In [0]:

```
## Reading Math Papers
```

In [0]:

```
## Read the math papers from 2016 to 2018 and store it in a file
import urllib
url = 'http://export.arxiv.org/oai2?verb=ListRecords&set=math&from=2016-01-01&until=2018-11-31&metadataPrefi
x=arXiv'
data = urllib.request.urlopen(url).read()

m = open('math1', 'wb')
m.write(data)
```

Out[0]:

```
1692561
```

In [0]:

```
## Extract the title and abstract from papers - Read from finance1 to finance2
!xml_grep 'title|abstract' math1  > math2.txt
```

In [0]:

```
## Remove Junk lines , here we remove first 3 lines and last 3 lines which are not necessary
!cat math2.txt | tail -n +4 | head -n -3 > math3.txt
```

In [0]:

```
## Reading packages for Text classification
from sklearn import model_selection, preprocessing, linear_model, naive_bayes, metrics, svm
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn import decomposition, ensemble

import pandas, numpy, string
from keras.preprocessing import text, sequence
from keras import layers, models, optimizers
from nltk import word_tokenize
from nltk.corpus import stopwords
import sklearn
#import sklearn_crfsuite
#from sklearn_crfsuite import scorers
#from sklearn_crfsuite import metrics
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.decomposition import TruncatedSVD
from sklearn.metrics import accuracy_score
from sklearn import metrics
```

In [0]:

```
## Stopwords import and removal
import nltk
from nltk.corpus import stopwords

nltk.download('stopwords')
stopwords = set(stopwords.words('english'))
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [0]:

```
# load the dataset # dataset contains combined labels and text from all training papers
data = open('labeled_sentences (1).txt').read()[:-2]
labels, texts = [], []
for i, line in enumerate(data.split("\n")):
    content = line.split()
    #print(content)
    labels.append(content[0])
    filtered_sentence = [w.lower() for w in content[1:] if not w in stopwords]
    texts.append(filtered_sentence)

# create a dataframe using texts and lables
trainDF = pandas.DataFrame()
trainDF['text'] = texts
trainDF['label'] = labels
print(trainDF['label'].unique())
trainDF.head(2)
```

```
['MISC' 'AIMX' 'OWNX' 'CONT' 'BASE']
```

Out[0]:

	text	label
0	[minimum, description, length, principle, onli...	MISC
1	[underlying, model, class, discrete,, total, e...	MISC

In [0]:

```
## Used the obtained dataset for training
train_x, valid1_x, train_y, valid1_y = model_selection.train_test_split(trainDF['text'], trainDF['label'],te
st_size=0)
```

In [0]:

```
## Convert from list to string
tempp = []

for item in train_x:
    tempp.append(" ".join(item))
#print(len(train_x))

#tempp1 =[]
#for item1 in valid_x:
    #tempp1.append(" ".join(item1))

#print(len(tempp1))

temp =[]
temp_len=0
for item2 in texts:
    temp.append(" ".join(item2))
    temp_len = temp_len+len(texts)
print(len(temp))
print(temp_len)
print(type(temp))
```

```
19162
367182244
<class 'list'>
```

In [0]:

```
# create a count vectorizer object
count_vect = CountVectorizer(analyzer='word', token_pattern=r'\w{1,}')
count_vect.fit(temp)

# transform the training and validation data using count vectorizer object
xtrain_count =  count_vect.transform(tempp)
```

In [0]:

```
## Create a classifier
import csv
trainDF2 = pandas.DataFrame()

def train_model(classifier, feature_vector_train, label, feature_vector_valid, is_neural_net=False):
    # fit the training dataset on the classifier
    #std_clf = make_pipeline(StandardScaler(with_mean=False), TruncatedSVD(100), MultinominalNB())
    #std_clf.fit(feature_vector_train, label)
    classifier.fit(feature_vector_train, label)

    # predict the labels on validation dataset
    #predictions = classifier.predict(feature_vector_valid)
    predictions = classifier.predict(feature_vector_valid)
    return predictions
    #tt = classifier.predict(feature_vector_valid)
    #labels3 = classifier.predict(feature_vector_valid)

    #trainDF2['labels'] = labels3
    #trainDF2['text']= valid_x
    #print(trainDF2)
```

In [0]:

