In [1]:
```
# set tf 1.x for colab
%tensorflow_version 1.x
```

```
UsageError: Line magic function `%tensorflow_version` not found.
```

In [2]:
```
import warnings
warnings.filterwarnings('ignore', category=DeprecationWarning)
warnings.filterwarnings('ignore', category=FutureWarning)
```

Predict tags on StackOverflow with linear models

In this assignment you will learn how to predict tags for posts from StackOverflow (https://stackoverflow.com). To solve this task you will use multilabel classification approach.

Libraries

In this task you will need the following libraries:

- Numpy (http://www.numpy.org) — a package for scientific computing.
- Pandas (https://pandas.pydata.org) — a library providing high-performance, easy-to-use data structures and data analysis tools for the Python
- scikit-learn (http://scikit-learn.org/stable/index.html) — a tool for data mining and data analysis.
- NLTK (http://www.nltk.org) — a platform to work with natural language.

Data

The following cell will download all data required for this assignment into the folder `week1/data`.

In [3]:
```
import sys
sys.path.append("..")
from common.download_utils import download_week1_resources

download_week1_resources()
```

```
File data\train.tsv is already downloaded.
File data\validation.tsv is already downloaded.
File data\test.tsv is already downloaded.
File data\text_prepare_tests.tsv is already downloaded.
```

Grading

We will create a grader instance below and use it to collect your answers. Note that these outputs will be stored locally inside grader and will be uploaded to platform only after running submitting function in the last part of this assignment. If you want to make partial submission, you can run that cell any time you want.

In [4]:
```
from grader import Grader
```

In [5]:
```
grader = Grader()
```

Text preprocessing

For this and most of the following assignments you will need to use a list of stop words. It can be downloaded from *nltk*:

In [6]:
```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Xiaowei\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In this task you will deal with a dataset of post titles from StackOverflow. You are provided a split to 3 sets: *train*, *validation* and *test*. All corpora (except for *test*) contain titles of the posts and corresponding tags (100 tags are available). The *test* set is provided for Coursera's grading and doesn't contain answers. Upload the corpora using *pandas* and look at the data:

In [7]:
```
from ast import literal_eval
import pandas as pd
import numpy as np
```

In [8]:
```
def read_data(filename):
    data = pd.read_csv(filename, sep='\t')
    data['tags'] = data['tags'].apply(literal_eval)
    return data
```

In [9]:
```
train = read_data('data/train.tsv')
validation = read_data('data/validation.tsv')
test = pd.read_csv('data/test.tsv', sep='\t')
```

In [10]:
```
train.head()
```

Out[10]:

	title	tags
0	How to draw a stacked dotplot in R?	[r]
1	mysql select all records where a datetime fiel...	[php, mysql]
2	How to terminate windows phone 8.1 app	[c#]
3	get current time in a specific country via jquery	[javascript, jquery]
4	Configuring Tomcat to Use SSL	[java]

As you can see, *title* column contains titles of the posts and *tags* column contains the tags. It could be noticed that a number of tags for a post is not fixed and could be as many as necessary.

For a more comfortable usage, initialize *X_train*, *X_val*, *X_test*, *y_train*, *y_val*.

In [11]:
```
X_train, y_train = train['title'].values, train['tags'].values
X_val, y_val = validation['title'].values, validation['tags'].values
X_test = test['title'].values
```

In [12]:
```
type(train['title'])
```

Out[12]:
```
pandas.core.series.Series
```

One of the most known difficulties when working with natural data is that it's unstructured. For example, if you use it "as is" and extract tokens just by splitting the titles by whitespaces, you will see that there are many "weird" tokens like *3.5?*, *"Flip*, etc. To prevent the problems, it's usually useful to prepare the data somehow. In this task you'll write a function, which will be also used in the other assignments.

Task 1 (TextPrepare). Implement the function *text_prepare* following the instructions. After that, run the function *test_text_prepare* to test it on tiny cases and submit it to Coursera.

In [13]:
```
import re
```

In [14]:
```
REPLACE_BY_SPACE_RE = re.compile('[/(){}\[\]\|@,;]')
BAD_SYMBOLS_RE = re.compile('[^0-9a-z #+_]')
STOPWORDS = set(stopwords.words('english'))

def text_prepare(text):
    """
        text: a string

        return: modified initial string
    """
    text = text.lower() # lowercase text
    text = REPLACE_BY_SPACE_RE.sub(' ', text) # replace REPLACE_BY_SPACE_RE symbols by space in text
    text = BAD_SYMBOLS_RE.sub('', text) # delete symbols which are in BAD_SYMBOLS_RE from text

    # delete stopwords from text
    # convert multi space into single space, then split by space
    text = re.sub(r"\s+", " ", text)
    text = text.split(' ')
    text = [i for i in text if i not in STOPWORDS]
    text = ' '.join(text)

    return text
```

In [15]:
```
def test_text_prepare():
    examples = ["SQL Server - any equivalent of Excel's CHOOSE function?",
                "How to free c++ memory vector<int> * arr?"]
    answers = ["sql server equivalent excels choose function",
               "free c++ memory vectorint arr"]
    for ex, ans in zip(examples, answers):
        if text_prepare(ex) != ans:
            return "Wrong answer for the case: '%s'" % ex
    return 'Basic tests are passed.'
```

In [16]:
```
print(test_text_prepare())
```

