**1)Dataset :**

**Stack over flow tags predictions (Title , tags)**

**EX…**

How to draw a stacked dotplot in R? ['r']

mysql select all records where a datetime field is less than a specified value ['php', 'mysql']

How to terminate windows phone 8.1 app ['c#']

**Reshape of all data :**

Train (100,000 , 2 )

Validation (30,000 , 2)

Test (20,000 , 1)

**2)Clean the data :**

Replace all ('[/(){}\[\]\|@,;]') to space and this is our RE ('[^0-9a-z #+\_]') and remove all **stopwords**

BAD\_SYMBOLS\_RE = re.compile('[^0-9a-z #+\_]')

STOPWORDS = set(stopwords.words('english'))

**3)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.

**4) tfidf\_vectorizer**

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.

Function *tfidf\_features* using class [TfidfVectorizer](http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html) from *scikit-learn*. Using *train* corpus to train a vectorizer.

Once we 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:

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.

**5)Model**

### MultiLabel classifier

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) from sklearn.

**train\_classifier**

Function train\_classifier for training a classifier.We use One-vs-Rest approach, which is implemented in [OneVsRestClassifier](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). 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.

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

**6)Evaluation**

Bag-of-words :

Accuracy Score 0.3578

F1 Score 0.6486667031464047

Precision Score 0.3444038613007691

Tf-idf :

Accuracy Score 0.33393333333333336

F1 Score 0.6142668931088263

Precision Score 0.30181976655232984

|  |  |  |
| --- | --- | --- |
| Model | Acc-mybag | Acc-tfidf |
| OneVsRestClassifier(SGDClassifier(loss='log', penalty='l1'), n\_jobs=-1) | Accuracy Score: 0.329  F1 Score : 0.612623163  Precision Score : 0.331168216 | Accuracy Score : 0.274  F1 Score : 0.55171  Precision Score : 0.272855 |
| lr = LogisticRegression(solver='newton-cg',C=C, penalty=penalty,n\_jobs=-1)  # lr.fit(X\_train, y\_train)  ovr = OneVsRestClassifier(lr) | Accuracy Score : 0.3578  F1 Score : 0.6486667Precision Score : 0.344403861 | Accuracy Score : 0.3339333  F1 Score : 0.61426689310  Precision Score : 0.3018 |