```
## Read title and abstracts and loop through them
import re
global_list = []
title_list =[]

test = open("math3.txt",'r').read().split("</abstract>")
#print(test[1])
for idx,i in enumerate(test):
  title = re.findall(r"(?<=<title>).*(?=</title>)",i.replace("\n",""))
  #print(title)
  abstract = re.findall(r"(?<=<abstract>).*",i.replace("\n",""))
  #print(abstract[0].replace("\n",""))
  nlist = re.split(r"(?:(?<=[^i]\.)|\.(?=[^e]))",abstract[0].replace('"',"").replace('\n',''))
  #temp_abs = re.sub(r"((?<=[^i]\.)|\.(?=[^e]))","\n",abstract[0])
  #print(abstract)
  #temp_str = temp_abs.split("\n")
  #print(temp_str[0])
  #print(nlist[1])
  global_list.append(nlist)
  title_list.append(title)
  #print(global_list)

  if idx >50:
    #print(global_list)
    break
  #print(abstract[0])
  #nlist = re.split(r"(?:(?<=[^i]\.)|\.(?=[^e]))",str(abstract))

  #print(nlist[1])

  #tempp1 =[]
  '''
  for idx, item1 in enumerate(nlist):

    if idx > 1 :
      break;
      print(item1)
      tempp1.append(" ".join(item1))
    #print(tempp1)

    xvalid_count =  count_vect.transform(tempp1)
    for item in nlist:
      print(item)
      valid_x = item
      #accuracy = train_model(naive_bayes.MultinomialNB(), xtrain_count, train_y, xvalid_count)

  '''
  #print(global_list[0])
  #print(global_list[1])
  #print(global_list[2])
  #for idx, item1 in enumerate(global_list) :
  #  if idx > 1:
  #    break
  #  print(item1)
    #tempp1.append(" ".join(item1))
    #xvalid_count =  count_vect.transform(tempp1)
    #accuracy = train_model(naive_bayes.MultinomialNB(), xtrain_count, train_y, xvalid_count)
```

```
/usr/lib/python3.6/re.py:212: FutureWarning: split() requires a non-empty pattern match.
  return _compile(pattern, flags).split(string, maxsplit)
```

In [0]:

```
## Print triples from data

#print(global_list[1])
for idx, (item, title) in enumerate(zip(global_list, title_list)):



  #print(item)
  valid_x = item
  xvalid_count =  count_vect.transform(valid_x)
  accuracy = train_model(linear_model.LogisticRegression(), xtrain_count, train_y, xvalid_count)
  #print("\n\n")
  if idx>1:
    break