```
Basic tests are passed.
```

Run your implementation for questions from file *text_prepare_tests.tsv* to earn the points.

In [17]:
```
prepared_questions = []
for line in open('data/text_prepare_tests.tsv', encoding='utf-8'):
    line = text_prepare(line.strip())
    prepared_questions.append(line)
text_prepare_results = '\n'.join(prepared_questions)

grader.submit_tag('TextPrepare', text_prepare_results)
```

```
Current answer for task TextPrepare is:
 sqlite php readonly
creating multiple textboxes dynamically
self one prefer javascript
save php date...
```

Now we can preprocess the titles using function *text_prepare* and making sure that the headers don't have bad symbols:

In [18]:
```
X_train = [text_prepare(x) for x in X_train]
X_val = [text_prepare(x) for x in X_val]
X_test = [text_prepare(x) for x in X_test]
```

In [19]:
```
X_train[:3]
```

Out[19]:
```
['draw stacked dotplot r',
 'mysql select records datetime field less specified value',
 'terminate windows phone 81 app']
```

For each tag and for each word calculate how many times they occur in the train corpus.

Task 2 (WordsTagsCount). Find 3 most popular tags and 3 most popular words in the train data and submit the results to earn the points.

In [20]:
```
# Dictionary of all tags from train corpus with their counts.
tags_counts = {}

## Use Python Counter from collections module to deal with the list data type, it's unsorted.
from collections import Counter

tags = [tag for sub_tags in y_train for tag in sub_tags]
tags_counts = dict(Counter(tags))

# Dictionary of all words from train corpus with their counts.
########################################
######### YOUR CODE HERE #############
########################################

words_counts = {}
map_result = map(lambda x: x.strip().split(' '), X_train)
list_result = [item for sublist in list(map_result) for item in sublist]

words_counts = dict(Counter(list_result))
```

In [21]:
```
len(tags_counts), tags_counts
```

Out[21]:
```
(100,
 {'r': 1727,
  'php': 13907,
  'mysql': 3092,
  'c#': 19077,
  'javascript': 19078,
  'jquery': 7510,
  'java': 18661,
  'ruby-on-rails': 3344,
  'ruby': 2326,
  'ruby-on-rails-3': 692,
  'json': 2026,
  'spring': 1346,
  'spring-mvc': 618,
  'codeigniter': 786,
  'class': 509,
  'html': 4668,
  'ios': 3256,
  'c++': 6469,
  'eclipse': 992,
  'python': 8940,
  'list': 693,
  'objective-c': 4338,
  'swift': 1465,
  'xaml': 438,
  'asp.net': 3939,
  'wpf': 1289,
  'multithreading': 1118,
  'image': 672,
  'performance': 512,
  'twitter-bootstrap': 501,
  'linq': 964,
  'xml': 1347,
  'numpy': 502,
  'ajax': 1767,
  'django': 1835,
  'laravel': 525,
  'android': 2818,
  'rest': 456,
  'asp.net-mvc': 1244,
  'web-services': 633,
  'string': 1573,
  'excel': 443,
  'winforms': 1468,
  'arrays': 2277,
  'c': 3119,
  'sockets': 579,
  'osx': 490,
  'entity-framework': 649,
  'mongodb': 350,
  'opencv': 401,
  'xcode': 900,
  'uitableview': 460,
  'algorithm': 419,
  'python-2.7': 421,
  'angularjs': 1353,
  'dom': 400,
  'swing': 759,
  '.net': 3872,
  'vb.net': 1918,
  'google-maps': 408,
  'hibernate': 807,
  'wordpress': 478,
  'iphone': 1909,
  'sql': 1272,
  'visual-studio': 574,
  'linux': 793,
  'facebook': 508,
  'database': 740,
  'file': 582,
  'generics': 420,
  'visual-studio-2010': 588,
  'regex': 1442,
  'html5': 842,
  'jsp': 680,
  'csv': 435,
  'forms': 872,
  'validation': 558,
  'parsing': 403,
  'function': 487,
  'pandas': 479,
  'sorting': 375,
  'qt': 451,
  'wcf': 389,
  'css': 1769,
  'date': 560,
  'node.js': 771,
  'sql-server': 585,
```

```
 'unit-testing': 449,
 'python-3.x': 379,
 'loops': 389,
 'windows': 838,
 'pointers': 350,
 'oop': 425,
 'datetime': 557,
 'servlets': 498,
 'session': 415,
 'cocoa-touch': 507,
 'apache': 441,
 'selenium': 431,
 'maven': 432})
```

In [22]:
```
print(len(words_counts))
print({k:v for k, v in list(words_counts.items())[:100]})
```

