In [17]:

```
## Testing the Naive Bayes Classification Model

## We have given the 3 different sample inputs for each of the labels
## Results are classified output with the probable chances of classificatio
n of being the position from Training Model.

##############################################################

import csv
from sklearn.naive_bayes import MultinomialNB

# import method for split train/test data set
from sklearn.model_selection import train_test_split

# import method to calculate metrics
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import classification_report
from sklearn.feature_extraction.text import TfidfVectorizer

# initialize the TfidfVectorizer



with open("indeed_scraped_data_science.csv", "r",encoding="ISO-8859-1") as
f:
    reader = csv.reader(f, delimiter=',')
    text = [(row[8]) for row in reader]

with open("indeed_scraped_data_science.csv", "r",encoding="ISO-8859-1") as
f:
    reader = csv.reader(f, delimiter=',')
    target = [(row[0]) for row in reader]


# convert tuple to list
text=list(text)
#print(len(text))

## all labels
target=list(target)
#print(len(target))

#print(target[0:10])

tfidf_vect = TfidfVectorizer()

# with stop words removed
#tfidf_vect = TfidfVectorizer(stop_words="english")

# generate tfidf matrix
dtm= tfidf_vect.fit_transform(text)

print("type of dtm:", type(dtm))
print("size of tfidf matrix:", dtm.shape)
```

```
# split dataset into train (70%) and test sets (30%)
X_train, X_test, y_train, y_test = train_test_split(\
                dtm, target, test_size=0.3, random_state=0)

# train a multinomial naive Bayes model using the testing data
clf = MultinomialNB().fit(X_train, y_train)

# predict the news group for the test dataset
predicted=clf.predict(X_test)

# get the list of unique labels
labels=sorted(list(set(target)))

# calculate performance metrics.
# Support is the number of occurrences of each label

precision, recall, fscore, support=\
     precision_recall_fscore_support(\
     y_test, predicted, labels=labels)

print(labels)
print(precision)
print(recall)
print(fscore)
print(support)

# another way to get all performance metrics
print(classification_report(y_test, predicted, target_names=labels))
```

```
type of dtm: <class 'scipy.sparse.csr.csr_matrix'>
size of tfidf matrix: (1818, 29670)
['1', '2', '3']
[ 0.58991228  1.          0.68965517]
[ 0.9676259   0.45522388  0.14925373]
[ 0.73297003  0.62564103  0.24539877]
[278 134 134]
             precision    recall  f1-score   support

          1       0.59      0.97      0.73       278
          2       1.00      0.46      0.63       134
          3       0.69      0.15      0.25       134

avg / total       0.72      0.64      0.59       546
```

Testing the Convolutional Neural Network Model

We have given the 3 different sample inputs for each of the lables

As a result we are getting the classified lables.

- We are converting them all into the according weights in order to sum up with the classification model.
- ouput has 3 labels where:
    - 1 indicates "Data Scientist"
    - 2 indicates "Senior Software Engineer"

- 2 indicates "Senior Software Engineer"
- 3 indicates "Vice President"

In [10]:

```
docs_new = ['I\'m a Vice President / Fashion designer of my own company. I design clothing and fashion ranges.',\
            'Developed client module to be used in Windows environment in C++, MFC, COM, TCP/IP Sockets', \
            'Designed and developed a demonstration system that includes all of the options and security \
            features that are available']

#Responsibilities: Costing, estimating and planning projects.
# generate tifid for new documents
X_new_tfidf = tfidf_vect.transform(docs_new)

print(X_new_tfidf.shape)

# predict classes for new documents
predicted = clf.predict(X_new_tfidf)

for idx, doc in enumerate(docs_new):
    print('%r => %s' % (doc, predicted[idx]))
```

```
(3, 28026)
"I'm a Vice President Vice President/ Fashion designer of my own company. I design clothing and fashion ranges." => 3
'Developed client module to be used in Windows environment in C++, MFC, COM , TCP/IP Sockets' => 2
'Designed and developed a demonstration system that includes all of the options and security features that are available' => 1
```

Applying MultiLabel Binarizer

In [6]:

```
from sklearn.preprocessing import MultiLabelBinarizer
import numpy as np

mlb = MultiLabelBinarizer()
Y=mlb.fit_transform(target)
# check size of indicator matrix
Y.shape
# check classes
mlb.classes_

# check # of samples in each class
np.sum(Y, axis=0)
```

Out[6]:

```
(1818, 3)
```

Out[6]:

```
array(['1', '2', '3'], dtype=object)
```

Out[6]:

```
array([923, 454, 441])
```

Tokenization for CNN model

In [11]:

```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

In [12]:

