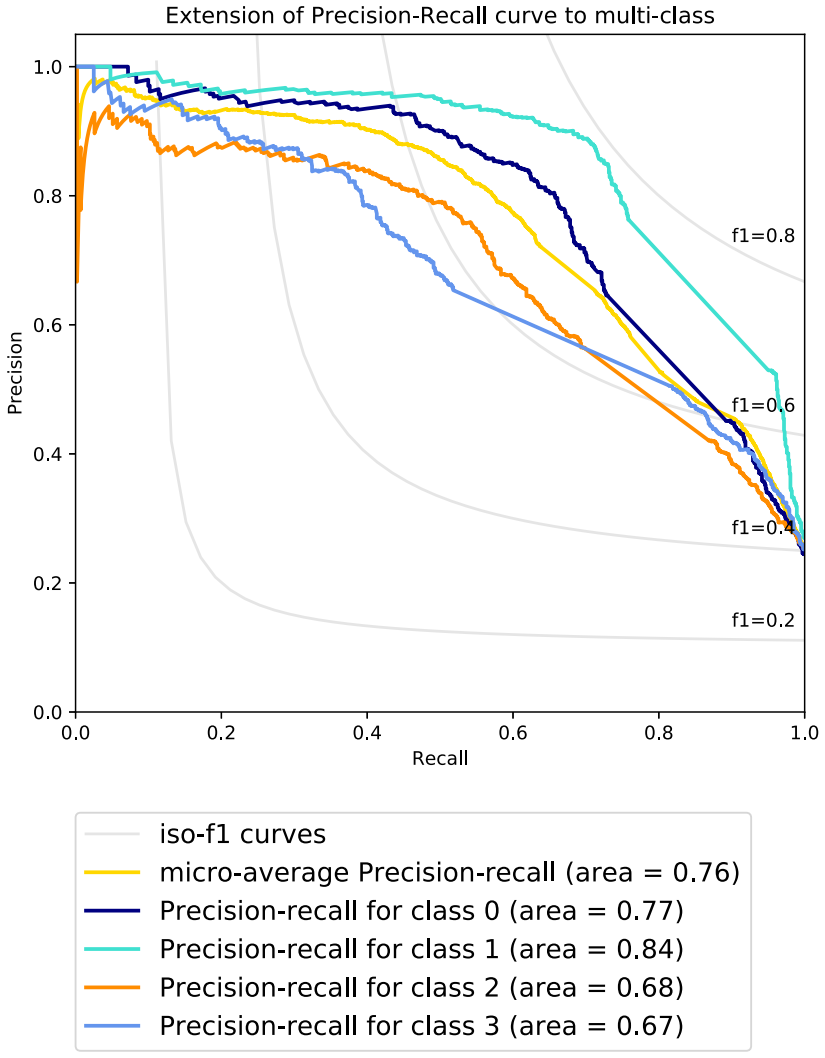


SVM for TF/IDF N-grams

In [35]:
```
run_svm(x_train_tfidf_ngram, y_train, x_test_tfidf_ngram, 'TF/IDF N-Grams')
```

```
TF/IDF N-Grams
Average precision score, micro-averaged over all classes: 0.76
```

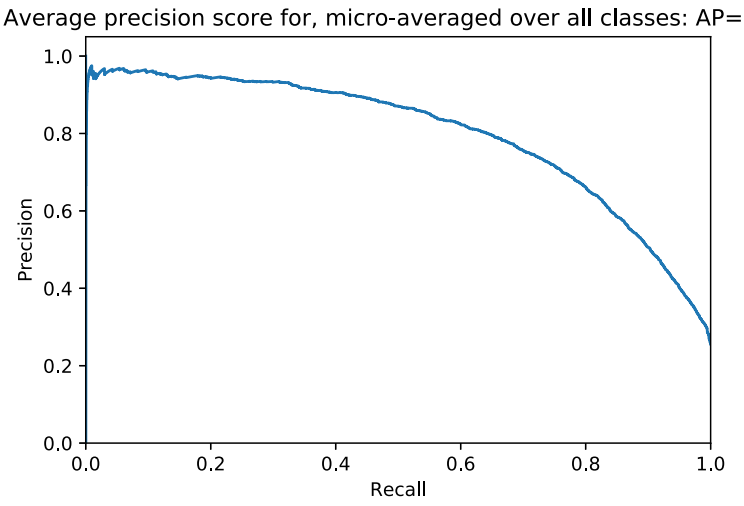


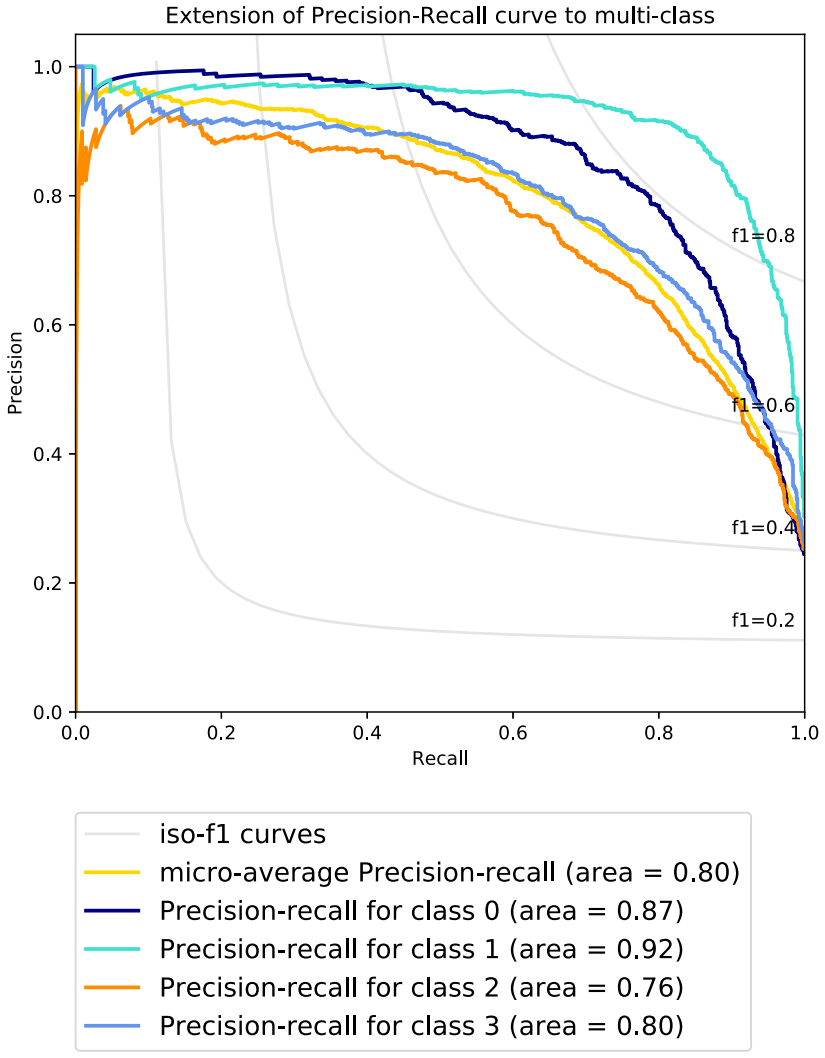


SVM for TF/IDF Chars

In [36]:
```
run_svm(x_train_tfidf_char, y_train, x_test_cv_char, 'TF/IDF Chars')
```

```
TF/IDF Chars
Average precision score, micro-averaged over all classes: 0.80
```

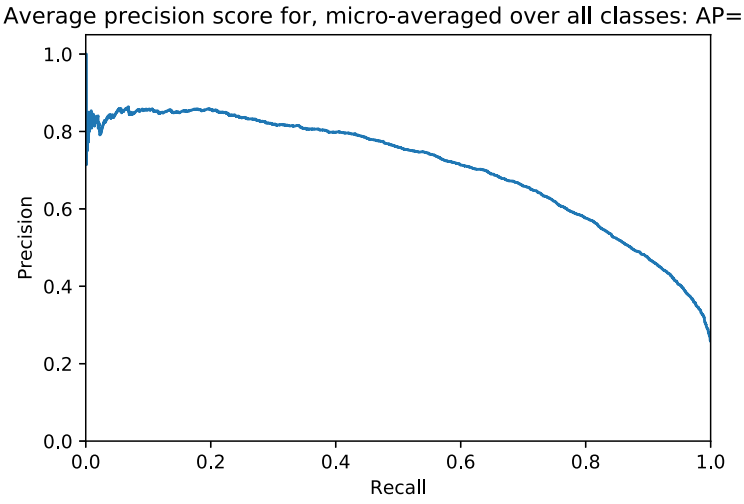


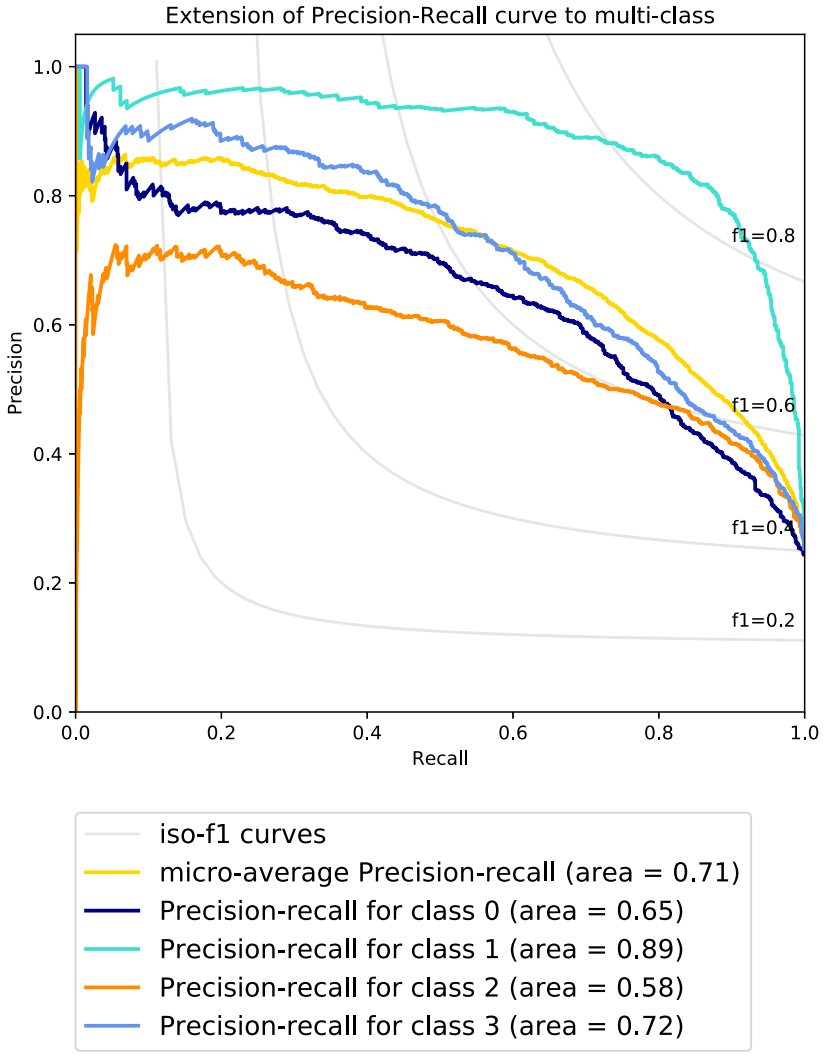


SVM for Word2Vec

In [37]:
```
run_svm(x_train_w2v, y_train, x_test_w2v, 'Word2Vec')
```

```
Word2Vec
Average precision score, micro-averaged over all classes: 0.71
```



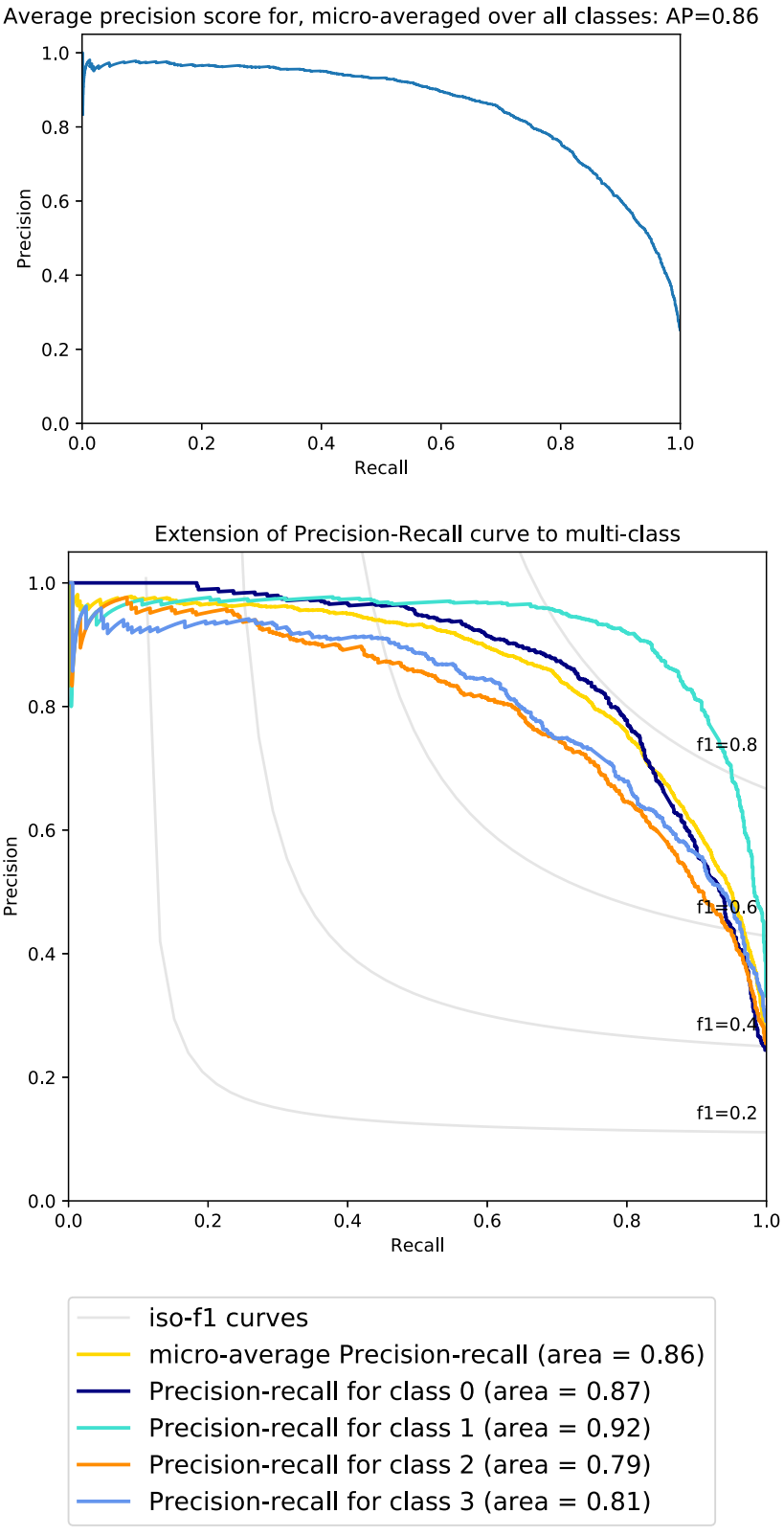


Let's explore also the SVM performance on 90th percentile feature selection

SVM for Bag of Words 90th percentile

In [38]:
```
run_svm(x_train_bagofwords_10p, y_train, x_test_bagofwords_10p, 'Bag of Words - 90th percentile')
```

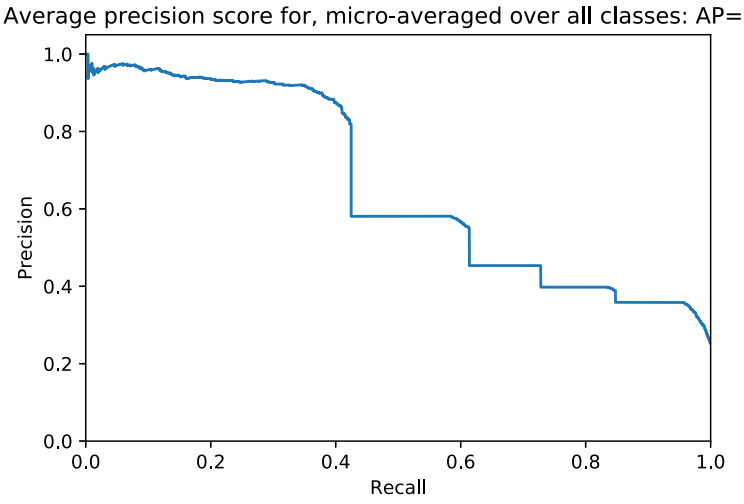
```
Bag of Words - 90th percentile
Average precision score, micro-averaged over all classes: 0.86
```