```
31497
{'draw': 173, 'stacked': 20, 'dotplot': 1, 'r': 896, 'mysql': 1572, 'select': 980, 'records': 175, 'datetime': 363,
'field': 778, 'less': 79, 'specified': 162, 'value': 3175, 'terminate': 33, 'windows': 1339, 'phone': 223, '81': 59,
'app': 1116, 'get': 4301, 'current': 523, 'time': 1213, 'specific': 691, 'country': 37, 'via': 590, 'jquery': 3293,
'configuring': 32, 'tomcat': 255, 'use': 2422, 'ssl': 145, 'awesome': 7, 'nested': 446, 'set': 1640, 'plugin': 270,
'add': 1611, 'new': 931, 'children': 79, 'tree': 209, 'various': 27, 'levels': 25, 'create': 1757, 'map': 496, 'jso
n': 1557, 'response': 396, 'ruby': 1164, 'rails': 1778, '3': 641, 'rspec': 144, 'test': 540, 'method': 2123, 'calle
d': 316, 'springboot': 27, 'catalina': 3, 'lifecycle': 9, 'exception': 944, 'import': 354, 'data': 3298, 'excel': 54
0, 'database': 1631, 'using': 8278, 'php': 5614, 'obtaining': 15, 'object': 2646, 'javalangclasst': 1, 'parameterize
d': 17, 'type': 1699, 'without': 1369, 'constructing': 8, 'class': 2558, 'q_uestion': 1, 'ipad': 82, 'selecting': 12
5, 'text': 1929, 'inside': 947, 'input': 1120, 'tap': 20, 'jquerys': 22, 'function': 2896, 'eclipse': 624, 'c++': 231
2, 'mingw': 39, 'lauch': 1, 'program': 584, 'terminated': 19, 'javascript': 4746, 'call': 1261, 'one': 1370, 'prototy
pe': 71, 'another': 1115, 'intersection': 17, 'list': 2137, 'sets': 57, 'longer': 40, 'able': 103, 'hide': 363, 'keyb
oard': 123, 'viewwilldisappear': 2, 'ios7': 26, 'fetch': 142, 'key': 776, 'swift': 934, 'change': 1441}
```

We are assuming that *tags_counts* and *words_counts* are dictionaries like `{'some_word_or_tag': frequency}`. After applying the sorting procedure, results will be look like this: `[('most_popular_word_or_tag', frequency), ('less_popular_word_or_tag', frequency), ...]`. The grader gets the results in the following format (two comma-separated strings with line break):

```
tag1,tag2,tag3
word1,word2,word3
```

Pay attention that in this assignment you should not submit frequencies or some additional information.

In [23]:
```
most_common_tags = sorted(tags_counts.items(), key=lambda x: x[1], reverse=True)[:3]
most_common_words = sorted(words_counts.items(), key=lambda x: x[1], reverse=True)[:3]

grader.submit_tag('WordsTagsCount', '%s\n%s' % (','.join(tag for tag, _ in most_common_tags),
                                                ','.join(word for word, _ in most_common_words)))
```

```
Current answer for task WordsTagsCount is:
 javascript,c#,java
using,php,java...
```

Transforming text to a vector

Machine Learning algorithms work with numeric data and we cannot use the provided text data "as is". There are many ways to transform text data to numeric vectors. In this task you will try to use two of them.

Bag of words

One of the well-known approaches is a *bag-of-words* representation. To create this transformation, follow the steps:

1. Find *N* most popular words in train corpus and numerate them. Now we have a dictionary of the most popular words.
2. For each title in the corpora create a zero vector with the dimension equals to *N*.
3. For each text in the corpora iterate over words which are in the dictionary and increase by 1 the corresponding coordinate.

Let's try to do it for a toy example. Imagine that we have *N* = 4 and the list of the most popular words is

```
['hi', 'you', 'me', 'are']
```

Then we need to numerate them, for example, like this:

```
{'hi': 0, 'you': 1, 'me': 2, 'are': 3}
```

And we have the text, which we want to transform to the vector:

```
'hi how are you'
```

For this text we create a corresponding zero vector

```
[0, 0, 0, 0]
```

And iterate over all words, and if the word is in the dictionary, we increase the value of the corresponding position in the vector:

```
'hi':  [1, 0, 0, 0]
'how': [1, 0, 0, 0] # word 'how' is not in our dictionary
'are': [1, 0, 0, 1]
'you': [1, 1, 0, 1]
```

The resulting vector will be

```
[1, 1, 0, 1]
```

Implement the described encoding in the function *my_bag_of_words* with the size of the dictionary equals to 5000. To find the most common words use train data. You can test your code using the function *test_my_bag_of_words*.

In [24]:
```
DICT_SIZE = 10000

####### YOUR CODE HERE #######
top_words = sorted(words_counts.items(), key=lambda x: x[1], reverse=True)[:DICT_SIZE]
top_words = list(map(lambda x: x[0], top_words))
enum_words_dict = dict(enumerate(top_words))
WORDS_TO_INDEX = {v:k for k, v in enum_words_dict.items()}