  title_id = hash(str(title))
  abstract_id = hash(str(item))
  line1 = "<https://w3id.org/skg/articles/" + str(title_id) + "> <http://xmlns.com/foaf/0.1/name>" + '"' + "
 ".join(title) + '"' +"."
  line2 = "<https://w3id.org/skg/articles/" + str(title_id) + "> <http://purl.org/dc/terms/abstract> <http:/
/purl.org/dc/terms/abstract/" + str(abstract_id)+ ">"
  line3 = "<https://w3id.org/skg/articles/" + str(abstract_id) +"><http://purl.org/dc/terms/abstract/text>"
+ '"' + " ".join(item) + '"'
  print(line1,line2,line3,sep ="\n")
  for acc,element in zip(accuracy,item):
    print('<http://purl.org/dc/terms/abstract/{} > "{}"'.format(acc, element))
    #line4 = ("<http://purl.org/dc/terms/abstract/" + str(acc) + ">" + '"' + str(element) + '"' )
```

<https://w3id.org/skg/articles/-2688487514614604503> <http://xmlns.com/foaf/0.1/name>"Monoid generalizations of the Richard Thompson groups".
<https://w3id.org/skg/articles/-2688487514614604503> <http://purl.org/dc/terms/abstract> <http://purl.org/dc/terms/abstract/-4269569602989116255>
<https://w3id.org/skg/articles/-4269569602989116255><http://purl.org/dc/terms/abstract/text>"The groups G_{k,1} of Richard Thompson and Graham Higman can be generalizedin a natural way to monoids, that we call M_{k,1}, and to inverse monoids,called Inv_{k,1}; this is done by simply generalizing bijections to partialfunctions or partial injective functions The monoids M_{k,1} have connectionswith circuit complexity (studied in another paper) Here we prove that M_{k,1} and Inv_{k,1} are congruence-simple for all k Their Green relations J and Dare characterized: M_{k,1} and Inv_{k,1} are J-0-simple, and they have k-1non-zero D-classes They are submonoids of the multiplicative part of the Cuntzalgebra O_k They are finitely generated, and their word problem over anyfinite generating set is in P Their word problem is coNP-complete over certaininfinite generating sets Changes in this version: Section 4 has been thoroughly revised, and errorshave been corrected; however, the main results of Section 4 do not change Sections 1, 2, and 3 are unchanged, except for the proof of Theorem 2 3, whichwas incomplete; a complete proof was published in the Appendix of reference[6], and is also given here."
<http://purl.org/dc/terms/abstract/OWNX > " The groups G_{k,1} of Richard Thompson and Graham Higman can be generalizedin a natural way to monoids, that we call M_{k,1}, and to inverse monoids,called Inv_{k,1}; this is done by simply generalizing bijections to partialfunctions or partial injective functions"
<http://purl.org/dc/terms/abstract/AIMX > " The monoids M_{k,1} have connectionswith circuit complexity (studied in another paper)"
<http://purl.org/dc/terms/abstract/OWNX > " Here we prove that M_{k,1}and Inv_{k,1} are congruence-simple for all k"
<http://purl.org/dc/terms/abstract/MISC > " Their Green relations J and Dare characterized: M_{k,1} and Inv_{k,1} are J-0-simple, and they have k-1non-zero D-classes"
<http://purl.org/dc/terms/abstract/OWNX > " They are submonoids of the multiplicative part of the Cuntzalgebra O_k"
<http://purl.org/dc/terms/abstract/MISC > " They are finitely generated, and their word problem over anyfinite generating set is in P"
<http://purl.org/dc/terms/abstract/MISC > " Their word problem is coNP-complete over certaininfinite generating sets"
<http://purl.org/dc/terms/abstract/OWNX > " Changes in this version: Section 4 has been thoroughly revised, and errorshave been corrected; however, the main results of Section 4 do not change"
<http://purl.org/dc/terms/abstract/OWNX > "Sections 1, 2, and 3 are unchanged, except for the proof of Theorem 2"
<http://purl.org/dc/terms/abstract/OWNX > "3, whichwas incomplete; a complete proof was published in the Appendix of reference[6], and is also given here."
<https://w3id.org/skg/articles/5327473888016701964> <http://xmlns.com/foaf/0.1/name>"Pseudo-random Puncturing: A Technique to Lower the Error Floor of Turbo Codes".
<https://w3id.org/skg/articles/5327473888016701964> <http://purl.org/dc/terms/abstract> <http://purl.org/dc/terms/abstract/-7335421883133534815>
<https://w3id.org/skg/articles/-7335421883133534815><http://purl.org/dc/terms/abstract/text>"It has been observed that particular rate-1/2 partially systematic parallelconcatenated convolutional codes (PCCCs) can achieve a lower error floor thanthat of their rate-1/3 parent codes Nevertheless, good puncturing patterns canonly be identified by means of an exhaustive search, whilst convergence towardslow bit error probabilities can be problematic when the systematic output of arate-1/2 partially systematic PCCC is heavily punctured In this paper, wepresent and study a family of rate-1/2 partially systematic PCCCs, which wecall pseudo-randomly punctured codes We evaluate their bit error rateperformance and we show that they always yield a lower error floor than that oftheir rate-1/3 parent codes Furthermore, we compare analytic results tosimulations and we demonstrate that their performance converges towards theerror floor region, owning to the moderate puncturing of their systematicoutput Consequently, we propose pseudo-random puncturing as a means ofimproving the bandwidth efficiency of a PCCC and simultaneously lowering itserror floor."
<http://purl.org/dc/terms/abstract/MISC > " It has been observed that particular rate-1/2 partially systematic parallelconcatenated convolutional codes (PCCCs) can achieve a lower error floor thanthat of their rate-1/3 parent codes"
<http://purl.org/dc/terms/abstract/MISC > " Nevertheless, good puncturing patterns canonly be identified by means of an exhaustive search, whilst convergence towardslow bit error probabilities can be problematic when the systematic output of arate-1/2 partially systematic PCCC is heavily punctured"
<http://purl.org/dc/terms/abstract/AIMX > " In this paper, wepresent and study a family of rate-1/2 partially systematic PCCCs, which wecall pseudo-randomly punctured codes"
<http://purl.org/dc/terms/abstract/OWNX > " We evaluate their bit error rateperformance and we show that they always yield a lower error floor than that oftheir rate-1/3 parent codes"
<http://purl.org/dc/terms/abstract/OWNX > " Furthermore, we compare analytic results tosimulations and we demonstrate that their performance converges towards theerror floor region, owning to the moderate puncturing of their systematicoutput"
<http://purl.org/dc/terms/abstract/OWNX > " Consequently, we propose pseudo-random puncturing as a means ofimproving the bandwidth efficiency of a PCCC and simultaneously lowering itserror floor."