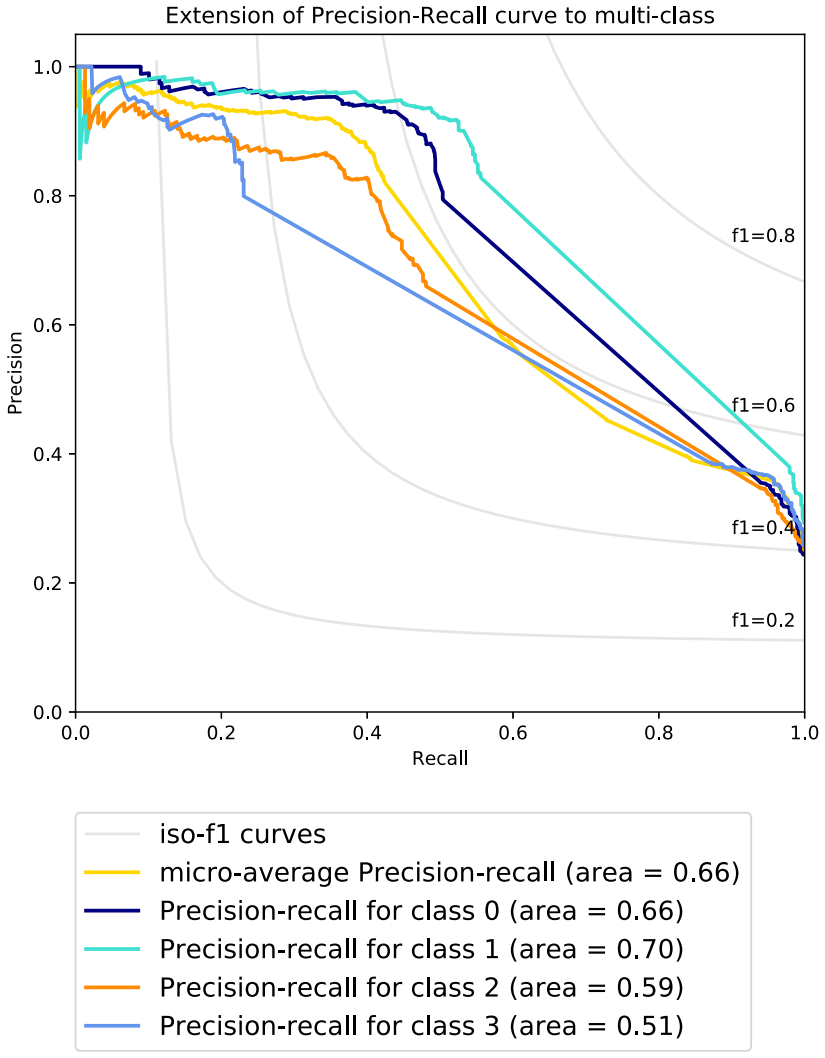


SVM for Bag of N-grams 90th percentile

In [39]:
```
run_svm(x_train_bagofngrams_10p, y_train, x_test_bagofngrams_10p, 'Bag of N-Grams - 90th percentile')
```

```
Bag of N-Grams - 90th percentile
Average precision score, micro-averaged over all classes: 0.66
```

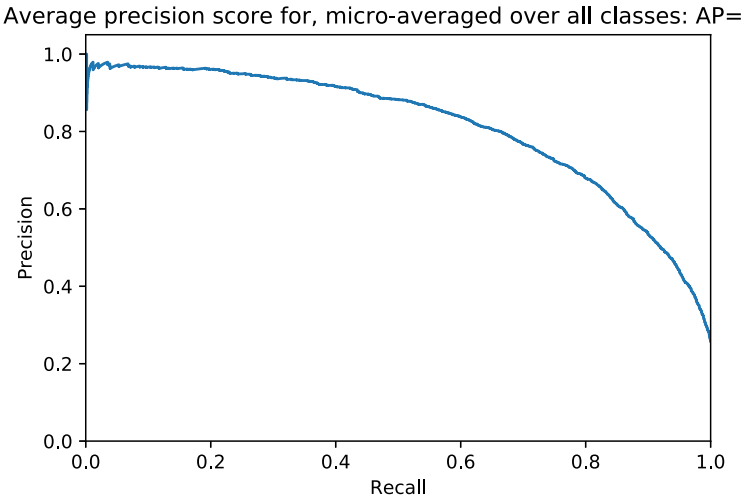


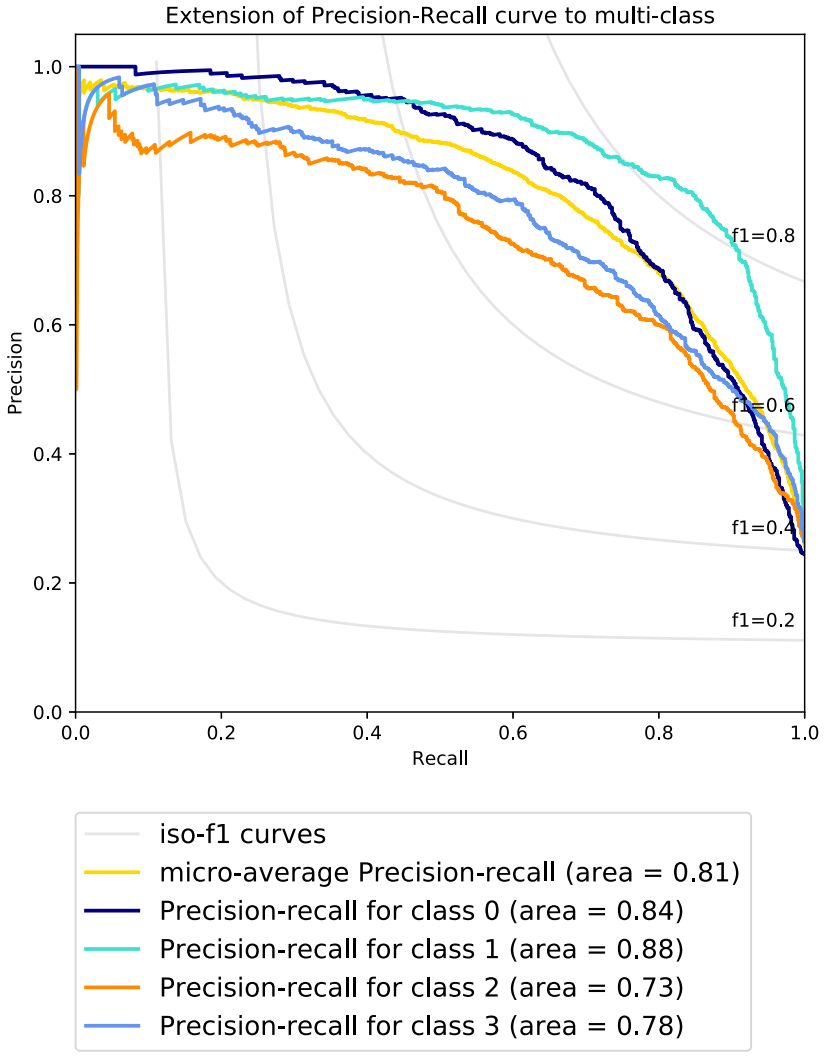


SVM for Bag of Chars 90th percentile

In [40]:
```
run_svm(x_train_cv_char_10p, y_train, x_test_cv_char_10p, 'Bag of Chars - 90th percentile')
```

```
Bag of Chars - 90th percentile
Average precision score, micro-averaged over all classes: 0.81
```





References - Code sample sources disclaimer:

Code for this project is either directly from (with some modification), or inspired by, but not limited to the following sources:

- Kelly Epley Naive Bayes: https://towardsdatascience.com/naive-bayes-document-classification-in-python-e33ff50f937e (https://towardsdatascience.com/naive-bayes-document-classification-in-python-e33ff50f937e)
- MLWhiz's excellent blogs about text classification and NLP: https://mlwhiz.com/blog/2018/12/17/text_classification/ (https://mlwhiz.com/blog/2018/12/17/text_classification/) https://mlwhiz.com/blog/2019/01/17/deeplearning_nlp_preprocess/ (https://mlwhiz.com/blog/2019/01/17/deeplearning_nlp_preprocess/) https://mlwhiz.com/blog/2019/02/08/deeplearning_nlp_conventional_methods/ (https://mlwhiz.com/blog/2019/02/08/deeplearning_nlp_conventional_methods/) https://www.kaggle.com/mlwhiz/conventional-methods-for-quora-classification/ (https://www.kaggle.com/mlwhiz/conventional-methods-for-quora-classification/)
- Christof Henkel preprocessing: https://www.kaggle.com/christofhenkel/how-to-preprocessing-when-using-embeddings (https://www.kaggle.com/christofhenkel/how-to-preprocessing-when-using-embeddings)
- datanizing GmbH: https://medium.com/@datanizing/modern-text-mining-with-python-part-1-of-5-introduction-cleaning-and-linguistics-647f9ec85b6a (https://medium.com/@datanizing/modern-text-mining-with-python-part-1-of-5-introduction-cleaning-and-linguistics-647f9ec85b6a)
- Datacamp wordcloud: https://www.datacamp.com/community/tutorials/wordcloud-python (https://www.datacamp.com/community/tutorials/wordcloud-python)
- Seaborn Pydata tutorials: https://seaborn.pydata.org/introduction.html#intro-plot-customization (https://seaborn.pydata.org/introduction.html#intro-plot-customization)
- Dipanjan S's tutorials: https://github.com/dipanjanS (https://github.com/dipanjanS)
- Analytics Vidhya: https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/ (https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/)
- Jason Brownlee's Feature Selection For Machine Learning in Python https://machinelearningmastery.com/feature-selection-machine-learning-python/ (https://machinelearningmastery.com/feature-selection-machine-learning-python/)
- Susan Li's Multi-class text classification with Scikit-learn: https://towardsdatascience.com/multi-class-text-classification-with-scikit-learn-12f1e60e0a9f (https://towardsdatascience.com/multi-class-text-classification-with-scikit-learn-12f1e60e0a9f)
- Vadim Smolyakov Ensemble Learning to Improve Machine Learning Results: https://blog.statsbot.co/ensemble-learning-d1dcd548e936 (https://blog.statsbot.co/ensemble-learning-d1dcd548e936)
- Udacity course video on Youtube UD120: https://www.youtube.com/watch?v=GdsLRKjjKLw (https://www.youtube.com/watch?v=GdsLRKjjKLw)

Models and Ensembling Methods + Interpretability with LIME

Import dependencies

In [1]:
```
import numpy
from gensim.models import word2vec
from gensim.models import KeyedVectors
import pandas
from nltk import WordPunctTokenizer
from sklearn.preprocessing import label_binarize
import sqlite3
from sklearn.multiclass import OneVsRestClassifier
from matplotlib import pyplot as plt
import seaborn as sns
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import average_precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import matthews_corrcoef
from sklearn.metrics import precision_recall_fscore_support
from sklearn import svm
from itertools import cycle
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_validate
from sklearn.metrics import precision_score, recall_score, roc_auc_score
from sklearn.metrics import multilabel_confusion_matrix, confusion_matrix
from sklearn.metrics import make_scorer
from sklearn.ensemble import StackingClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn import tree
from sklearn.model_selection import GridSearchCV
from mlxtend.plotting import plot_learning_curves
import lime
import lime.lime_tabular
import lime.lime_text
```

Define Constants

In [2]:
```
W2V_FEATURE_SIZE = 300
N_CLASSES = 4
RANDOM_STATE = 123
N_FOLDS = 5
```

Read in the data

Load raw train and test data

Load in the data from the database

In [3]:
```
dbconn = sqlite3.connect('./data/cleanedtraintest_v2.db')
train_data_df = pandas.read_sql_query(
    'SELECT category, content_cleaned FROM train_data', dbconn)
test_data_df = pandas.read_sql_query(
    'SELECT category, content_cleaned FROM test_data', dbconn)
dbconn.commit()
dbconn.close()
```

Check the if the data was loaded correctly

In [4]:
```
train_data_df
```

Out[4]:

	category	content_cleaned
0	3	wall street seeing green
1	3	private investment firm carlyle group reputati...
2	3	soaring crude prices plus economy outlook earn...
3	3	authorities halted oil main pipeline southern ...
4	3	tearaway world oil prices toppling records str...
...	...	...
119995	1	pakistani president pervez musharraf said stay...
119996	2	red sox general manager theo epstein acknowled...
119997	2	miami dolphins put courtship lsu coach nick sa...
119998	2	pittsburgh ny giants time line steelers record...
119999	2	vince carter traded toronto raptors new jersey...

120000 rows × 2 columns

In [5]:
```
test_data_df
```

Out[5]:

	category	content_cleaned
0	3	unions representing workers turner newall say ...
1	4	toronto canada rocketeers competing million an...
2	4	company founded chemistry researcher universit...
3	4	barely dawn mike fitzpatrick starts shift blur...
4	4	southern california agency went emissions bovi...
...	...	...
7595	1	ukrainian presidential candidate viktor yushch...
7596	2	supply attractive pitching options dwindling d...
7597	2	like roger clemens almost exactly eight years ...
7598	3	singapore doctors united states warned painkil...
7599	3	ebay plans buy apartment home rental service m...

7600 rows × 2 columns

Train & Test data where x is the predictor features, y is the predicted feature

In [6]:
```
x_train = train_data_df.content_cleaned
y_train = label_binarize(train_data_df.category, classes=range(1, N_CLASSES +
1))

x_test = test_data_df.content_cleaned
y_test = label_binarize(test_data_df.category, classes=range(1, N_CLASSES + 1
))
```

Load word2vec data

Load word2vec feature arrays from .npz files

load dict of arrays

In [7]:
```
w2v_train_features_array_dict = numpy.load(
    './data/word2vec-train-features-120000-min5dim300.npz')
w2v_test_features_array_dict = numpy.load(
    './data/word2vec-test-features-120000-min5dim300.npz')
# extract the first array from train
data = w2v_train_features_array_dict['arr_0']
# print the array
print(data)
# extract the first array from test
data = w2v_test_features_array_dict['arr_0']
# print the array
print(data)
```

```
[[-0.43092448  0.50092196  0.08331972 ...  1.3914201   1.2953259
  -1.8574607 ]
 [-0.10783155 -0.35169265  0.90062636 ... -0.38979718  0.13664657
   0.5066641 ]
 [-1.0086536  -0.29255652 -0.7550053  ... -0.18521406  0.7896786
  -0.23576818]
 ...
 [-0.02566049  0.23409443 -0.8595321  ... -0.05427613 -0.89297265
  -0.09055152]
 [-0.6081659   0.42683512 -0.9105423  ... -0.06156884 -0.40654626
   0.07195716]
 [-1.0819023  -0.04211196 -0.16453283 ... -0.40625843 -0.13644677
  -0.0066904 ]]
[[-0.02657197 -1.0014614  -0.035705   ...  0.48677683  0.3947945
  -0.9894788 ]
 [-0.54866743 -1.3801866   0.66031504 ... -0.4012159   0.6803215
   0.92033225]
 [ 0.11171789  0.3781767  -0.26057357 ... -0.5006595   0.13674003
   0.10530389]
 ...
 [-0.46190766  0.7501185  -0.20256642 ... -0.32613838  0.09363924
   0.46578252]
 [-0.023529   -0.33200815 -0.63418424 ... -0.46149412  0.39634904
  -0.46027517]
 [-0.25388533 -0.6177681   0.9628809  ... -0.66557425 -0.1068292
  -0.64577085]]
```

Load word2vec model trained key vectors

In [8]:
```
w2v_model_train = KeyedVectors.load(
    './data/custom-trained-word2vec-120000-min5dim300.kv')
```

Get the word2vec data back into usable form

In [9]:
```
wpt = WordPunctTokenizer()
tokenized_corpus_train = [wpt.tokenize(document) for document in x_train]
tokenized_corpus_test = [wpt.tokenize(document) for document in x_test]

def average_word_vectors(words, model, vocabulary, num_features):
    feature_vector = numpy.zeros((num_features,), dtype="float32")
    nwords = 0.

    for word in words:
        if word in vocabulary:
            nwords = nwords + 1.
            feature_vector = numpy.add(feature_vector, model[word])

    if nwords:
        feature_vector = numpy.divide(feature_vector, nwords)

    return feature_vector

def averaged_word_vectorizer(corpus, model, num_features):
    vocabulary = set(model.wv.index2word)
    features = [average_word_vectors(tokenized_sentence, model, vocabulary, nu
m_features)
            for tokenized_sentence in corpus]
    return numpy.array(features)
```

Obtain document level embeddings

In [10]:
```
w2v_feature_array_train = averaged_word_vectorizer(corpus=tokenized_corpus_tra
in,
    model=w2v_model_train, num_features=W2V_FEATURE_SIZE)
w2v_feature_array_test = averaged_word_vectorizer(corpus=tokenized_corpus_test
,
    model=w2v_model_train, num_features=W2V_FEATURE_SIZE)

x_train_w2v = pandas.DataFrame(w2v_feature_array_train)
x_test_w2v = pandas.DataFrame(w2v_feature_array_test)
```

Sample down for speed, for now. (use when testing)

x_train_w2v = x_train_w2v.sample(n = 3000, replace = False, random_state = RANDOM_STATE) y_train = train_data_df.category.sample(n = 3000, replace = False, random_state = RANDOM_STATE) y_train = label_binarize(y_train, classes=range(1, N_CLASSES + 1))

Build Models

SVM Model Building Function

In [11]:
```
def run_svm(x_train, y_train):
    classifier = OneVsRestClassifier(svm.LinearSVC(random_state=RANDOM_STATE))
    classifier.fit(x_train, y_train)
    return classifier
```

Logistic Regression Model Building Function

In [12]:
```
def run_logreg(x_train, y_train):
    classifier = OneVsRestClassifier(LogisticRegression(random_state=RANDOM_ST
ATE))
    classifier.fit(x_train, y_train)
    return classifier
```

Naive Bayes Function

In [13]:
```
def run_nb(x_train, y_train):
    classifier = OneVsRestClassifier(GaussianNB())
    classifier.fit(x_train, y_train)
    return classifier
```

Decision Trees Function

In [14]:
```
def run_dectree(x_train, y_train):
    classifier = OneVsRestClassifier(tree.DecisionTreeClassifier())
    classifier.fit(x_train, y_train)
    return classifier
```