####### YOUR CODE HERE #######
INDEX_TO_WORDS = enum_words_dict
ALL_WORDS = WORDS_TO_INDEX.keys()

def my_bag_of_words(text, words_to_index, dict_size):
    """
        text: a string
        dict_size: size of the dictionary

        return a vector which is a bag-of-words representation of 'text'
    """
    result_vector = np.zeros(dict_size)
    ########################################
    ######### YOUR CODE HERE ##############
    ########################################
    words = text.split(' ')
    for word in words:
        id = words_to_index.get(word, -1)
        if id != -1:
            result_vector[id] = 1
    return result_vector
```

In [25]:
```
def test_my_bag_of_words():
    words_to_index = {'hi': 0, 'you': 1, 'me': 2, 'are': 3}
    examples = ['hi how are you']
    answers = [[1, 1, 0, 1]]
    for ex, ans in zip(examples, answers):
        if (my_bag_of_words(ex, words_to_index, 4) != ans).any():
            return "Wrong answer for the case: '%s'" % ex
    return 'Basic tests are passed.'
```

In [26]:
```
print(test_my_bag_of_words())
```

```
Basic tests are passed.
```

Now apply the implemented function to all samples (this might take up to a minute):

In [27]:
```
from scipy import sparse as sp_sparse
```

In [28]:
```
X_train_mybag = sp_sparse.vstack([sp_sparse.csr_matrix(my_bag_of_words(text, WORDS_TO_INDEX, DICT_SIZE)) for text in X
_train])
X_val_mybag = sp_sparse.vstack([sp_sparse.csr_matrix(my_bag_of_words(text, WORDS_TO_INDEX, DICT_SIZE)) for text in X_v
al])
X_test_mybag = sp_sparse.vstack([sp_sparse.csr_matrix(my_bag_of_words(text, WORDS_TO_INDEX, DICT_SIZE)) for text in X_
test])
print('X_train_mybag shape ', X_train_mybag.shape)
print('X_val_mybag shape ', X_val_mybag.shape)
print('X_test_mybag shape ', X_test_mybag.shape)
```

```
X_train_mybag shape  (100000, 10000)
X_val_mybag shape  (30000, 10000)
X_test_mybag shape  (20000, 10000)
```

In [29]:
```
print(X_train[0])
print(my_bag_of_words(X_train[0], WORDS_TO_INDEX, DICT_SIZE))
print(X_train_mybag[0])
```

```
draw stacked dotplot r
[0. 0. 0. ... 0. 0. 0.]
  (0, 98)       1.0
  (0, 605)      1.0
  (0, 2854)     1.0
```

As you might notice, we transform the data to sparse representation, to store the useful information efficiently. There are many types (https://docs.scipy.org/doc/scipy/reference/sparse.html) of such representations, however sklearn algorithms can work only with csr (https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csr_matrix.html#scipy.sparse.csr_matrix) matrix, so we will use this one.

Task 3 (BagOfWords). For the 11th row in *X_train_mybag* find how many non-zero elements it has. In this task the answer (variable *non_zero_elements_count*) should be a number, e.g. 20.

In [30]:
```
row = X_train_mybag[10].toarray()[0]
non_zero_elements_count = int(sum(row)) ####### YOUR CODE HERE #######

grader.submit_tag('BagOfWords', non_zero_elements_count)
```

```
Current answer for task BagOfWords is:
 7...
```

TF-IDF

The second approach extends the bag-of-words framework by taking into account total frequencies of words in the corpora. It helps to penalize too frequent words and provide better features space.

Implement function *tfidf_features* using class TfidfVectorizer (http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html) from *scikit-learn*. Use *train* corpus to train a vectorizer. Don't forget to take a look into the arguments that you can pass to it. We suggest that you filter out too rare words (occur less than in 5 titles) and too frequent words (occur more than in 90% of the titles). Also, use bigrams along with unigrams in your vocabulary.

In [31]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [32]:
```
def tfidf_features(X_train, X_val, X_test):
    """
        X_train, X_val, X_test — samples
        return TF-IDF vectorized representation of each sample and vocabulary
    """
    # Create TF-IDF vectorizer with a proper parameters choice
    # Fit the vectorizer on the train set
    # Transform the train, test, and val sets and return the result


    ####### YOUR CODE HERE #######
    tfidf_vectorizer = TfidfVectorizer(min_df = 5, max_df = 0.9, ngram_range=(1,2), token_pattern = '(\S+)')

    ########################################
    ######### YOUR CODE HERE #############
    ########################################
    tfidf_vectorizer = tfidf_vectorizer.fit(X_train)
    X_train = tfidf_vectorizer.transform(X_train)
    X_val = tfidf_vectorizer.transform(X_val)
    X_test = tfidf_vectorizer.transform(X_test)

    return X_train, X_val, X_test, tfidf_vectorizer.vocabulary_
```

Once you have done text preprocessing, always have a look at the results. Be very careful at this step, because the performance of future models will drastically depend on it.

In this case, check whether you have c++ or c# in your vocabulary, as they are obviously important tokens in our tags prediction task:

In [33]:
```
X_train_tfidf, X_val_tfidf, X_test_tfidf, tfidf_vocab = tfidf_features(X_train, X_val, X_test)
tfidf_reversed_vocab = {i:word for word,i in tfidf_vocab.items()}
```

In [34]:
```
X_train_tfidf.shape
```

Out[34]:
```
(100000, 18300)
```

In [35]:
```
print('Text:\n', X_train[0])
print('After BOW, dictionary size 10000, compressed spark row matrix:\n', X_train_mybag [0])
print('After tfidf, minimum document frequency 5:\n', X_train_tfidf[0])
```