```
#####################   CNN
import pandas as pd
import nltk,string
import csv
# Load data

with open("indeed_scraped_data_science.csv", "r",encoding="ISO-8859-1") as
f:
    reader = csv.reader(f, delimiter=',')
    text = [(row[8]) for row in reader]
# tokenize each document into a list of unigrams
# strip punctuations and leading/trailing spaces from unigrams
# only unigrams with 2 or more characters are taken
sentences=[ [token.strip(string.punctuation).strip() \
             for token in nltk.word_tokenize(doc.lower()) \
                 if token not in string.punctuation and \
                 len(token.strip(string.punctuation).strip())>=2]\
             for doc in text]
print(sentences[0:1])
```

```
[['description', 'fis', 'provides', 'financial', 'software', 'world-class',
'services', 'and', 'global', 'business', 'solutions', 'let', 'us', 'help',
'you', 'compete', 'and', 'win', 'in', 'today', 'chaotic', 'marketplace', 'f
idelity', 'national', 'information', 'services', 'inc', 'better', 'known',
'by', 'the', 'abbreviation', 'fis', 'is', 'an', 'international', 'provider'
, 'of', 'financial', 'services', 'technology', 'and', 'outsourcing', 'servi
ces', 'fis', 'is', 'the', 'world', 'largest', 'global', 'provider', 'dedica
ted', 'to', 'financial', 'technology', 'wealth', 'management', 'risk', 'and
', 'compliance', 'trade', 'enablement', 'transaction', 'processing', 'and',
'record-keeping', 'responsibilities', 'extracted', 'data', 'from', 'hdfs',
'and', 'prepared', 'data', 'for', 'exploratory', 'analysis', 'using', 'data
', 'munging', 'built', 'models', 'using', 'statistical', 'techniques', 'lik
e', 'bayesian', 'hmm', 'and', 'machine', 'learning', 'classification', 'mod
els', 'like', 'xgboost', 'svm', 'and', 'random', 'forest', 'participated',
'in', 'all', 'phases', 'of', 'data', 'mining', 'data', 'cleaning', 'data',
'collection', 'developing', 'models', 'validation', 'visualization', 'and',
'performed', 'gap', 'analysis', 'highly', 'immersive', 'data', 'science', '
program', 'involving', 'data', 'manipulation', 'visualization', 'web', 'scr
aping', 'machine', 'learning', 'python', 'programming', 'sql', 'git', 'mong
odb', 'hadoop', 'setup', 'storage', 'and', 'data', 'analysis', 'tools', 'in
', 'aws', 'cloud', 'computing', 'infrastructure', 'installed', 'and', 'used
', 'caffe', 'deep', 'learning', 'framework', 'worked', 'on', 'different', '
data', 'formats', 'such', 'as', 'json', 'xml', 'and', 'performed', 'machine
', 'learning', 'algorithms', 'in', 'python', 'worked', 'as', 'data', 'archi
tects', 'and', 'it', 'architects', 'to', 'understand', 'the', 'movement', '
of', 'data', 'and', 'its', 'storage', 'and', 'er', 'studio', '9.7', 'used',
'pandas', 'numpy', 'seaborn', 'matplotlib', 'scikit-learn', 'scipy', 'nltk'
, 'in', 'python', 'for', 'developing', 'various', 'machine', 'learning', 'a
lgorithms', 'data', 'manipulation', 'and', 'aggregation', 'from', 'differen
```

t', 'source', 'using', 'nexus', 'business', 'objects', 'toad', 'power', 'bi
', 'and', 'smart', 'view', 'implemented', 'agile', 'methodology', 'for', 'b
uilding', 'an', 'internal', 'application', 'focus', 'on', 'integration', 'o
verlap', 'and', 'informatica', 'newer', 'commitment', 'to', 'mdm', 'with',
'the', 'acquisition', 'of', 'identity', 'systems', 'coded', 'proprietary',
'packages', 'to', 'analyze', 'and', 'visualize', 'spcfile', 'data', 'to', '
identify', 'bad', 'spectra', 'and', 'samples', 'to', 'reduce',
'unnecessary', 'procedures', 'and', 'costs', 'programmed', 'utility', 'in',
'python', 'that', 'used', 'multiple', 'packages', 'numpy', 'scipy', 'pandas
', 'implemented', 'classification', 'using', 'supervised', 'algorithms', 'l
ike', 'logistic', 'regression', 'decision', 'trees', 'naive', 'bayes', 'knn
', 'as', 'architect', 'delivered', 'various', 'complex',
'olapdatabases/cubes', 'scorecards', 'dashboards', 'and', 'reports', 'updat
ed', 'python', 'scripts', 'to', 'match', 'training', 'data', 'with', 'our',
'database', 'stored', 'in', 'aws', 'cloud', 'search', 'so', 'that', 'we', '
would', 'be', 'able', 'to', 'assign', 'each', 'document', 'response', 'labe
l', 'for', 'further', 'classification', 'used', 'teradata', 'utilities', 's