Functions to calculate scores and to plot them

Calculate, then plot the Precision, Recall, Average Precision, F1

In [15]:
```
def prf1_calc(classifier, algo_name, n_classes, x_test, y_test):
    # Get the decision function from the classifier
    if algo_name == 'SVM':
        y_score = classifier.decision_function(x_test)
    else:
        y_score = classifier.predict_proba(x_test)
    y_pred = classifier.predict(x_test)

    # The average precision score in multi-label settings
    # For each class
    precision = dict()
    recall = dict()
    average_f1 = dict()
    average_precision = dict()
    mcc = dict()
    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])
        average_f1[i] = f1_score(y_test[:, i], y_pred[:, i])
        mcc[i] = matthews_corrcoef(y_test[:, i], y_pred[:, i])

    # A "micro-average": quantifying score on all classes jointly
    precision["micro"], recall["micro"], _ = precision_recall_curve(y_test.rav
el(),
        y_score.ravel())
    average_precision["micro"] = average_precision_score(y_test, y_score,
                                                         average="micro")
    average_f1['micro'] = f1_score(y_test, y_pred, average='micro')
    mcc['micro'] = sum(mcc.values())/4

    # Plot the data
    prf1_plot(precision, recall, average_precision, algo_name, n_classes)

    # Return all metrics
    results = pandas.DataFrame()

    for k in average_precision.keys():
        results.at[algo_name, f'P-R {k}'] = numpy.round(average_precision[k],
3)
        results.at[algo_name, f'F1 {k}'] = numpy.round(average_f1[k], 3)
        results.at[algo_name, f'MCC {k}'] = numpy.round(mcc[k], 3)

    return results

# Function to Plot Precision, Recall, F1
def prf1_plot(precision, recall, average_precision, algo_name, n_classes):
    print(algo_name)
    print('Average precision score, micro-averaged over all classes: {0:0.2f}'
        .format(average_precision["micro"]))

    # Plot the micro-averaged Precision-Recall curve
    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(average_precision["micro"]))

    # Plot Precision-Recall curve for each class and iso-f1 curves
    # setup plot details
    colors = cycle(['navy', 'turquoise', 'darkorange', 'cornflowerblue', 'tea
l'])

    plt.figure(figsize=(7, 8))
    f_scores = numpy.linspace(0.2, 0.8, num=4)
    lines = []
    labels = []
    for f_score in f_scores:
        x = numpy.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["micro"]))

    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, -.5), prop=dict(size=14))

    plt.show()
```

Run the Base Models

Run SVM Model

In [16]:
```
svm_model = run_svm(x_train_w2v, y_train)
```

Run Logistic Regression Model

In [17]:
```
logreg_model = run_logreg(x_train_w2v, y_train)
```

Run Naive Bayes Classifier

In [18]:
```
nb_model = run_nb(x_train_w2v, y_train)
```

Run Decision Trees Classifier

In [19]:
```
dectree_model = run_dectree(x_train_w2v, y_train)
```

Get the scores

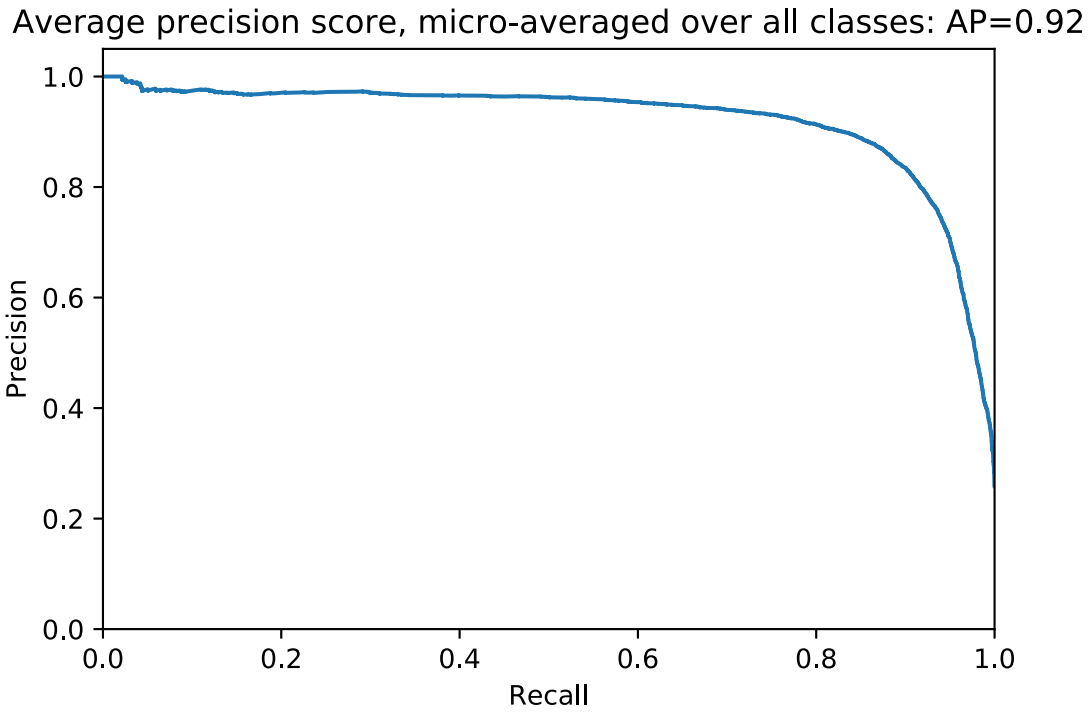
Initialize the dataframe to keep track of the scores

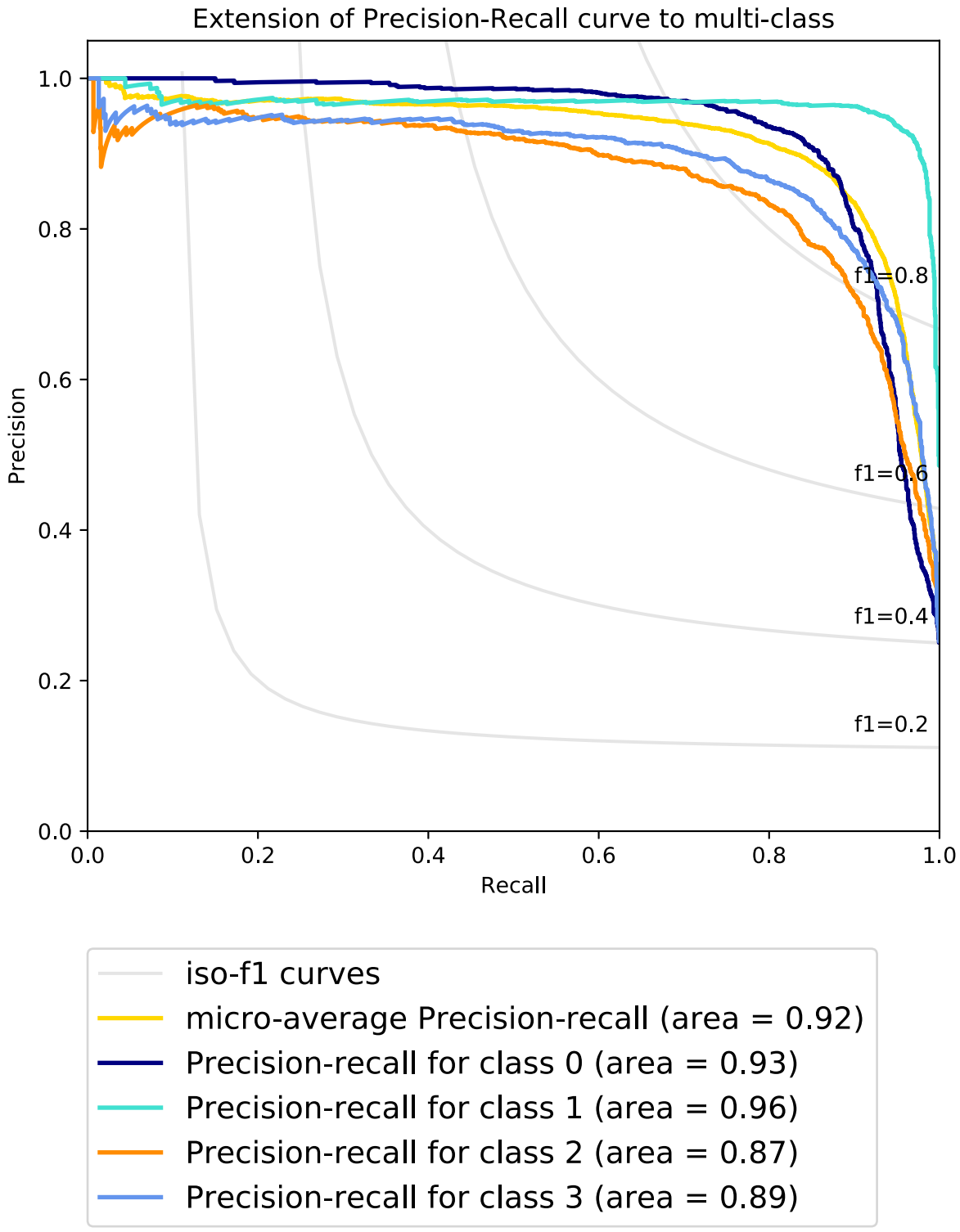
In [20]:
```
scores = pandas.DataFrame()
```

Precision, Recall, Avg. Precision for SVM

In [21]:
```
scores = scores.append(prf1_calc(svm_model, 'SVM', N_CLASSES, x_test_w2v, y_te
st))
```

SVM
Average precision score, micro-averaged over all classes: 0.92

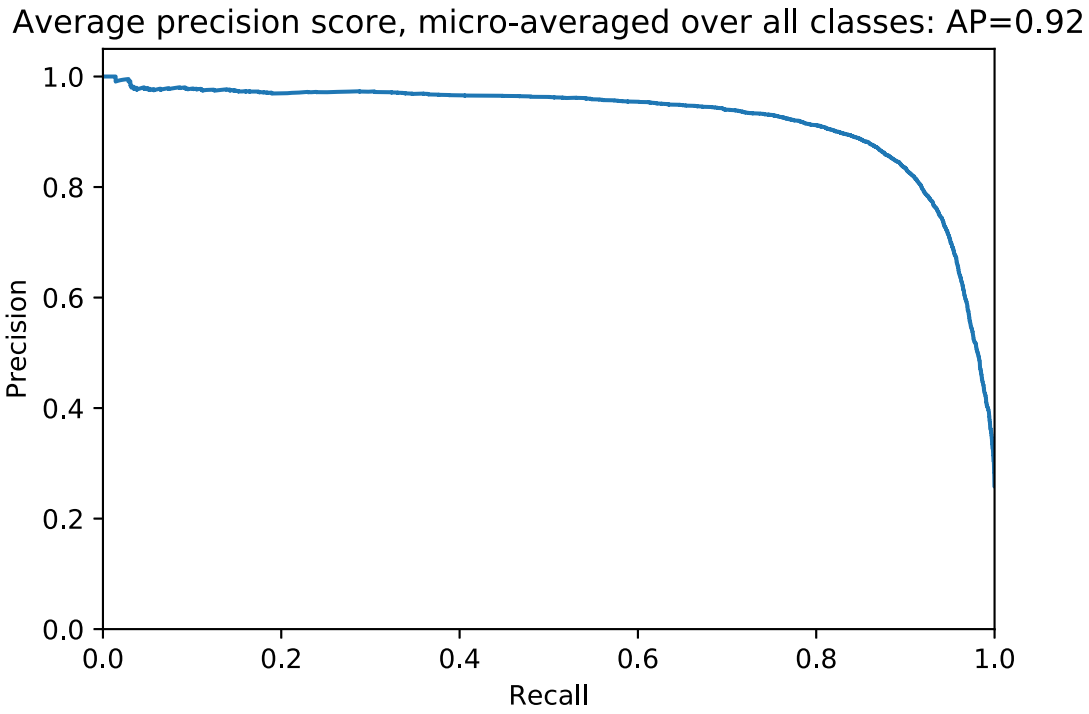


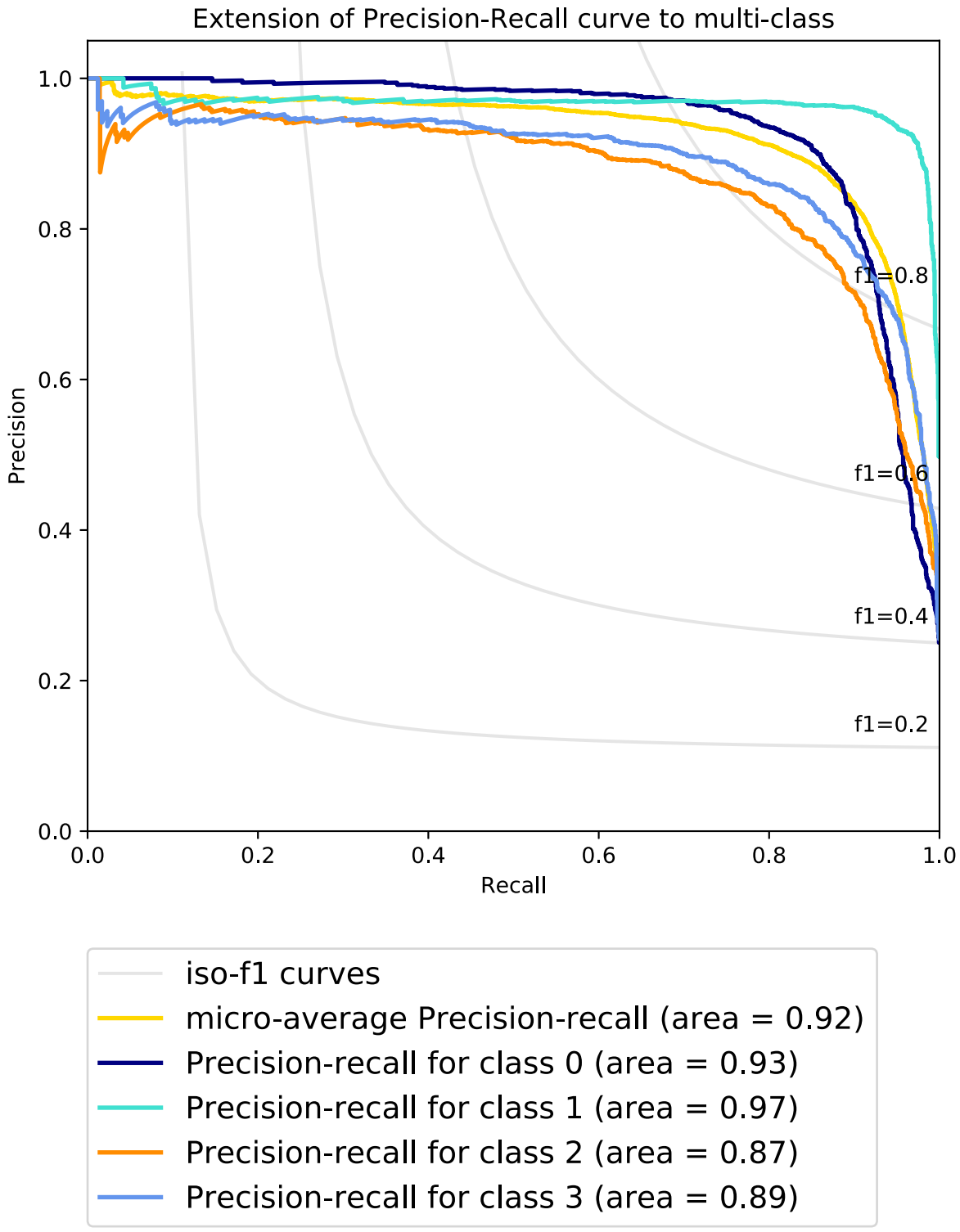


Precision, Recall, Avg. Precision for LOG REG

In [22]:
```
scores = scores.append(prf1_calc(logreg_model, 'LOGREG', N_CLASSES, x_test_w2v
, y_test))
```