```
Text:
 draw stacked dotplot r
After BOW, dictionary size 10000, compressed spark row matrix:
   (0, 98)      1.0
  (0, 605)      1.0
  (0, 2854)     1.0
After tfidf, minimum document frequency 5:
   (0, 14941)   0.7126565202061851
  (0, 12748)    0.4309937630129157
  (0, 4792)     0.5535025387941576
```

In [36]:
```
######### YOUR CODE HERE #############
print('c++' in tfidf_reversed_vocab.values())
print('c#' in tfidf_reversed_vocab.values())
```

```
True
True
```

If you can't find it, we need to understand how did it happen that we lost them? It happened during the built-in tokenization of TfidfVectorizer. Luckily, we can influence on this process. Get back to the function above and use '(\S+)' regexp as a *token_pattern* in the constructor of the vectorizer.

Now, use this transormation for the data and check again.

In [37]:
```
######### YOUR CODE HERE #############
print('c++' in tfidf_reversed_vocab.values())
print('c#' in tfidf_reversed_vocab.values())
```

```
True
True
```

In [38]:
```
print(X_train[1])
print(X_train_mybag[1])
print(X_train_tfidf[1])
```

```
mysql select records datetime field less specified value
  (0, 13)	1.0
  (0, 36)	1.0
  (0, 84)	1.0
  (0, 116)	1.0
  (0, 292)	1.0
  (0, 599)	1.0
  (0, 645)	1.0
  (0, 1142)	1.0
  (0, 17129)	0.18110148646398527
  (0, 14801)	0.2999430853319639
  (0, 14054)	0.40893120409824163
  (0, 14019)	0.22859508855051244
  (0, 13008)	0.2975359437533552
  (0, 10426)	0.36213766165290934
  (0, 10394)	0.2088886377002491
  (0, 9077)	0.32871667093872164
  (0, 5815)	0.23823684465290781
  (0, 4093)	0.3963922496423734
  (0, 4089)	0.2692803496632626
```

MultiLabel classifier

As we have noticed before, in this task each example can have multiple tags. To deal with such kind of prediction, we need to transform labels in a binary form and the prediction will be a mask of 0s and 1s. For this purpose it is convenient to use [MultiLabelBinarizer (http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MultiLabelBinarizer.html)](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MultiLabelBinarizer.html) from *sklearn*.

In [39]:
```
from sklearn.preprocessing import MultiLabelBinarizer
```

In [40]:
```
print(y_train[0])
```

```
['r']
```

In [41]:
```
mlb = MultiLabelBinarizer(classes=sorted(tags_counts.keys()))
y_train = mlb.fit_transform(y_train)
y_val = mlb.fit_transform(y_val)
```

In [42]:
```
print(mlb)
print(y_train[0])
```

```
MultiLabelBinarizer(classes=['.net', 'ajax', 'algorithm', 'android',
                             'angularjs', 'apache', 'arrays', 'asp.net',
                             'asp.net-mvc', 'c', 'c#', 'c++', 'class',
                             'cocoa-touch', 'codeigniter', 'css', 'csv',
                             'database', 'date', 'datetime', 'django', 'dom',
                             'eclipse', 'entity-framework', 'excel', 'facebook',
                             'file', 'forms', 'function', 'generics', ...])
[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 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 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 0 0]
```

Implement the function *train_classifier* for training a classifier. In this task we suggest to use One-vs-Rest approach, which is implemented in [OneVsRestClassifier (http://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsRestClassifier.html)](http://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsRestClassifier.html) class. In this approach *k* classifiers (= number of tags) are trained. As a basic classifier, use [LogisticRegression (http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html)](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html). It is one of the simplest methods, but often it performs good enough in text classification tasks. It might take some time, because a number of classifiers to train is large.

In [43]:
```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import LogisticRegression, RidgeClassifier
```

In [44]:
```
def train_classifier(X_train, y_train):
    """
      X_train, y_train — training data
      
      return: trained classifier
    """
    
    # Create and fit LogisticRegression wraped into OneVsRestClassifier.

    ######################################
    ######### YOUR CODE HERE #############
    ######################################
    return OneVsRestClassifier(LogisticRegression(random_state=0), n_jobs=-1).fit(X_train, y_train)
```

Train the classifiers for different data transformations: *bag-of-words* and *tf-idf*.

In [45]:
```
%%time
classifier_mybag = train_classifier(X_train_mybag, y_train)
classifier_tfidf = train_classifier(X_train_tfidf, y_train)
```