LOGREG
Average precision score, micro-averaged over all classes: 0.92



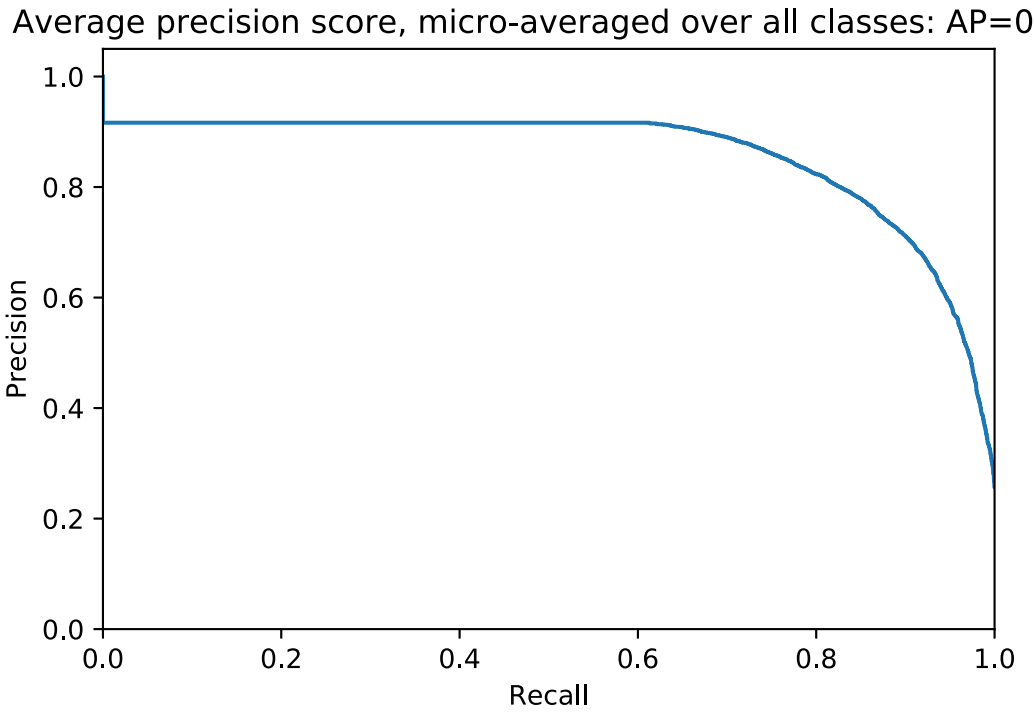


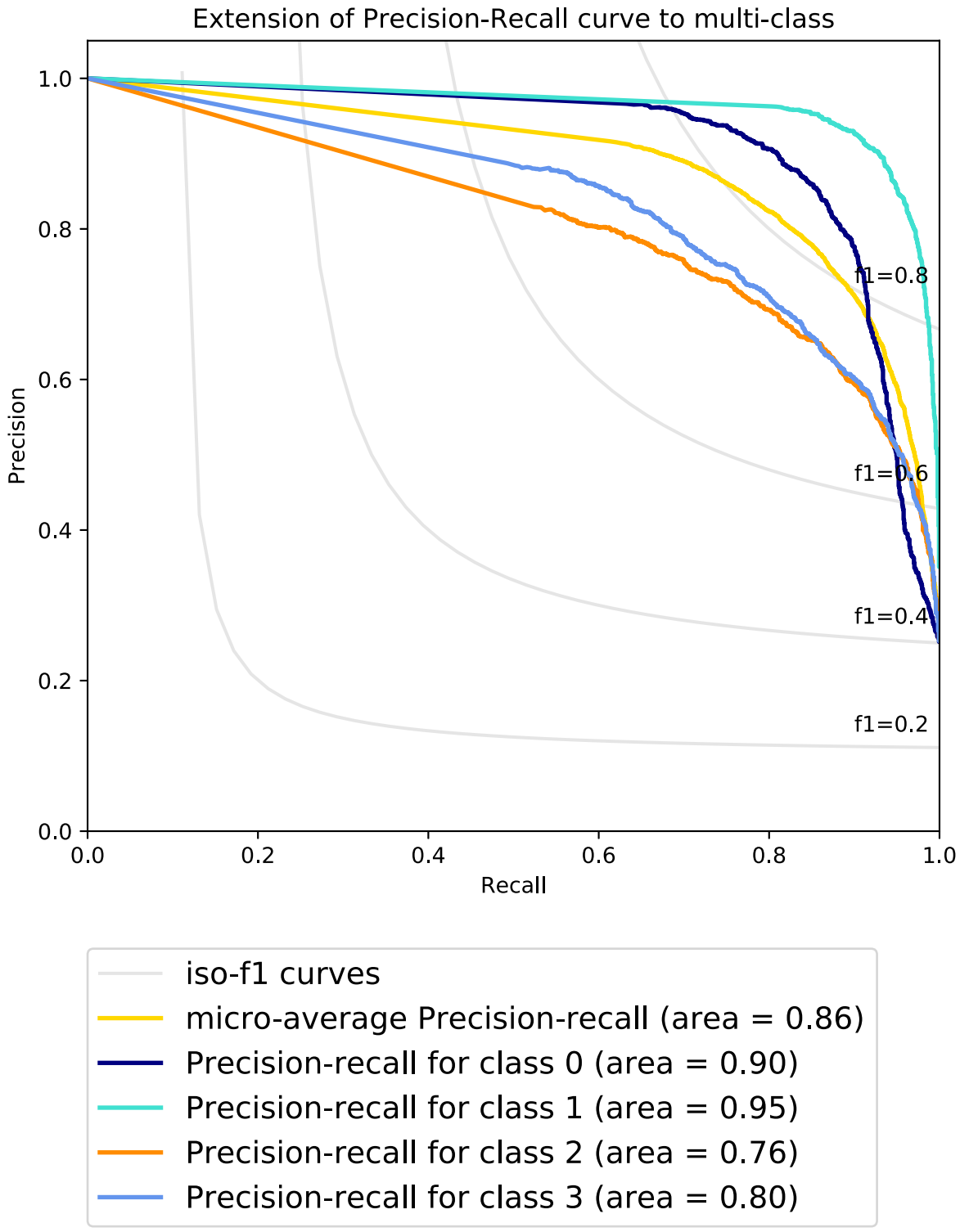
Precision, Recall, Avg. Precision for Naive Bayes

In [23]:
```
scores = scores.append(prf1_calc(nb_model, 'NB', N_CLASSES, x_test_w2v, y_test
))
```

NB

```
Average precision score, micro-averaged over all classes: 0.86
```

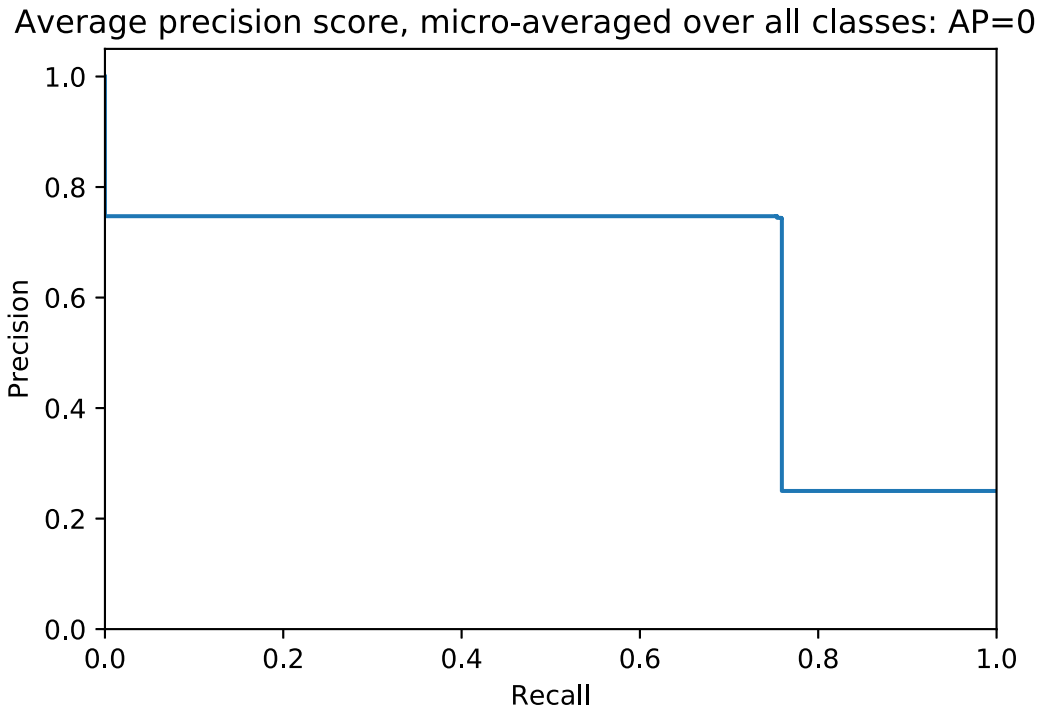


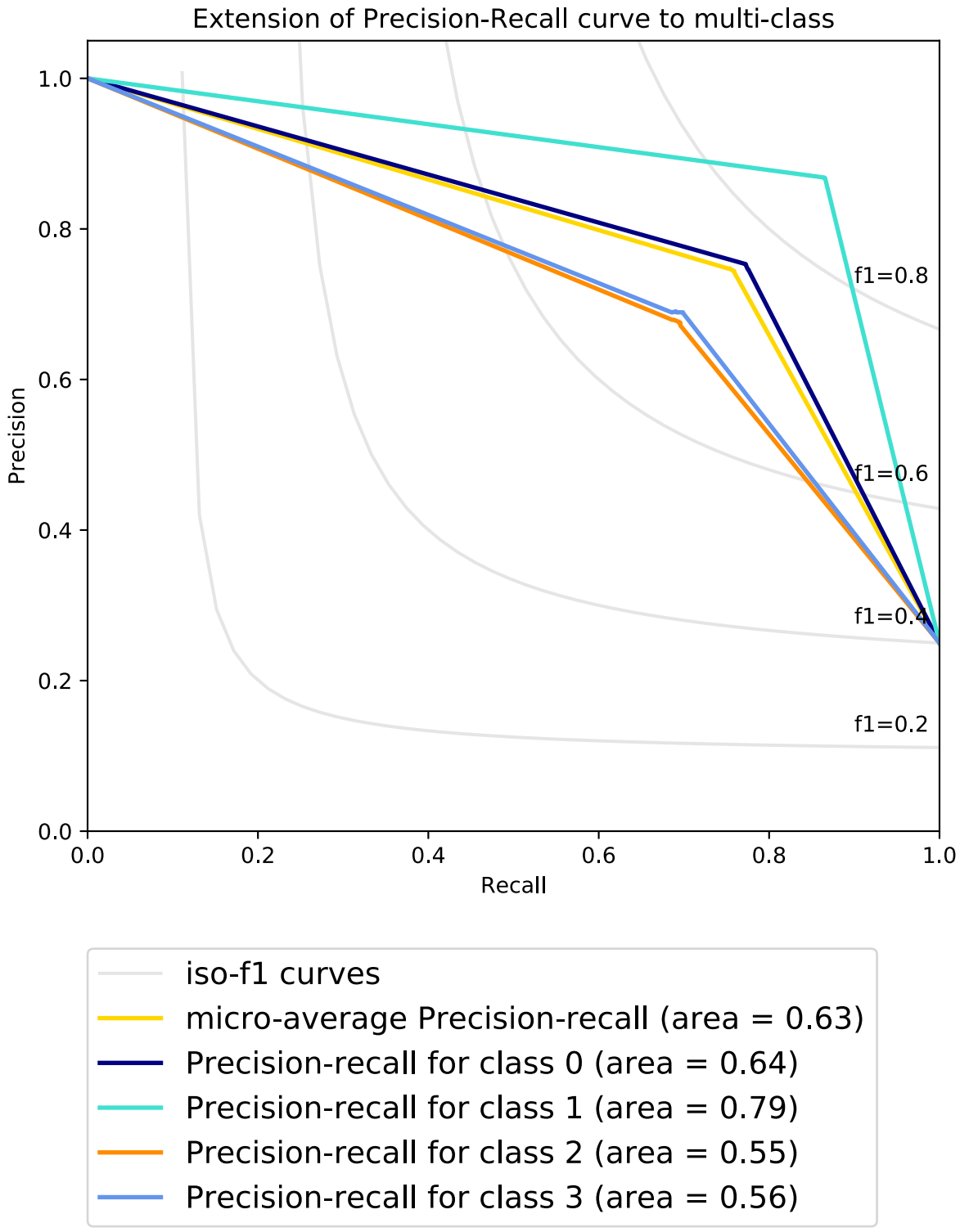


Precision, Recall, Avg. Precision for Decision Trees

In [24]:
```
scores = scores.append(prf1_calc(dectree_model, 'DT', N_CLASSES, x_test_w2v, y
_test))
```

DT
Average precision score, micro-averaged over all classes: 0.63





Look at Cross-Validation

Create model list to iterate through for cross validation

In [25]:
```
gnb = OneVsRestClassifier(GaussianNB())
sv = OneVsRestClassifier(svm.LinearSVC(random_state=RANDOM_STATE))
lreg = OneVsRestClassifier(LogisticRegression(random_state=RANDOM_STATE))
dtree = OneVsRestClassifier(tree.DecisionTreeClassifier())

model_list = [gnb, sv, lreg, dtree]
model_namelist = ['Gaussian Naive Bayes',
                  'SVM/Linear SVC',
                  'Logistic Regression',
                  'Decision Tree']
```

Make scoring metrics to pass cv function through

In [26]:
```
scoring = {'precision': make_scorer(precision_score, average='micro'),
           'recall': make_scorer(recall_score, average='micro'),
           'f1': make_scorer(f1_score, average='micro'),
           'roc_auc': make_scorer(roc_auc_score, average='micro'),
           # 'mcc': make_scorer(matthews_corrcoef) <- cannot support multi-lab
el
          }

cv_result_entries = []
i = 0
```

Loop cross validation through various models and generate results

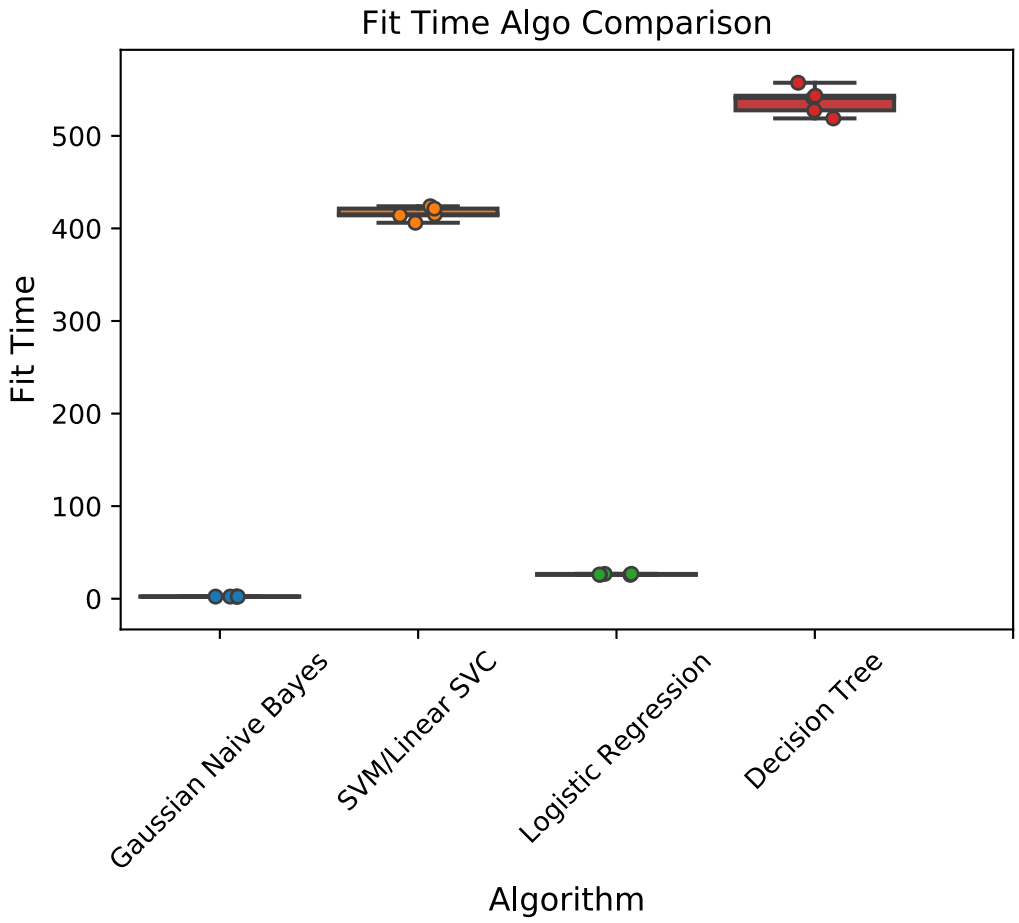
In [27]:
```
for mod in model_list:
    metrics = cross_validate(
        mod,
        x_train_w2v,
        y_train,
        cv=N_FOLDS,
        scoring = scoring,
        return_train_score=False,
        n_jobs=-1
    )
    for key in metrics.keys():
        for fold_index, score in enumerate(metrics[key]):
            cv_result_entries.append((model_namelist[i], fold_index, key, scor
e))
    i += 1
```

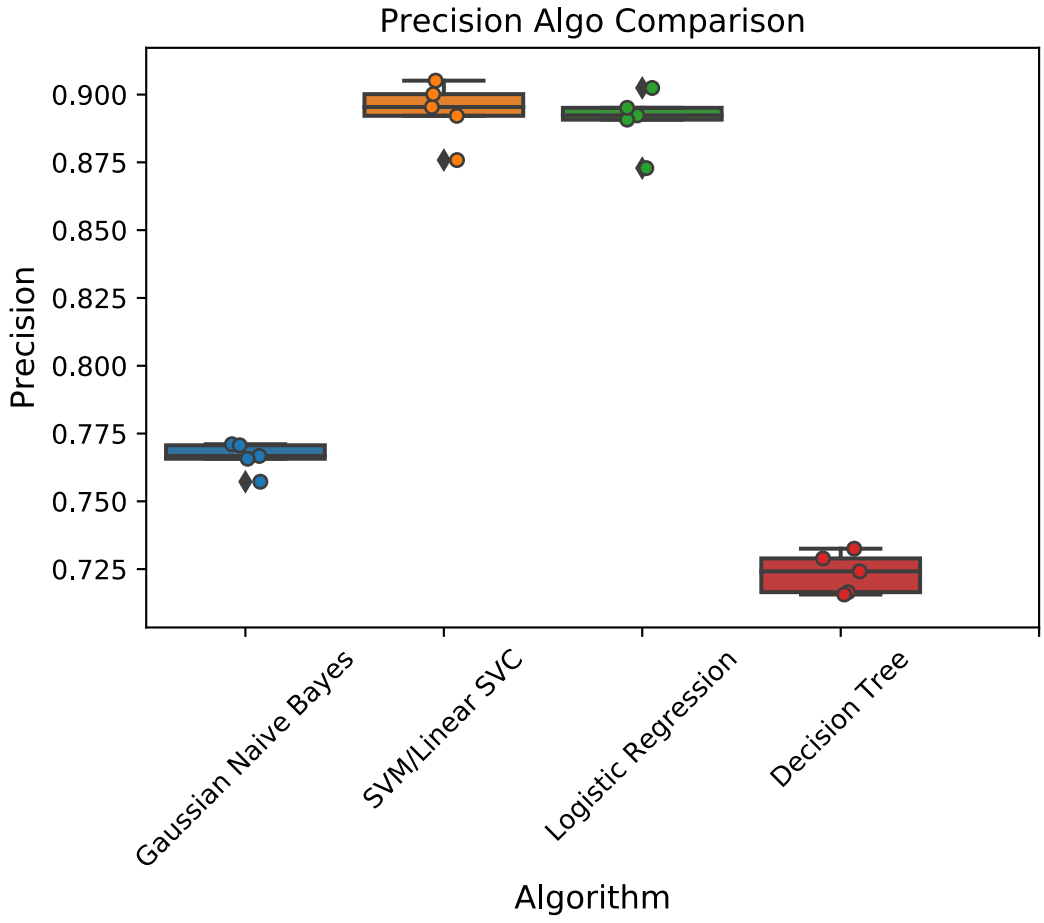
Save the cv results to a dataframe

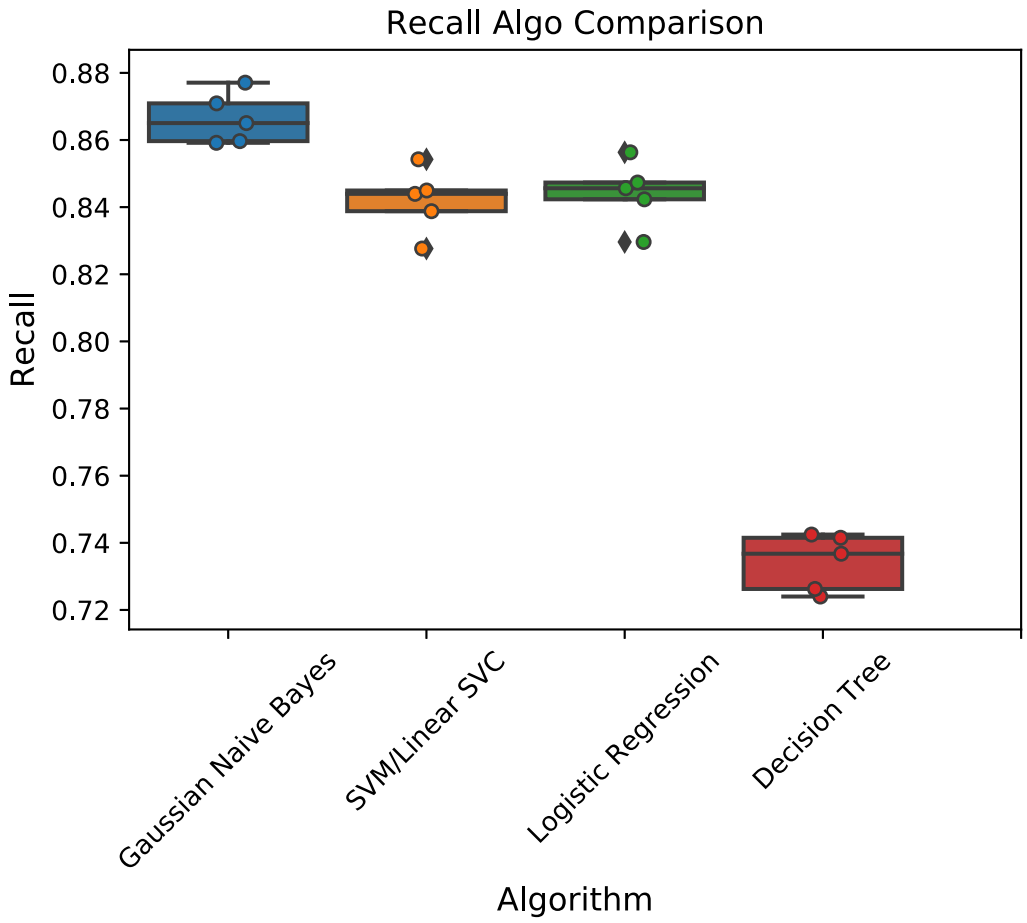
In [28]:
```
cv_results_df = pandas.DataFrame(cv_result_entries)
cv_results_df.columns = ['algo', 'cv fold', 'metric', 'value']
```

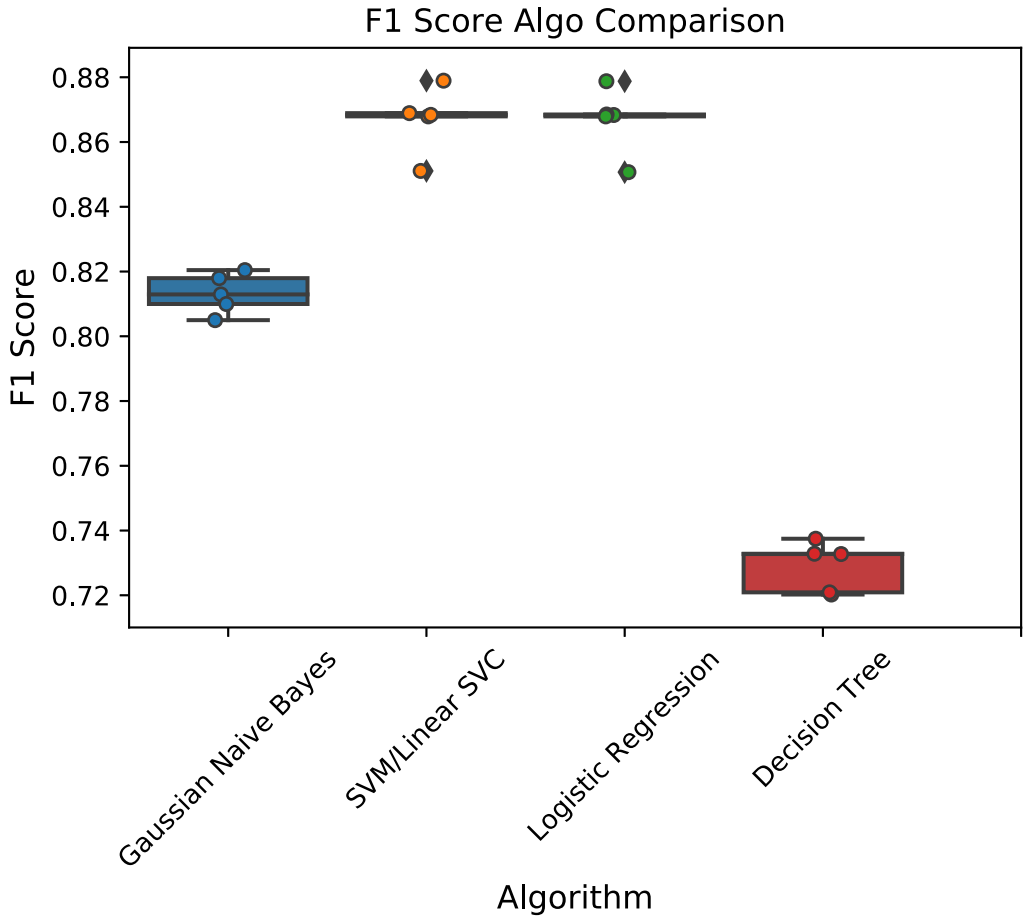
Plot cv results

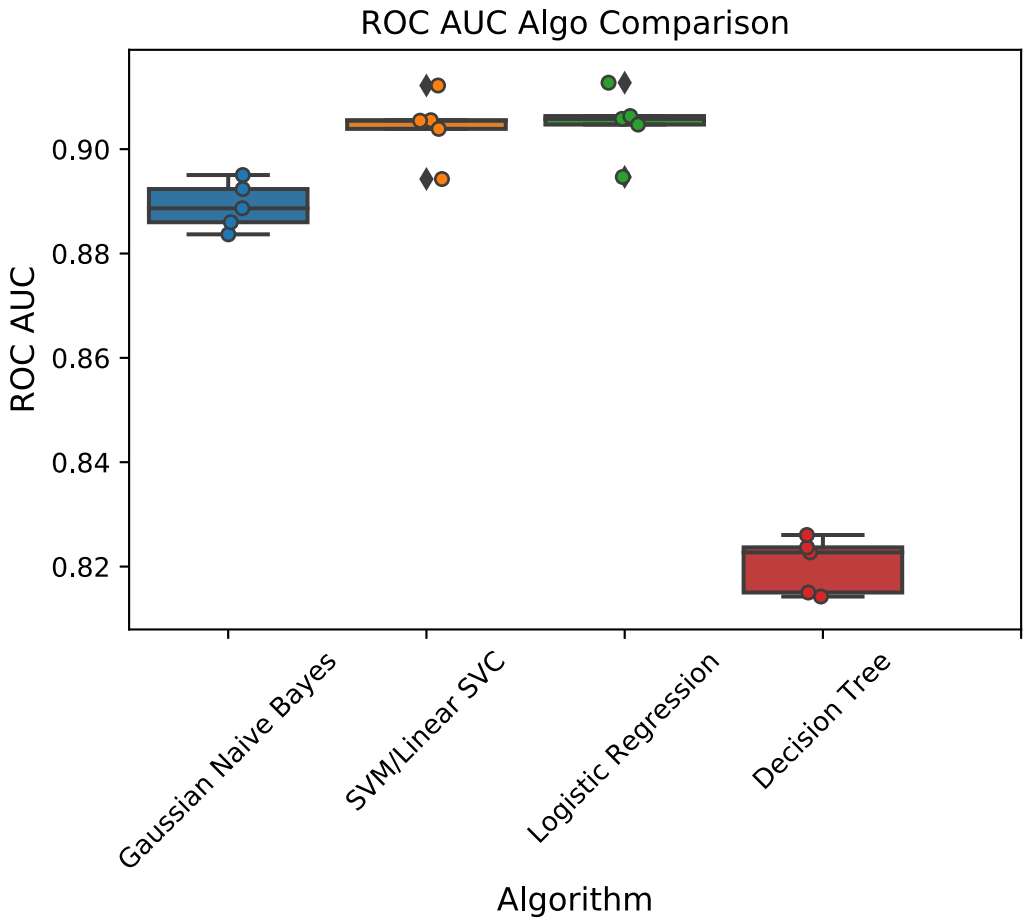
In [29]:
```
for metric_name, metric in zip(['fit_time',
                                'test_precision',
                                'test_recall',
                                'test_f1',
                                'test_roc_auc'],
                               ['Fit Time',
                                'Precision',
                                'Recall',
                                'F1 Score',
                                'ROC AUC']):
    sns.boxplot(x='algo', y='value', #hue='algo',
        data=cv_results_df[cv_results_df.metric.eq(f'{metric_name}')])
    sns.stripplot(x='algo', y = 'value',
        data = cv_results_df[cv_results_df.metric.eq(f'{metric_name}')],
        size = 5, linewidth = 1)
    plt.title(f'{metric} Algo Comparison', fontsize=12)
    plt.xlabel('Algorithm', fontsize=12)
    plt.ylabel(f'{metric}', fontsize=12)
    plt.xticks([0, 1, 2, 3, 4])
    plt.xticks(rotation=45)
    plt.show()
```


Fit Time Algo Comparison
Fit Time
500
400
300
200
100
0
Gaussian Naive Bayes
SVM/Linear SVC
Logistic Regression
Decision Tree
Algorithm


Precision Algo Comparison
Precision
0.900
0.875
0.850
0.825
0.800
0.775
0.750
0.725
Gaussian Naive Bayes
SVM/Linear SVC
Logistic Regression
Decision Tree
Algorithm


Recall Algo Comparison
Recall
0.88
0.86
0.84
0.82
0.80
0.78
0.76
0.74
0.72
Gaussian Naive Bayes
SVM/Linear SVC
Logistic Regression
Decision Tree
Algorithm

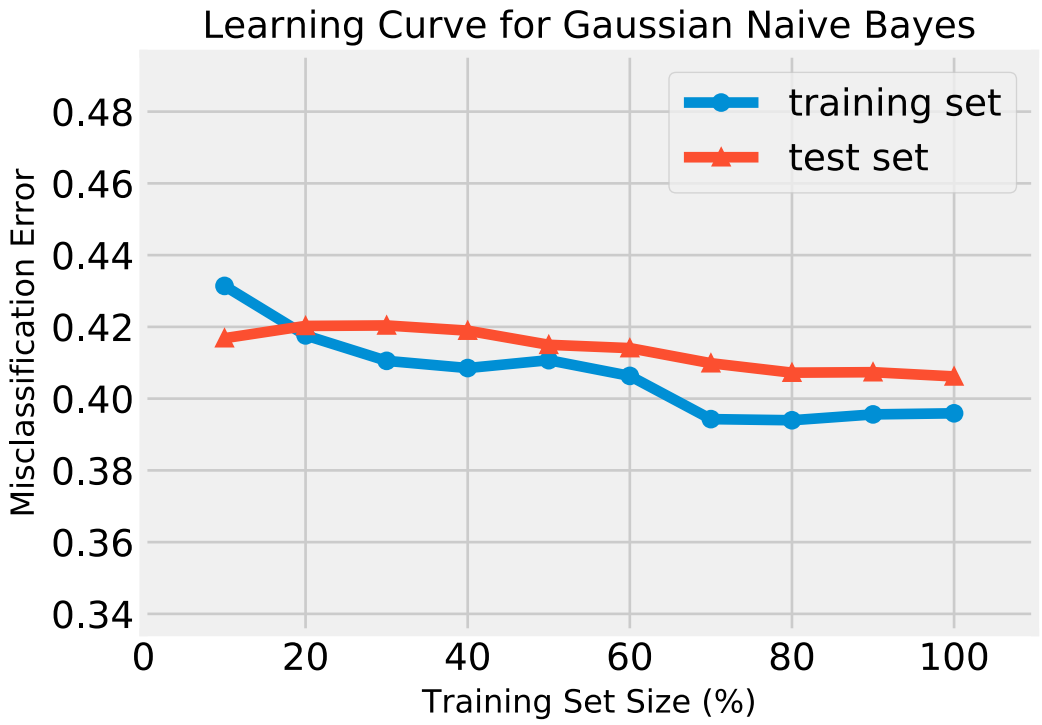

F1 Score Algo Comparison
F1 Score
0.88
0.86
0.84
0.82
0.80
0.78
0.76
0.74
0.72
Gaussian Naive Bayes
SVM/Linear SVC
Logistic Regression
Decision Tree
Algorithm