```
Wall time: 22.3 s
```

In [46]:
```
y_train_predicted_labels_tfidf = classifier_tfidf.predict(X_train_tfidf)
y_train_predicted_scores_tfidf = classifier_tfidf.decision_function(X_train_tfidf)
y_train_pred_inversed = mlb.inverse_transform(y_train_predicted_labels_tfidf)
```

In [47]:
```
print('%d samples, %d have predicted labels.' %
      (len(y_train_pred_inversed), len(list(filter(lambda x: x != (), y_train_pred_inversed)))))
```

```
100000 samples, 75132 have predicted labels.
```

Now you can create predictions for the data. You will need two types of predictions: labels and scores.

In [48]:
```
y_val_predicted_labels_mybag = classifier_mybag.predict(X_val_mybag)
y_val_predicted_scores_mybag = classifier_mybag.decision_function(X_val_mybag)

y_val_predicted_labels_tfidf = classifier_tfidf.predict(X_val_tfidf)
y_val_predicted_scores_tfidf = classifier_tfidf.decision_function(X_val_tfidf)
```

Now take a look at how classifier, which uses TF-IDF, works for a few examples:

In [49]:
```
y_val_pred_inversed = mlb.inverse_transform(y_val_predicted_labels_tfidf)
y_val_inversed = mlb.inverse_transform(y_val)
for i in range(3):
    print('Title:\t{}\nTrue labels:\t{}\nPredicted labels:\t{}\n\n'.format(
        X_val[i],
        ','.join(y_val_inversed[i]),
        ','.join(y_val_pred_inversed[i])
    ))
```

```
Title:  odbc_exec always fail
True labels:    php,sql
Predicted labels:


Title:  access base classes variable within child class
True labels:    javascript
Predicted labels:


Title:  contenttype application json required rails
True labels:    ruby,ruby-on-rails
Predicted labels:       json,ruby-on-rails
```

In [50]:
```
print('%d samples, %d have predicted labels.' %
      (len(y_val_predicted_labels_tfidf), len(list(filter(lambda x: x != (), y_val_pred_inversed)))))
```

```
30000 samples, 22193 have predicted labels.
```

Now, we would need to compare the results of different predictions, e.g. to see whether TF-IDF transformation helps or to try different regularization techniques in logistic regression. For all these experiments, we need to setup evaluation procedure.

Evaluation

To evaluate the results we will use several classification metrics:

- Accuracy (http://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html)
- F1-score (http://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html)
- Area under ROC-curve (http://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html)
- Area under precision-recall curve (http://scikit-learn.org/stable/modules/generated/sklearn.metrics.average_precision_score.html#sklearn.metrics.average_precision_score)

Make sure you are familiar with all of them. How would you expect the things work for the multi-label scenario? Read about micro/macro/weighted averaging following the sklearn links provided above.

In [51]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import average_precision_score
from sklearn.metrics import recall_score
```

Implement the function *print_evaluation_scores* which calculates and prints to stdout:

- *accuracy*
- *F1-score macro/micro/weighted*
- *Precision macro/micro/weighted*

In [52]:
```
%time accuracy_score(y_val, y_val_predicted_labels_mybag)
```

```
Wall time: 114 ms
```

Out[52]:
```
0.3624
```

In [53]:
```
def print_evaluation_scores(y_val, predicted):

    ######################################
    ######### YOUR CODE HERE #############
    ######################################
    acc = accuracy_score(y_val, predicted)
    print('Accuracy score: %f' % acc)
    f1_macro = f1_score(y_val, predicted, average = 'macro')
    f1_micro = f1_score(y_val, predicted, average = 'micro')
    f1_weighted = f1_score(y_val, predicted, average = 'weighted')
    print('F1-score macro/micro/weighted: %f, %f, %f' % (f1_macro, f1_micro, f1_weighted))

    pre_macro = average_precision_score(y_val, predicted, average = 'macro')
    pre_micro = average_precision_score(y_val, predicted, average = 'micro')
    pre_weighted = average_precision_score(y_val, predicted, average = 'weighted')
    print('Precision score macro/micro/weighted: %f, %f, %f' % (pre_macro, pre_micro, pre_weighted))
```

In [54]:
```
# # Set below attribute to ignore error divide by 0
# np.seterr(divide='ignore', invalid='ignore')
```

In [55]:
```
print('Bag-of-words')
print_evaluation_scores(y_val, y_val_predicted_labels_mybag)
print('\n')

print('Tfidf')
print_evaluation_scores(y_val, y_val_predicted_labels_tfidf)
```