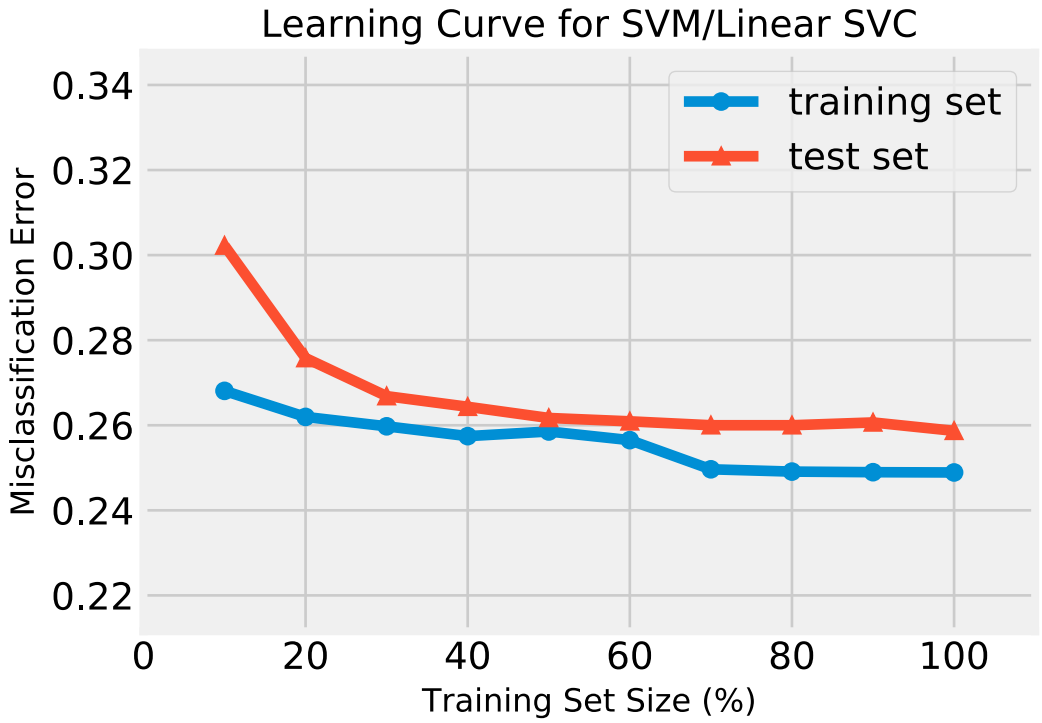


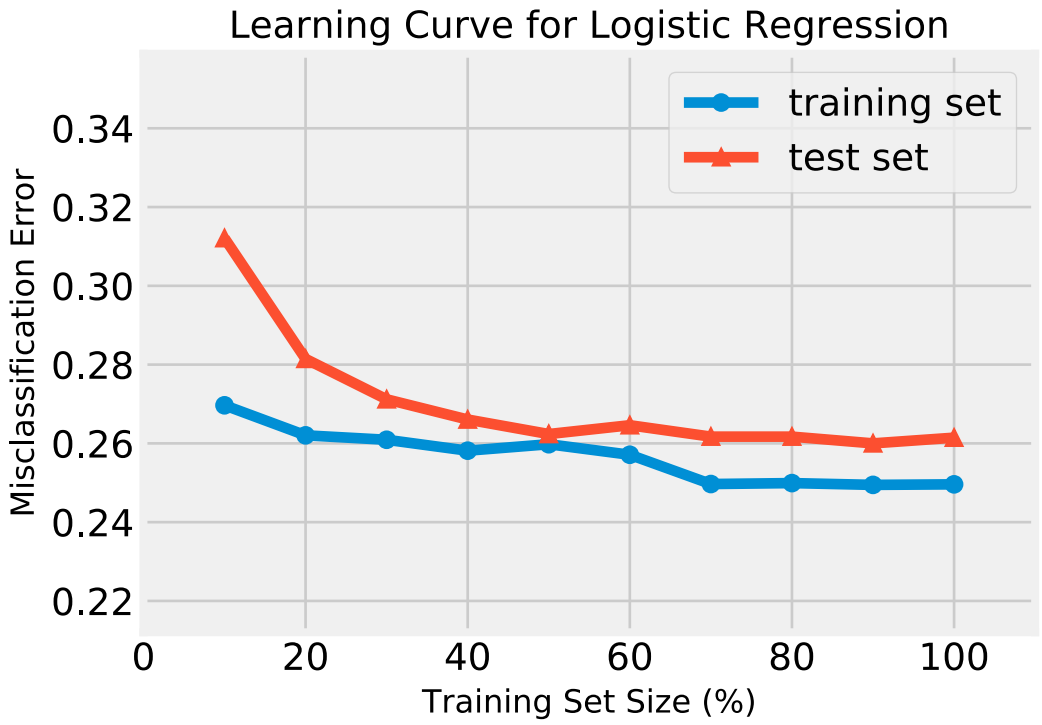
Misclassification Errors

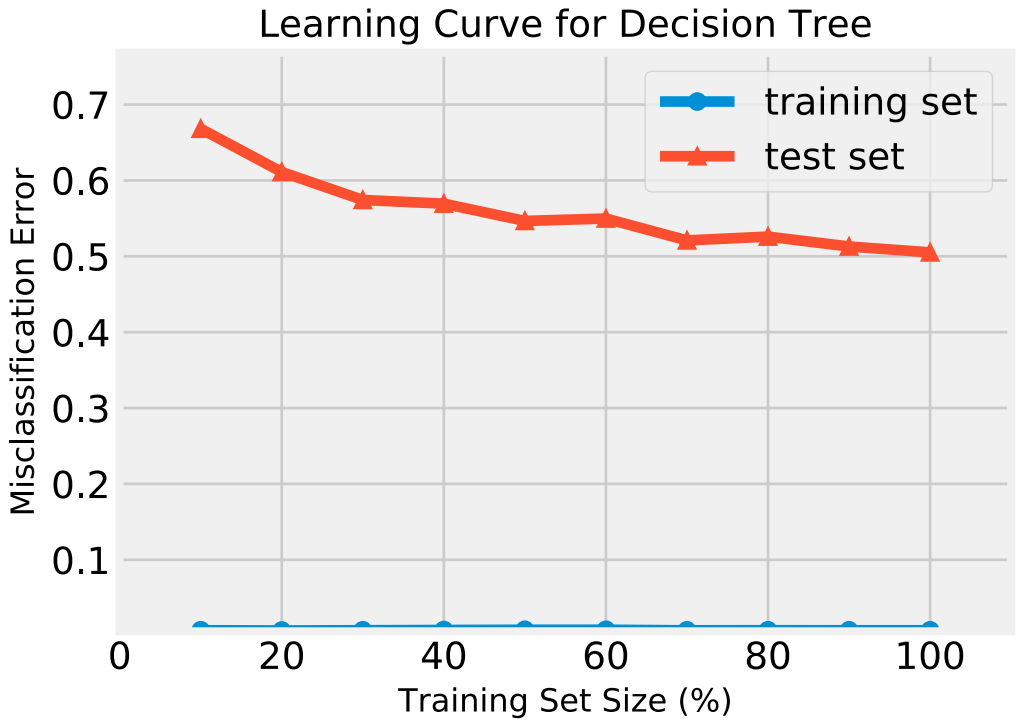
In [30]:

```
i=0
for model in model_list:
    plt.figure()
    plot_learning_curves(x_train_w2v, y_train, x_test_w2v, y_test, model)
    plt.title('Learning Curve for ' + model_namelist[i], fontsize=14)
    plt.xlabel('Training Set Size (%)', fontsize=12)
    plt.ylabel('Misclassification Error', fontsize=12)
    plt.show()
    i += 1
```


Learning Curve for Gaussian Naive Bayes
training set
test set
Misclassification Error
0.48
0.46
0.44
0.42
0.40
0.38
0.36
0.34
0
20
40
60
80
100
Training Set Size (%)


Learning Curve for SVM/Linear SVC
training set
test set
Misclassification Error
0.34
0.32
0.30
0.28
0.26
0.24
0.22
0
20
40
60
80
100
Training Set Size (%)



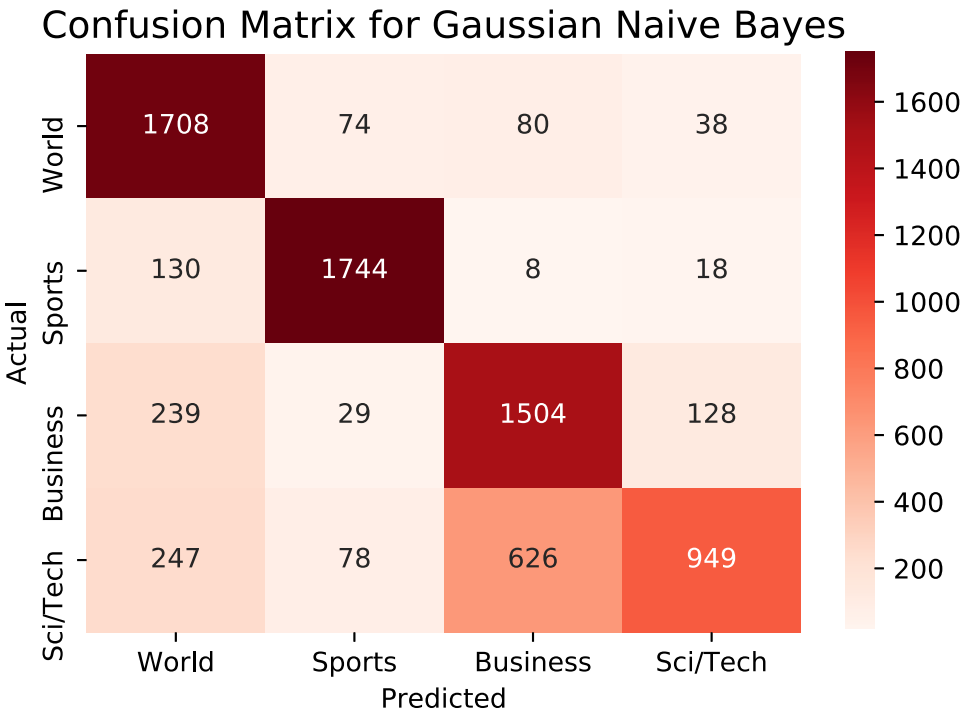


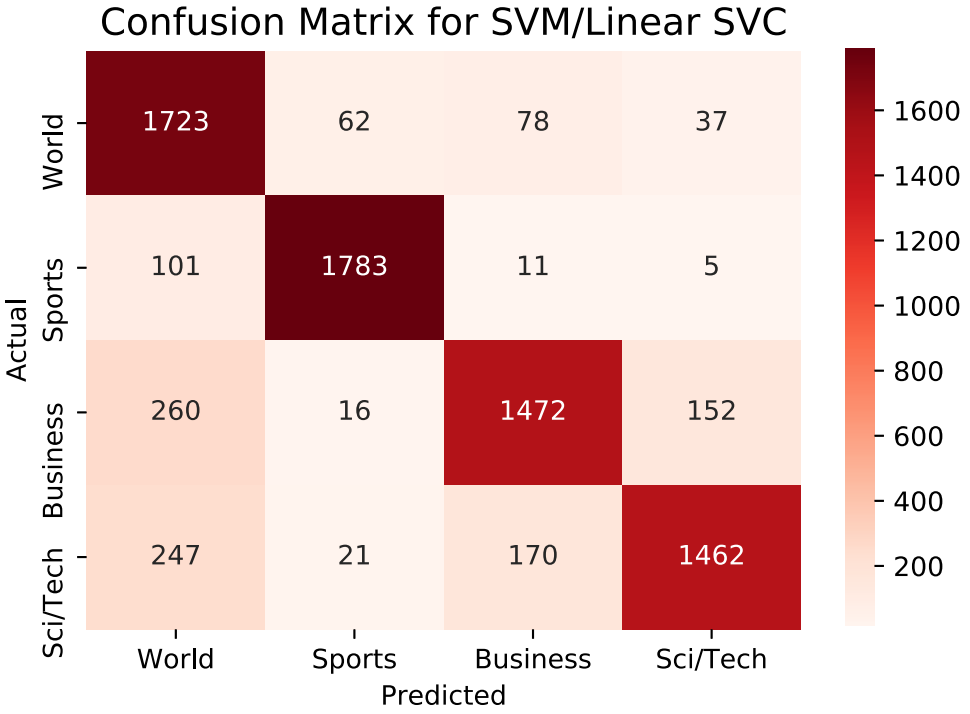
Get predictions

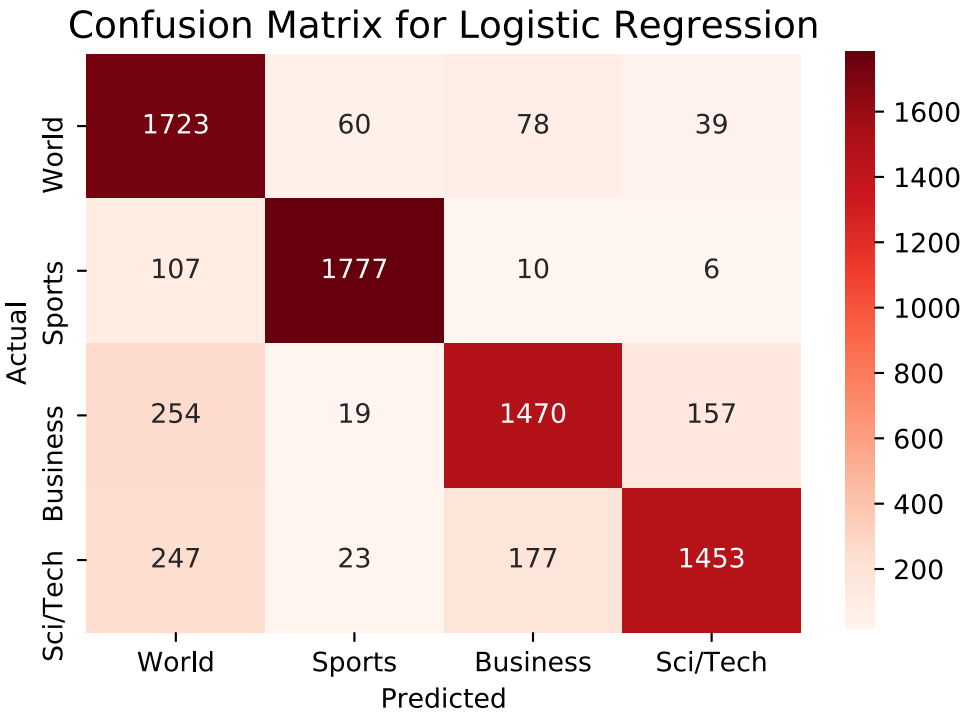
In [31]:
```
y_test_pred = []
for model in model_list:
    y_test_pred.append(model.predict(x_test_w2v))
```

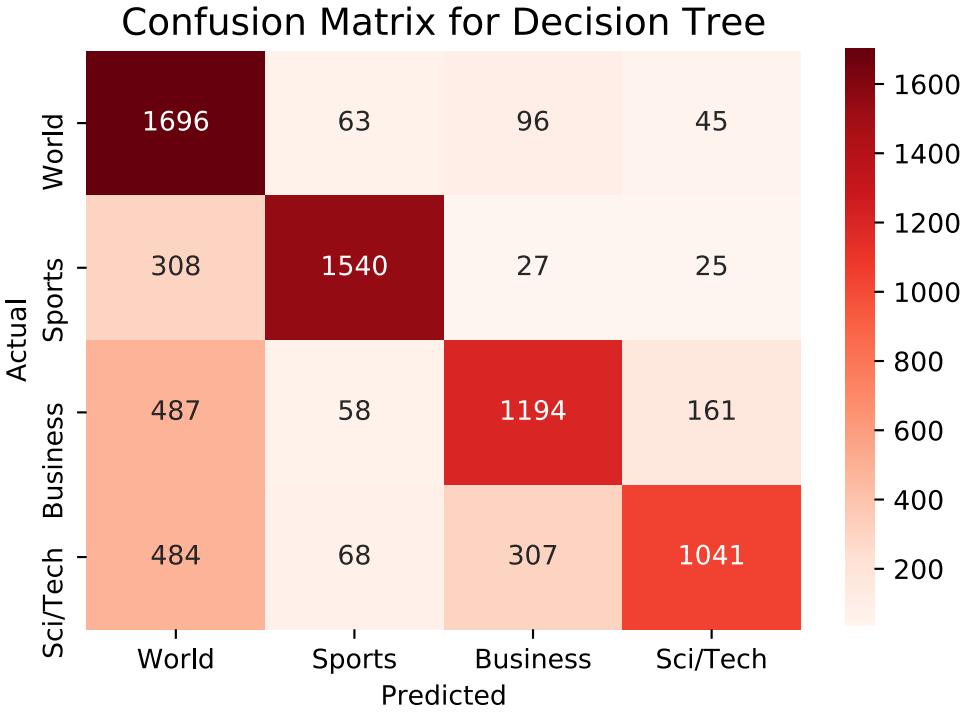
Confusion Matrix

In [32]:
```
CLASSES = ['World', 'Sports', 'Business', 'Sci/Tech']
i=0
for _ in model_list:
    cm = confusion_matrix(numpy.argmax(y_test, axis=1),
                          numpy.argmax(y_test_pred[i], axis=1))
    cm_df = pandas.DataFrame(cm, index = CLASSES, columns = CLASSES)
    cm_df.index.name = 'Actual'
    cm_df.columns.name = 'Predicted'
    plt.title('Confusion Matrix for ' + model_namelist[i], fontsize=14)
    sns.heatmap(cm_df, annot=True, fmt='.6g', annot_kws={"size": 10}, cmap='Re
ds')
    plt.show()
    i += 1
```


Confusion Matrix for Gaussian Naive Bayes
1708
74
80
38
130
1744
8
18
239
29
1504
128
247
78
626
949
World
Sports
Business
Sci/Tech
Actual
World
Sports
Business
Sci/Tech
Predicted
1600
1400
1200
1000
800
600
400
200


Confusion Matrix for SVM/Linear SVC
1723
62
78
37
101
1783
11
5
260
16
1472
152
247
21
170
1462
World
Sports
Business
Sci/Tech
Actual
World
Sports
Business
Sci/Tech
Predicted
1600
1400
1200
1000
800
600
400
200


Confusion Matrix for Logistic Regression
1723
60
78
39
107
1777
10
6
254
19
1470
157
247
23
177
1453
World
Sports
Business
Sci/Tech
Actual
World
Sports
Business
Sci/Tech
Predicted
1600
1400
1200
1000
800
600
400
200


Confusion Matrix for Decision Tree
1696
63
96
45
308
1540
27
25
487
58
1194
161
484
68
307
1041
World
Sports
Business
Sci/Tech
Actual
World
Sports
Business
Sci/Tech
Predicted
1600
1400
1200
1000
800
600
400
200

HYPER PARAMETER TUNING BY HYPEROPT (not working!!!)

from hyperopt import STATUS_OK N_FOLDS = 5

%%

Objective Function

def objective(params, n_folds = N_FOLDS): cv_results = cross_validate(OneVsRestClassifier(GaussianNB()), x_train_w2v, y_train, cv = n_folds, fit_params= params, scoring = {'f1': make_scorer(f1_score, average='micro')}, return_train_score=False, n_jobs=-1)

```
    # Extract the best score
    best_score = max(cv_results['test_f1'])
    # Loss must be minimized
    loss = 1 - best_score
    # Dictionary with information for evaluation
    return {'loss': loss, 'params': params, 'status': STATUS_OK}
```

Domain Space from hyperopt import hp space = {'estimator**var_smoothing':** **hp.uniform('estimator**var_smoothing', 1.e+00, 1.e-09)}

%%

Optimization Algorithm from hyperopt import tpe tpe_algo = tpe.suggest

%%

Results History from hyperopt import Trials bayes_trials = Trials()

%%

Run the optimization from hyperopt import fmin from hyperopt import rand MAX_EVALS = 500 params = space Optimize best = fmin(fn = objective, space = space, algo = tpe.suggest, max_evals = 100, trials = bayes_trials) print(best)

In [33]:
```
tempx = x_train_w2v
tempy = y_train
x_train_w2v = x_train_w2v.sample(n = 3000, replace = False, random_state = RAN
DOM_STATE)
y_train = train_data_df.category.sample(
    n = 3000, replace = False, random_state = RANDOM_STATE
)
y_train = label_binarize(y_train, classes=range(1, N_CLASSES + 1))
```

Hyper-parameter tuning with exhaustive Grid Search

Tune hyperparameters for Gaussian Naive-Bayes

In [34]:
```
params_gnb = {'estimator__var_smoothing': [1.e-09, 1.e-08, 1.e-07, 1.e-06, 1.e
-05,
                                           1.e-04, 1.e-03, 1.e-02, 1.e-01, 1.e
+00]
}
clf = GridSearchCV(estimator=gnb,
                   param_grid=params_gnb,
                   scoring='f1_micro',
                   n_jobs=-1,
                   cv=N_FOLDS,
                   return_train_score=True
                  )
clf_res = clf.fit(x_train_w2v, y_train)
print('Best Score: ', clf_res.best_score_)
print('Best Params: ', clf_res.best_params_)
```

```
Best Score:  0.801674851179991
Best Params:  {'estimator__var_smoothing': 0.001}
```

Tune hyperparameters for Logistic Regression

In [35]:
```
params_lreg = {
    "estimator__penalty": ['l1', 'l2'],
    "estimator__C": [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000],
    #"estimator__class_weight":[{1:0.5, 0:0.5}, {1:0.4, 0:0.6},
    #                           {1:0.6, 0:0.4}, {1:0.7, 0:0.3}],
    "estimator__solver": ["newton-cg", "sag", "saga", "lbfgs"]
}
clf = GridSearchCV(estimator=lreg,
                   param_grid=params_lreg,
                   scoring='f1_micro',
                   n_jobs=-1,
                   cv=N_FOLDS,
                   return_train_score=True
                  )
clf_res = clf.fit(x_train_w2v, y_train)
print('Best score:', clf_res.best_score_)
print('Best Params:', clf_res.best_params_)
```

```
Best score: 0.8320745341017949
Best Params: {'estimator__C': 0.01, 'estimator__penalty': 'l2', 'estimator__s
olver': 'newton-cg'}
```

Tune hyperparameters for SVM (Linear SVC)

In [36]:
```
params_sv = {
    "estimator__penalty":['l1', 'l2'],
    "estimator__tol": [1.e-08, 1.e-07, 1.e-06, 1.e-05,
                       1.e-04, 1.e-03, 1.e-02, 1.e-01, 1.e+00],
    "estimator__loss":['hinge','squared_hinge'],
    "estimator__C": [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
    #"estimator__class_weight":['None',{1:0.5, 0:0.5},
    #                           {1:0.4, 0:0.6}, {1:0.6, 0:0.4}, {1:0.7, 0:0.
3}],
}
clf = GridSearchCV(estimator=sv,
                   param_grid=params_sv,
                   scoring='f1_micro',
                   n_jobs=-1,
                   cv=N_FOLDS,
                   return_train_score=False
                  )
clf_res = clf.fit(x_train_w2v, y_train)
print('Best score:', clf_res.best_score_)
print('Best Params:', clf_res.best_params_)
```

```
Best score: 0.8440514791910111
Best Params: {'estimator__C': 0.001, 'estimator__loss': 'squared_hinge', 'est
imator__penalty': 'l2', 'estimator__tol': 1e-08}
```

Tune hyperparameters for Decision Trees

In [37]:
```
params_dtree = {
    "estimator__splitter":["best", "random"],
    "estimator__min_samples_split":range(1, 20, 1)
}
clf = GridSearchCV(estimator=dtree,
                   param_grid=params_dtree,
                   scoring='f1_micro',
                   n_jobs=-1,
                   cv=N_FOLDS,
                   return_train_score=False
                  )
clf_res = clf.fit(x_train_w2v, y_train)
print('Best score:', clf_res.best_score_)
print('Best Params:', clf_res.best_params_)
```

```
Best score: 0.6746835492233547
Best Params: {'estimator__min_samples_split': 9, 'estimator__splitter': 'bes
t'}
```

Conclusion:

Apparently the best params are pretty much the default ones. The algorithms are already pretty smart about the defaults or can calculate them. This tuning these hyper-parameters might actually cause overfitting.

Ensemble Methods

Stacking

In [39]:

```
estimators = [
              ('nb', GaussianNB()),
              ('svm', svm.LinearSVC())
             ]

sclf = OneVsRestClassifier(StackingClassifier(
    estimators=estimators, final_estimator=LogisticRegression())
)

metrics = cross_validate(
    sclf,
    x_train_w2v,
    y_train,
    cv=N_FOLDS,
    scoring = scoring,
    return_train_score=False,
    n_jobs=-1
)

res = []
for key in metrics.keys():
    for fold_index, score in enumerate(metrics[key]):
        res.append(('Stacking', fold_index, key, score))

res_df = pandas.DataFrame.from_dict(res)
res_df.columns = ['algo', 'cv fold', 'metric', 'value']
cv_results_inc_ens = pandas.concat([cv_results_df, res_df])
print(res_df)
```

	algo	cv fold	metric	value
0	Stacking	0	fit_time	2086.022402
1	Stacking	1	fit_time	2133.732171
2	Stacking	2	fit_time	2122.568591
3	Stacking	3	fit_time	2172.070424
4	Stacking	4	fit_time	2157.726120
5	Stacking	0	score_time	1.544760
6	Stacking	1	score_time	1.320186
7	Stacking	2	score_time	1.514731
8	Stacking	3	score_time	1.128256
9	Stacking	4	score_time	1.213134
10	Stacking	0	test_precision	0.872012
11	Stacking	1	test_precision	0.893003
12	Stacking	2	test_precision	0.891541
13	Stacking	3	test_precision	0.902576
14	Stacking	4	test_precision	0.896064
15	Stacking	0	test_recall	0.831500
16	Stacking	1	test_recall	0.846083
17	Stacking	2	test_recall	0.848042
18	Stacking	3	test_recall	0.856958
19	Stacking	4	test_recall	0.841292
20	Stacking	0	test_f1	0.851274
21	Stacking	1	test_f1	0.868910
22	Stacking	2	test_f1	0.869248
23	Stacking	3	test_f1	0.879176
24	Stacking	4	test_f1	0.867814
25	Stacking	0	test_roc_auc	0.895410
26	Stacking	1	test_roc_auc	0.906146
27	Stacking	2	test_roc_auc	0.906826
28	Stacking	3	test_roc_auc	0.913063
29	Stacking	4	test_roc_auc	0.904382

Bagging

In [40]:
```
sclf = OneVsRestClassifier(BaggingClassifier(
    base_estimator=LogisticRegression())
)

metrics = cross_validate(
    sclf,
    x_train_w2v,
    y_train,
    cv=N_FOLDS,
    scoring = scoring,
    return_train_score=False,
    n_jobs=-1
)

res = []
for key in metrics.keys():
    for fold_index, score in enumerate(metrics[key]):
        res.append(('Bagging', fold_index, key, score))

res_df = pandas.DataFrame.from_dict(res)
res_df.columns = ['algo', 'cv fold', 'metric', 'value']
cv_results_inc_ens = pandas.concat([cv_results_inc_ens, res_df])
print(res_df)
```

	algo	cv fold	metric	value
0	Bagging	0	fit_time	293.037744
1	Bagging	1	fit_time	288.202163
2	Bagging	2	fit_time	289.632665
3	Bagging	3	fit_time	297.167599
4	Bagging	4	fit_time	298.388267
5	Bagging	0	score_time	4.499739
6	Bagging	1	score_time	4.891524
7	Bagging	2	score_time	4.782574
8	Bagging	3	score_time	3.733881
9	Bagging	4	score_time	3.392392
10	Bagging	0	test_precision	0.872692
11	Bagging	1	test_precision	0.891740
12	Bagging	2	test_precision	0.890813
13	Bagging	3	test_precision	0.902427
14	Bagging	4	test_precision	0.895352
15	Bagging	0	test_recall	0.829167
16	Bagging	1	test_recall	0.845667
17	Bagging	2	test_recall	0.846792
18	Bagging	3	test_recall	0.856667
19	Bagging	4	test_recall	0.842750
20	Bagging	0	test_f1	0.850373
21	Bagging	1	test_f1	0.868092
22	Bagging	2	test_f1	0.868245
23	Bagging	3	test_f1	0.878952
24	Bagging	4	test_f1	0.868255
25	Bagging	0	test_roc_auc	0.894424
26	Bagging	1	test_roc_auc	0.905722
27	Bagging	2	test_roc_auc	0.906097
28	Bagging	3	test_roc_auc	0.912896
29	Bagging	4	test_roc_auc	0.904958

Boosting

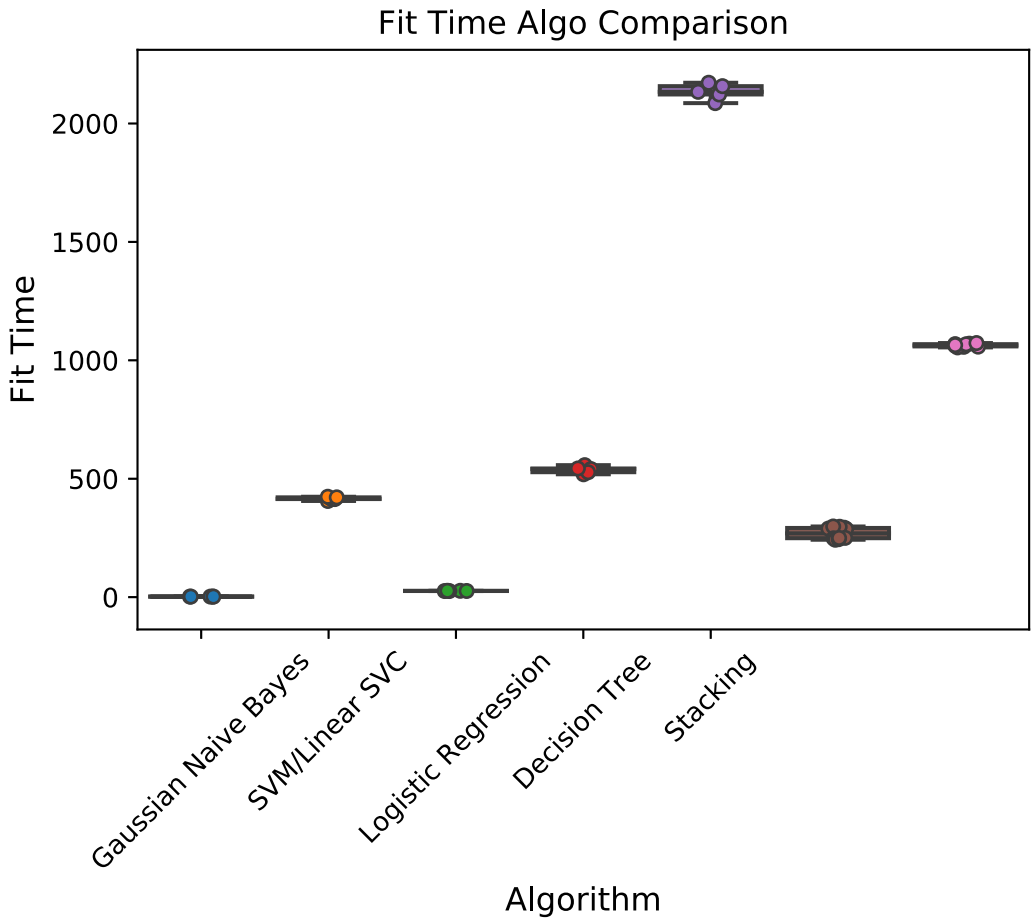
In [41]:
```
from sklearn.ensemble import AdaBoostClassifier
sclf = OneVsRestClassifier(AdaBoostClassifier(
    random_state=RANDOM_STATE)
)

metrics = cross_validate(
    sclf,
    x_train_w2v,
    y_train,
    cv=N_FOLDS,
    scoring = scoring,
    return_train_score=False,
    n_jobs=-1
)

res = []
for key in metrics.keys():
    for fold_index, score in enumerate(metrics[key]):
        res.append(('AdaBoost', fold_index, key, score))

res_df = pandas.DataFrame.from_dict(res)
res_df.columns = ['algo', 'cv fold', 'metric', 'value']
cv_results_inc_ens = pandas.concat([cv_results_inc_ens, res_df])
print(res_df)
```

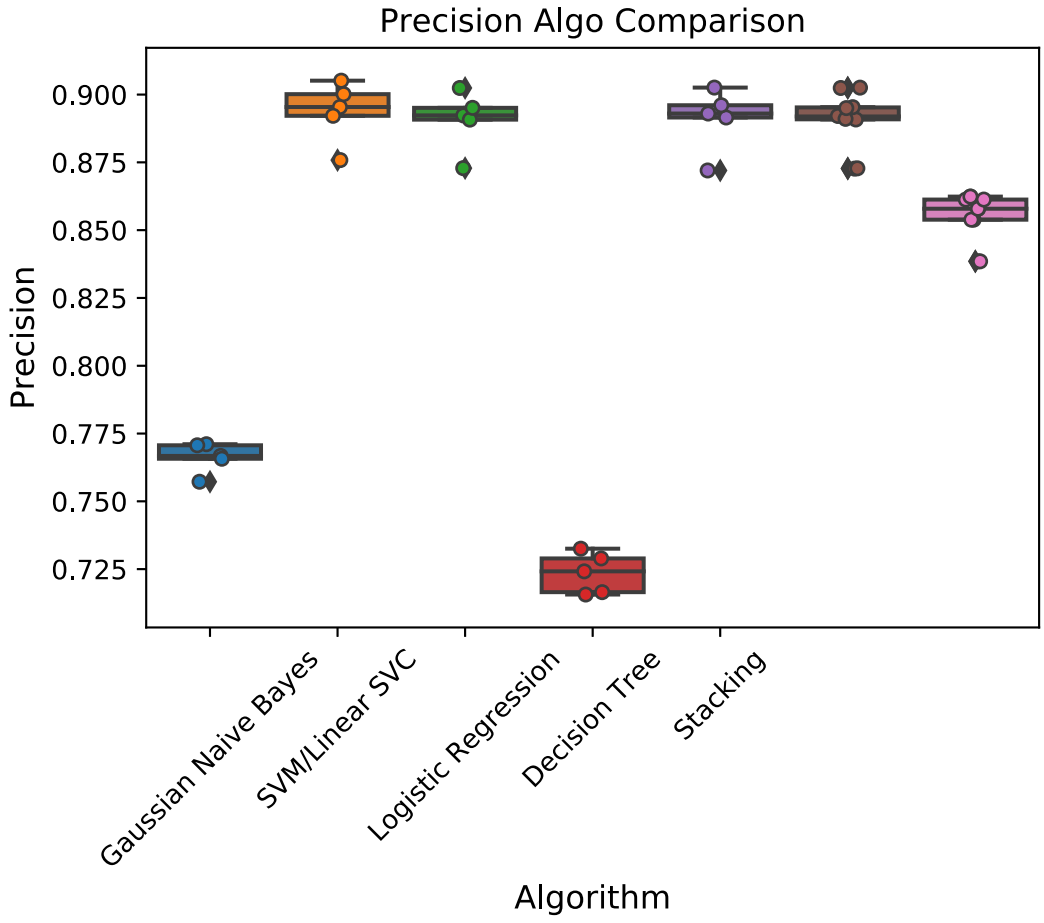
	algo	cv fold	metric	value
0	AdaBoost	0	fit_time	1062.197975
1	AdaBoost	1	fit_time	1057.098244
2	AdaBoost	2	fit_time	1057.839918
3	AdaBoost	3	fit_time	1054.535914
4	AdaBoost	4	fit_time	1061.208005
5	AdaBoost	0	score_time	1.882781
6	AdaBoost	1	score_time	2.074911
7	AdaBoost	2	score_time	2.073910
8	AdaBoost	3	score_time	2.134933
9	AdaBoost	4	score_time	1.903810
10	AdaBoost	0	test_precision	0.838507
11	AdaBoost	1	test_precision	0.853863
12	AdaBoost	2	test_precision	0.857904
13	AdaBoost	3	test_precision	0.862374
14	AdaBoost	4	test_precision	0.861283
15	AdaBoost	0	test_recall	0.784458
16	AdaBoost	1	test_recall	0.807292
17	AdaBoost	2	test_recall	0.805500
18	AdaBoost	3	test_recall	0.811458
19	AdaBoost	4	test_recall	0.793708
20	AdaBoost	0	test_f1	0.810583
21	AdaBoost	1	test_f1	0.829924
22	AdaBoost	2	test_f1	0.830876
23	AdaBoost	3	test_f1	0.836142
24	AdaBoost	4	test_f1	0.826116
25	AdaBoost	0	test_roc_auc	0.867049
26	AdaBoost	1	test_roc_auc	0.880618
27	AdaBoost	2	test_roc_auc	0.880514
28	AdaBoost	3	test_roc_auc	0.884146
29	AdaBoost	4	test_roc_auc	0.875549

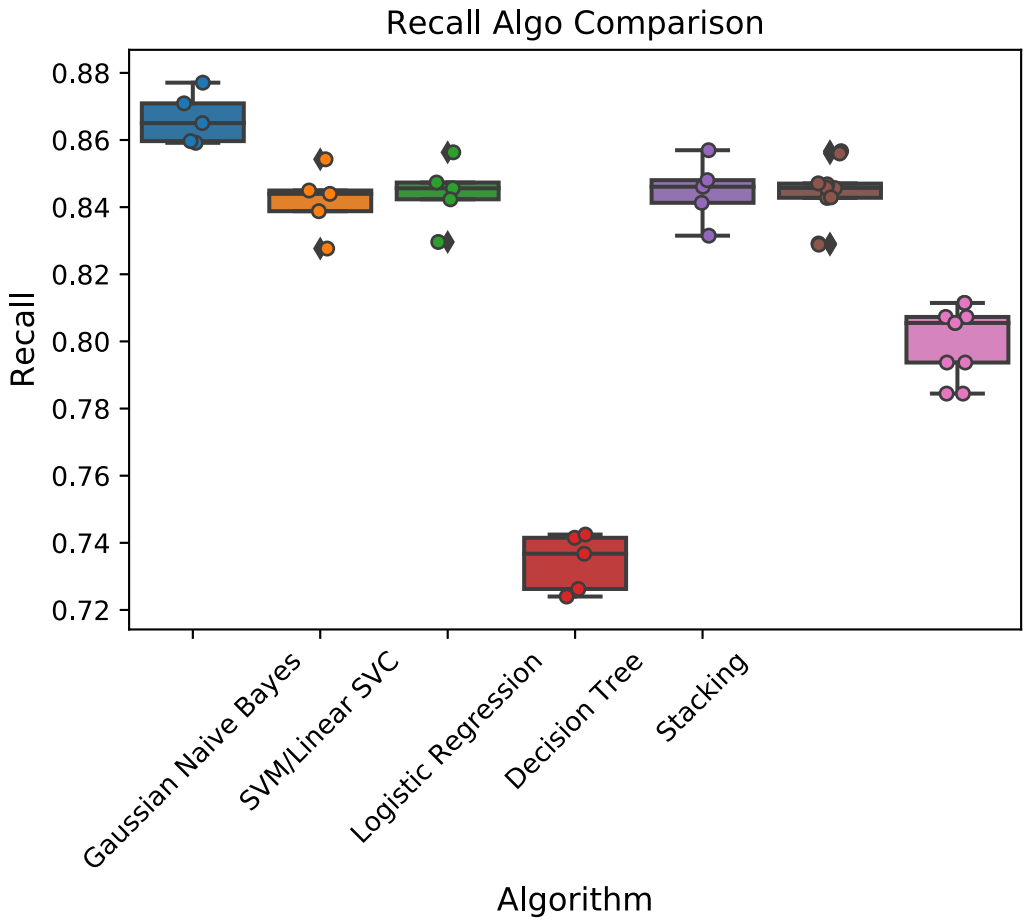
Plot cv results

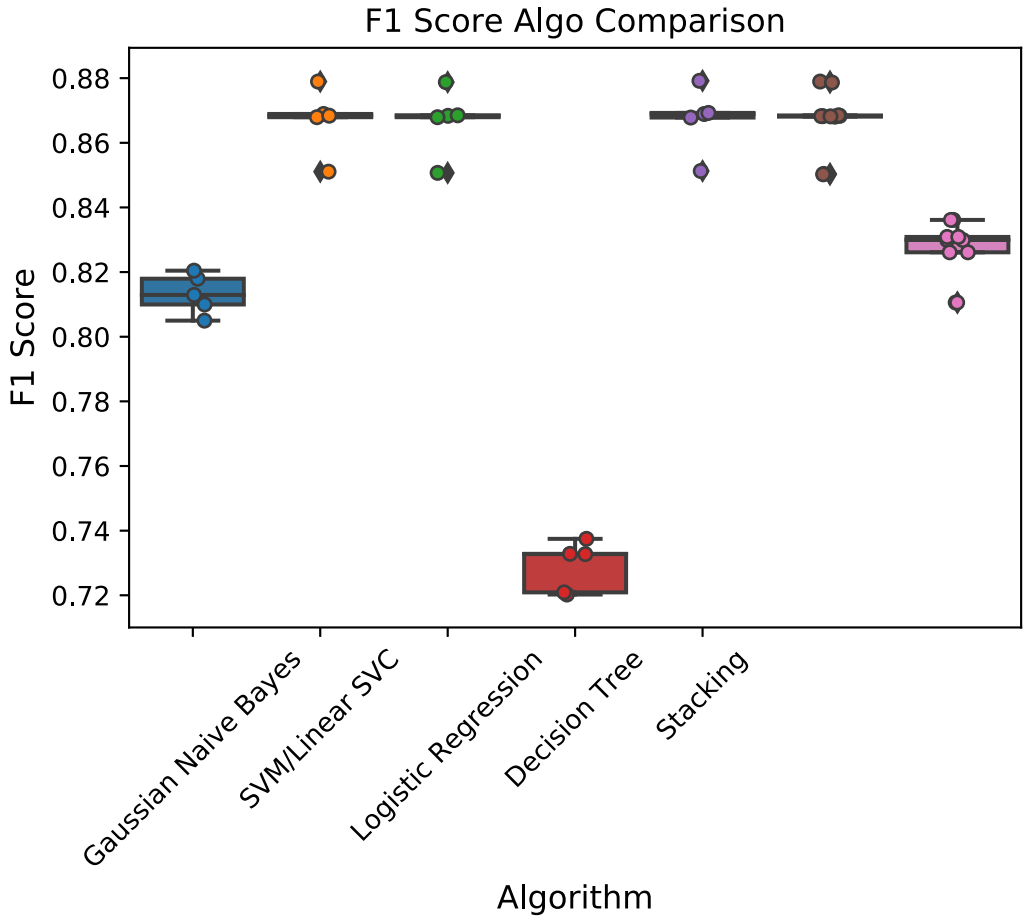
In [46]:

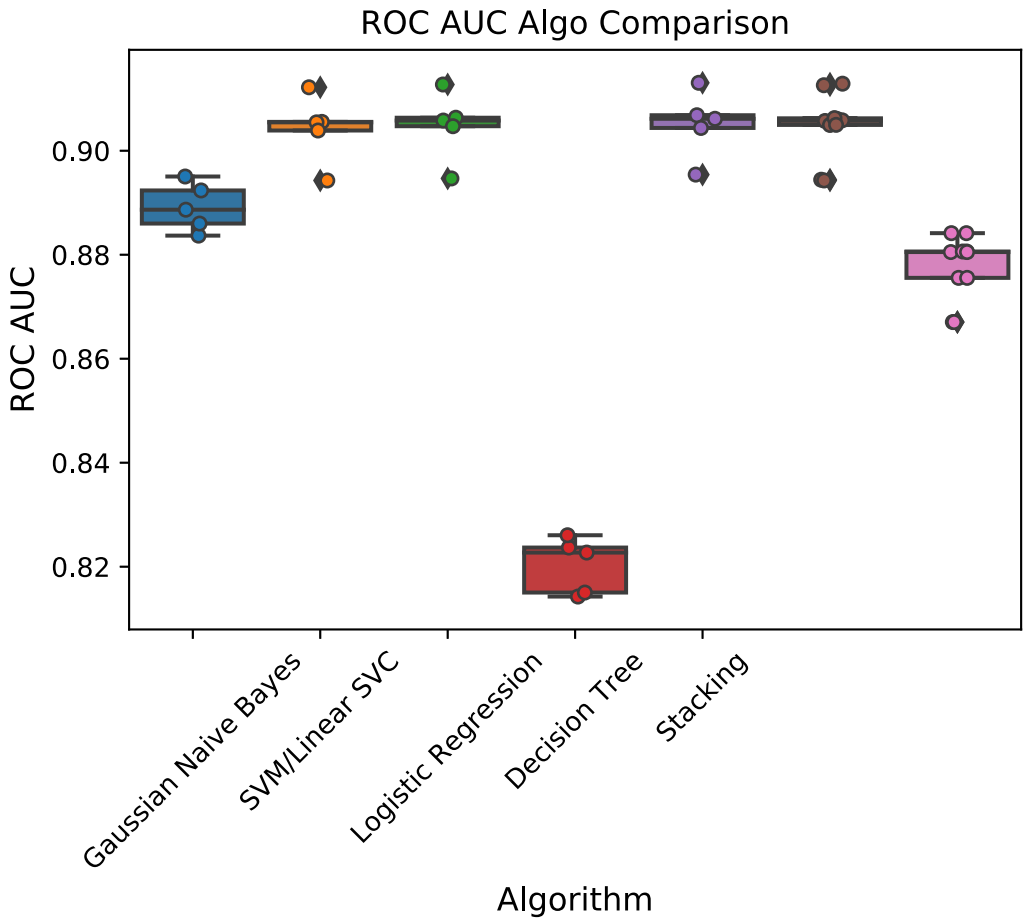
```
for metric_name, metric in zip(['fit_time',
                                'test_precision',
                                'test_recall',
                                'test_f1',
                                'test_roc_auc'],
                               ['Fit Time',
                                'Precision',
                                'Recall',
                                'F1 Score',
                                'ROC AUC']):
    sns.boxplot(x='algo', y='value', #hue='algo',
        data=cv_results_inc_ens[cv_results_inc_ens.metric.eq(f'{metric_name}'
)])
    sns.stripplot(x='algo', y = 'value',
        data = cv_results_inc_ens[cv_results_inc_ens.metric.eq(f'{metric_name}
')],
        size = 5, linewidth = 1)
    plt.title(f'{metric} Algo Comparison', fontsize=12)
    plt.xlabel('Algorithm', fontsize=12)
    plt.ylabel(f'{metric}', fontsize=12)
    plt.xticks([0, 1, 2, 3, 4])
    plt.xticks(rotation=45)
    plt.show()
```


Fit Time Algo Comparison
Fit Time
2000
1500
1000
500
0
Gaussian Naive Bayes
SVM/Linear SVC
Logistic Regression
Decision Tree
Stacking
Algorithm


Precision Algo Comparison
Precision
0.900
0.875
0.850
0.825
0.800
0.775
0.750
0.725
Gaussian Naive Bayes
SVM/Linear SVC
Logistic Regression
Decision Tree
Stacking
Algorithm


Recall Algo Comparison
Recall
0.88
0.86
0.84
0.82
0.80
0.78
0.76
0.74
0.72
Gaussian Naive Bayes
SVM/Linear SVC
Logistic Regression
Decision Tree
Stacking
Algorithm


F1 Score Algo Comparison
F1 Score
0.88
0.86
0.84
0.82
0.80
0.78
0.76
0.74
0.72
Gaussian Naive Bayes
SVM/Linear SVC
Logistic Regression
Decision Tree
Stacking
Algorithm



Save our results

In [47]:
```
cv_results_inc_ens.to_csv('./data/cv-results-inc-ens.csv')
```

LIME for model interpretation

In [53]:
```
class_names=['World','Sports','Business','Tech/Sci']
```

Instantiate explainer

In [54]:
```
tab_explainer = lime.lime_tabular.LimeTabularExplainer(x_train_w2v.values,
    mode='classification', class_names=class_names)
```

Get explanations for: idx = Document

In [55]:
```
idx = 34
tab_exp_lreg = tab_explainer.explain_instance(x_test_w2v.values[idx], lreg.pre
dict_proba,
    num_features=10, top_labels=1)
print('Document id: %d' % idx)
print('Predicted class =',
    class_names[numpy.argmax(lreg.predict(numpy.array(x_test_w2v.values[idx]).
reshape(1, -1)))])
print('True class =', class_names[numpy.argmax(y_test[idx])])
```

```
Document id: 34
Predicted class = World
True class = World
```

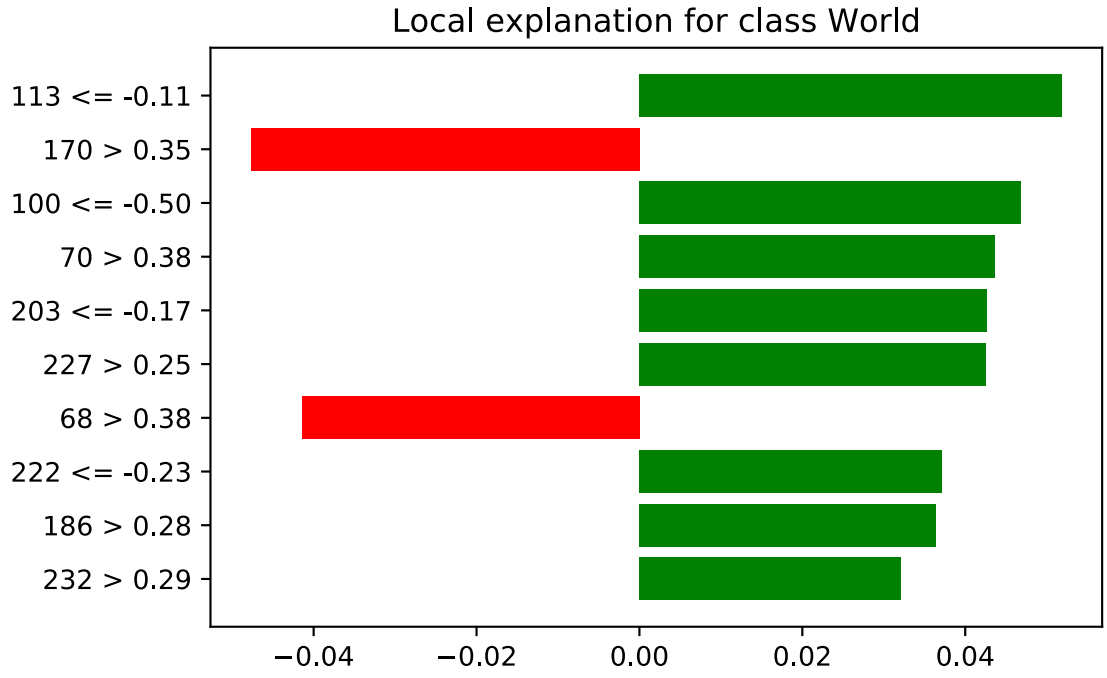
Get a text-only explanation

In [56]:
```
print ('\n'.join(map(str, tab_exp_lreg.as_list(label=numpy.argmax(y_test[idx
]))))))
```

```
('113 <= -0.11', 0.05183064476272634)
('170 > 0.35', -0.04768800249242717)
('100 <= -0.50', 0.04679713796889845)
('70 > 0.38', 0.04368779818621363)
('203 <= -0.17', 0.04269222543391017)
('227 > 0.25', 0.04247870611665241)
('68 > 0.38', -0.041380166579130497)
('222 <= -0.23', 0.03715680647558889)
('186 > 0.28', 0.036387157957656235)
('232 > 0.29', 0.032059681278521886)
```

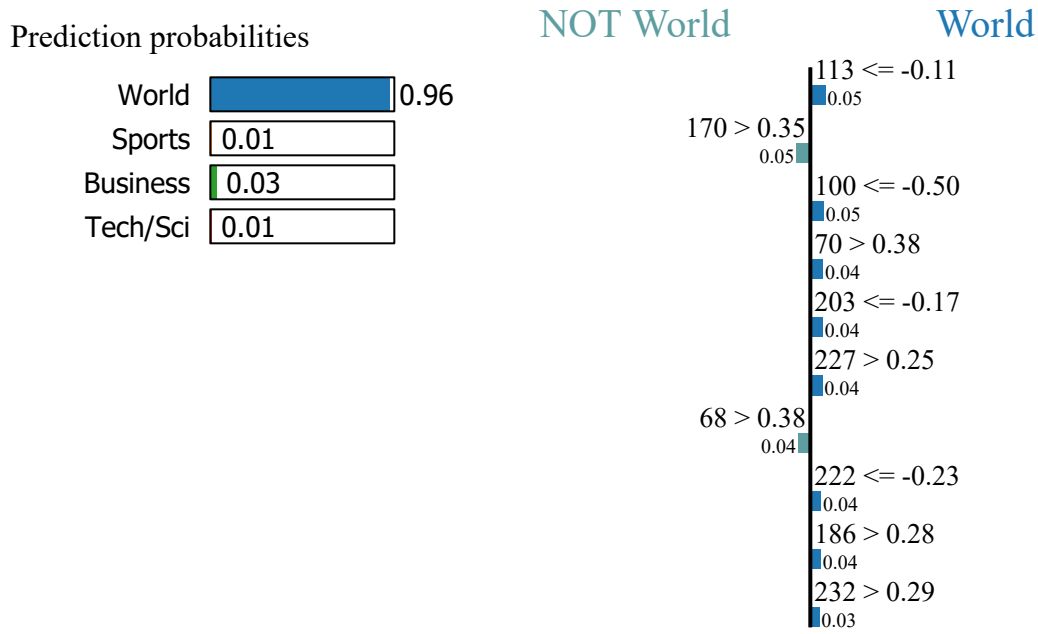
Get a graphical explanation of the predicted class

In [57]:
```
fig = tab_exp_lreg.as_pyplot_figure(label=numpy.argmax(y_test[idx]))
```



Get a graphical explanation with class probabilities

In [58]:
```
tab_exp_lreg.show_in_notebook()
```



Feature	Value
113	-1.22
170	0.64
100	-0.55
70	1.06
203	-0.24
227	0.28
68	0.42
222	-0.29
186	0.42

References - Code sample sources disclaimer:

Code for this project is either directly from (with some modification), or inspired by, but not limited to the following sources:

- Respective documentation and examples from each used API's doc/guide website
- Kelly Epley Naive Bayes: https://towardsdatascience.com/naive-bayes-document-classification-in-python-e33ff50f937e (https://towardsdatascience.com/naive-bayes-document-classification-in-python-e33ff50f937e)
- MLWhiz's excellent blogs about text classification and NLP: https://mlwhiz.com/blog/2018/12/17/text_classification/ (https://mlwhiz.com/blog/2018/12/17/text_classification/) https://mlwhiz.com/blog/2019/01/17/deeplearning_nlp_preprocess/ (https://mlwhiz.com/blog/2019/01/17/deeplearning_nlp_preprocess/) https://mlwhiz.com/blog/2019/02/08/deeplearning_nlp_conventional_methods/ (https://mlwhiz.com/blog/2019/02/08/deeplearning_nlp_conventional_methods/) https://www.kaggle.com/mlwhiz/conventional-methods-for-quora-classification/ (https://www.kaggle.com/mlwhiz/conventional-methods-for-quora-classification/)
- Christof Henkel preprocessing: https://www.kaggle.com/christofhenkel/how-to-preprocessing-when-using-embeddings (https://www.kaggle.com/christofhenkel/how-to-preprocessing-when-using-embeddings)
- datanizing GmbH: https://medium.com/@datanizing/modern-text-mining-with-python-part-1-of-5-introduction-cleaning-and-linguistics-647f9ec85b6a (https://medium.com/@datanizing/modern-text-mining-with-python-part-1-of-5-introduction-cleaning-and-linguistics-647f9ec85b6a)
- Datacamp wordcloud: https://www.datacamp.com/community/tutorials/wordcloud-python (https://www.datacamp.com/community/tutorials/wordcloud-python)
- Seaborn Pydata tutorials: https://seaborn.pydata.org/introduction.html#intro-plot-customization (https://seaborn.pydata.org/introduction.html#intro-plot-customization)
- Dipanjan S's tutorials: https://github.com/dipanjanS (https://github.com/dipanjanS)
- Analytics Vidhya: https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/ (https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/)
- Jason Brownlee's Feature Selection For Machine Learning in Python https://machinelearningmastery.com/feature-selection-machine-learning-python/ (https://machinelearningmastery.com/feature-selection-machine-learning-python/)
- Susan Li's Multi-class text classification with Scikit-learn: https://towardsdatascience.com/multi-class-text-classification-with-scikit-learn-12f1e60e0a9f (https://towardsdatascience.com/multi-class-text-classification-with-scikit-learn-12f1e60e0a9f)
- Vadim Smolyakov Ensemble Learning to Improve Machine Learning Results: https://blog.statsbot.co/ensemble-learning-d1dcd548e936 (https://blog.statsbot.co/ensemble-learning-d1dcd548e936)
- Udacity course video on Youtube UD120: https://www.youtube.com/watch?v=GdsLRKjjKLw (https://www.youtube.com/watch?v=GdsLRKjjKLw)
- Hyperparameter Tuning with Hyperopt https://towardsdatascience.com/automated-machine-learning-hyperparameter-tuning-in-python-dfda59b72f8a (https://towardsdatascience.com/automated-machine-learning-hyperparameter-tuning-in-python-dfda59b72f8a)
- Hyperparameter Tuning for Gaussian NB https://www.quora.com/Can-the-prior-in-a-naive-Bayes-be-considered-a-hyperparameter-and-tuned-for-better-accuracy (https://www.quora.com/Can-the-prior-in-a-naive-Bayes-be-considered-a-hyperparameter-and-tuned-for-better-accuracy)

- Hyperparameter Tuning for Decision Trees https://towardsdatascience.com/how-to-tune-a-decision-tree-f03721801680 (https://towardsdatascience.com/how-to-tune-a-decision-tree-f03721801680)
- Lime tutorial https://marcotcr.github.io/lime/tutorials/Lime%20-%20multiclass.html (https://marcotcr.github.io/lime/tutorials/Lime%20-%20multiclass.html)