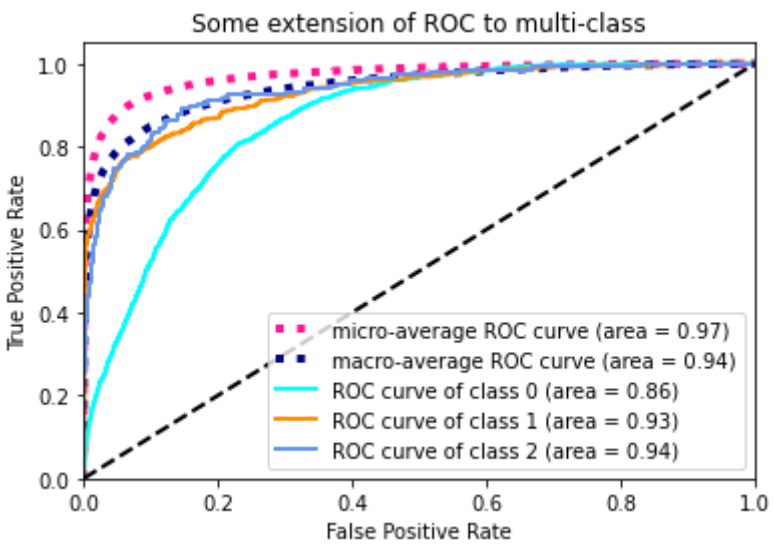
```
Bag-of-words
Accuracy score: 0.362400
F1-score macro/micro/weighted: 0.508632, 0.675459, 0.653083
Precision score macro/micro/weighted: 0.349302, 0.486433, 0.515752


Tfidf
Accuracy score: 0.334000
F1-score macro/micro/weighted: 0.445482, 0.641778, 0.614293
Precision score macro/micro/weighted: 0.301823, 0.456963, 0.485059
```

You might also want to plot some generalization of the [ROC curve (http://scikit-learn.org/stable/modules/model_evaluation.html#receiver-operating-characteristic-roc)](http://scikit-learn.org/stable/modules/model_evaluation.html#receiver-operating-characteristic-roc) for the case of multi-label classification. Provided function *roc_auc* can make it for you. The input parameters of this function are:

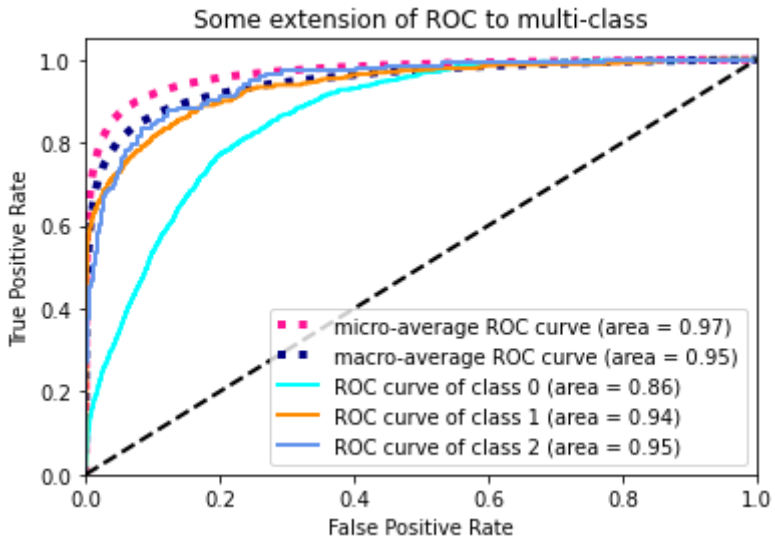
- true labels
- decision functions scores
- number of classes

In [56]:
```
from metrics import roc_auc
%matplotlib inline
```

In [57]:
```
n_classes = len(tags_counts)
roc_auc(y_val, y_val_predicted_scores_mybag, n_classes)
```



In [58]:
```
n_classes = len(tags_counts)
roc_auc(y_val, y_val_predicted_scores_tfidf, n_classes)
```



Task 4 (MultilabelClassification). Once we have the evaluation set up, we suggest that you experiment a bit with training your classifiers. We will use *F1-score weighted* as an evaluation metric. Our recommendation:

- compare the quality of the bag-of-words and TF-IDF approaches and chose one of them.
- for the chosen one, try *L1* and *L2*-regularization techniques in Logistic Regression with different coefficients (e.g. C equal to 0.1, 1, 10, 100).

You also could try other improvements of the preprocessing / model, if you want.

In [59]:
```
LogisticRegression.get_params(LogisticRegression).keys()
```

Out[59]:
```
dict_keys(['C', 'class_weight', 'dual', 'fit_intercept', 'intercept_scaling', 'l1_ratio', 'max_iter', 'multi_class',
'n_jobs', 'penalty', 'random_state', 'solver', 'tol', 'verbose', 'warm_start'])
```

In [60]:
```
%%time
########################################
######### YOUR CODE HERE #############
########################################
from sklearn.model_selection import GridSearchCV

C_parameters = {'estimator__C': [0.1, 1, 10, 100], 'estimator__penalty':['none', 'l2']}
base_clf = OneVsRestClassifier(LogisticRegression(random_state=0), n_jobs=-1)
grid_obj = GridSearchCV(base_clf, C_parameters, scoring='f1_weighted', cv=3)
grid_obj.fit(X_train_mybag, y_train)
```

```
Wall time: 3min 28s
```

Out[60]:
```
GridSearchCV(cv=3,
             estimator=OneVsRestClassifier(estimator=LogisticRegression(random_state=0),
                                           n_jobs=-1),
             param_grid={'estimator__C': [0.1, 1, 10, 100],
                         'estimator__penalty': ['none', 'l2']},
             scoring='f1_weighted')
```

In [61]:
```
model_best = grid_obj.best_estimator_
print(model_best)
```

```
OneVsRestClassifier(estimator=LogisticRegression(C=10, random_state=0),
                    n_jobs=-1)
```

When you are happy with the quality, create predictions for *test* set, which you will submit to Coursera.

In [62]:
```
######### YOUR CODE HERE #############
test_predictions = model_best.predict(X_test_mybag)
test_pred_inversed = mlb.inverse_transform(test_predictions)

test_predictions_for_submission = '\n'.join('%i\t%s' % (i, ','.join(row)) for i, row in enumerate(test_pred_inversed))
grader.submit_tag('MultilabelClassification', test_predictions_for_submission)
```

```
Current answer for task MultilabelClassification is:
 0       mysql,php
1        javascript,jquery
2
3        javascript,jquery
4        android
5        php,xml
6        ajax,json,php,web-servi...
```

In [63]:
```
print_evaluation_scores(y_val, model_best.predict(X_val_mybag))
```

```
Accuracy score: 0.354533
F1-score macro/micro/weighted: 0.514385, 0.674361, 0.658011
Precision score macro/micro/weighted: 0.335513, 0.473366, 0.507903
```

Analysis of the most important features

Finally, it is usually a good idea to look at the features (words or n-grams) that are used with the largest weigths in your logistic regression model.

Implement the function *print_words_for_tag* to find them. Get back to sklearn documentation on OneVsRestClassifier (http://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsRestClassifier.html) and LogisticRegression (http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html) if needed.

In [64]:
```
def print_words_for_tag(classifier, tag, tags_classes, index_to_words, all_words):
    """
        classifier: trained classifier
        tag: particular tag
        tags_classes: a list of classes names from MultiLabelBinarizer
        index_to_words: index_to_words transformation
        all_words: all words in the dictionary

        return nothing, just print top 5 positive and top 5 negative words for current tag
    """
    print('Tag:\t{}'.format(tag))

    # Extract an estimator from the classifier for the given tag.
    # Extract feature coefficients from the estimator.

    ######################################
    ######### YOUR CODE HERE #############
    ######################################

    tag_index = tags_classes.index(tag)
    tag_coef = classifier.coef_[tag_index]

    top_indices = np.argsort(tag_coef)[:-6:-1]
    bottom_indices = np.argsort(tag_coef)[:5]

    top_positive_words = list(map(lambda x: index_to_words[x], top_indices)) # top-5 words sorted by the coefficiens.
    top_negative_words = list(map(lambda x: index_to_words[x], bottom_indices)) # bottom-5 words  sorted by the coeffi
cients.
    print('Top positive words:\t{}'.format(', '.join(top_positive_words)))
    print('Top negative words:\t{}\n'.format(', '.join(top_negative_words)))
```

In [65]:
```
print_words_for_tag(classifier_tfidf, 'c', mlb.classes, tfidf_reversed_vocab, ALL_WORDS)
print_words_for_tag(classifier_tfidf, 'c++', mlb.classes, tfidf_reversed_vocab, ALL_WORDS)
print_words_for_tag(classifier_tfidf, 'linux', mlb.classes, tfidf_reversed_vocab, ALL_WORDS)
```

```
Tag:    c
Top positive words:     c, malloc, scanf, printf, gcc
Top negative words:     java, php, python, javascript, c#

Tag:    c++
Top positive words:     c++, qt, boost, mfc, opencv
Top negative words:     java, php, python, javascript, c#

Tag:    linux
Top positive words:     linux, ubuntu, c, address, signal
Top negative words:     javascript, c#, jquery, array, method
```

Authorization & Submission

To submit assignment parts to Cousera platform, please, enter your e-mail and token into variables below. You can generate token on this programming assignment page. **Note:** Token expires 30 minutes after generation.

In [56]:
```
grader.status()
```

```
You want to submit these parts:
Task TextPrepare:
 sqlite php readonly
creating multiple textboxes dynamically
self one prefer javascript
save php date...
Task WordsTagsCount:
 javascript,c#,java
using,php,java...
Task BagOfWords:
 7...
Task MultilabelClassification:
 0      mysql,php
1       javascript
2
3       javascript,jquery
4       android,java
5       php,xml
6       json,web-services
7       java,...
```

In [57]:
```
STUDENT_EMAIL = 'lxwvictor@gmail.com'
STUDENT_TOKEN = 'hafdw3llOOR4SYCQ'
grader.status()
```

```
You want to submit these parts:
Task TextPrepare:
 sqlite php readonly
creating multiple textboxes dynamically
self one prefer javascript
save php date...
Task WordsTagsCount:
 javascript,c#,java
using,php,java...
Task BagOfWords:
 7...
Task MultilabelClassification:
 0      mysql,php
1       javascript
2
3       javascript,jquery
4       android,java
5       php,xml
6       json,web-services
7       java,...
```

If you want to submit these answers, run cell below

In [58]:
```
grader.submit(STUDENT_EMAIL, STUDENT_TOKEN)
```

```
Submitted to Coursera platform. See results on assignment page!
```

In [ ]:
```

```