uch', 'as', 'fast', 'export', 'mload', 'for', 'handling', 'various', 'tasks
', 'data', 'migration/etl', 'from', 'oltp', 'source', 'systems', 'to', 'ola
p', 'target', 'systems', 'data', 'transformation', 'from', 'various', 'reso
urces', 'data', 'organization', 'features', 'extraction', 'from', 'raw', 'a
nd', 'stored', 'validated', 'the', 'machine', 'learning', 'classifiers', 'u
sing', 'roc', 'curves', 'and', 'lift', 'charts', 'environment', 'unix', 'py
thon', '3.5.2', 'mllib', 'sas', 'regression', 'logistic', 'regression', 'ha
doop', '2.7.4', 'nosql', 'teradata', 'oltp', 'random', 'forest', 'olap', 'h
dfs', 'ods', 'nltk', 'svm', 'json', 'xml', 'and', 'mapreduce', 'description
', 'cbre', 'group', 'inc', 'is', 'the', 'largest', 'commercial', 'real', 'e
state', 'services', 'and', 'investment', 'firm', 'in', 'the', 'world', 'it'
, 'is', 'based', 'in', 'los', 'angeles', 'california', 'and', 'operates', '
more', 'than', '450', 'offices', 'worldwide', 'and', 'has', 'clients', 'in'
, 'more', 'than', '100', 'countries', 'responsibilities', 'utilized', 'spar
k', 'scala', 'hadoop', 'hql', 'vql', 'oozie', 'pyspark', 'data', 'lake', 't
ensorflow', 'hbase', 'cassandra', 'redshift', 'mongodb', 'kafka', 'kinesis'
, 'spark', 'streaming', 'edward', 'cuda', 'mllib', 'aws', 'python', 'broad'
, 'variety', 'of', 'machine', 'learning', 'methods', 'including',
'classifications', 'regressions', 'dimensionally', 'reduction', 'etc', 'uti
lized', 'the', 'engine', 'to', 'increase', 'to', 'retrieve', 'datafrom', 'o
racle', 'database', 'and', 'used', 'etl', 'for', 'data', 'transformation',
'performed', 'data', 'cleaning', 'features', 'scaling', 'features', 'engine
ering', 'using', 'pandas', 'and', 'numpy', 'packages', 'in', 'python', 'exp
loring', 'dag', 'their', 'dependencies', 'and', 'logs', 'using', 'airflow',
'pipelines', 'for', 'automation', 'performed', 'data', 'cleaning', 'and', '
feature', 'selection', 'using', 'mllib', 'package', 'in', 'pyspark', 'diffe
rent', 'departments', 'created', 'and', 'designed', 'reports', 'that', 'wil
l', 'use', 'gathered', 'metrics', 'to', 'infer', 'and', 'draw', 'logical',
'conclusions', 'of', 'past', 'and', 'future', 'behavior', 'developed', 'map
reduce', 'pipeline', 'for', 'feature', 'extraction', 'using', 'hive', 'and'
, 'pig', 'created', 'data', 'quality', 'scripts', 'using', 'sql', 'and', 'h
ive', 'to', 'validate', 'successful', 'data', 'load', 'and', 'quality', 'of
', 'the', 'data', 'created', 'various', 'types', 'of', 'data',
'visualizations', 'using', 'python', 'and', 'tableau', 'communicated', 'the
', 'results', 'with', 'operations', 'team', 'for', 'taking', 'best', 'decis
ions', 'collected', 'data', 'needs', 'and', 'requirements', 'by', 'interact
ing', 'with', 'the', 'other', 'departments', 'environment', 'python', '2.x'
, 'cdh5', 'hdfs', 'hadoop', '2.3', 'hive', 'impala', 'aws', 'linux', 'spark
', 'tableau', 'desktop', 'sql', 'server', '2014', 'microsoft', 'excel', 'ma
tlab', 'spark', 'sql', 'pyspark', 'description', 'the', 'walgreens', 'compa
nyis', 'an', 'american', 'company', 'that', 'operatesas', 'the', 'second-la
rgest', 'pharmacy', 'store', 'chain', 'in', 'the', 'united', 'states', 'it'
, 'specializes', 'in', 'filling', 'prescriptions', 'health', 'and', 'wellne

```
ronment', 'python', 'mysql']]
```

Testing with Genism model

In [13]:

```python
from gensim.models import word2vec
import logging
import pandas as pd

# print out tracking information
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', \
                    level=logging.INFO)

wv_model = word2vec.Word2Vec(sentences, min_count=5, size=200, window=5,
workers=4 )
```

```
2018-04-26 01:31:10,357 : INFO : collecting all words and their counts
2018-04-26 01:31:10,359 : INFO : PROGRESS: at sentence #0, processed 0 word
s, keeping 0 word types
2018-04-26 01:31:10,593 : INFO : collected 44399 word types from a corpus o
f 1026764 raw words and 1818 sentences
2018-04-26 01:31:10,594 : INFO : Loading a fresh vocabulary
2018-04-26 01:31:10,681 : INFO : min_count=5 retains 10583 unique words (23
% of original 44399, drops 33816)
2018-04-26 01:31:10,682 : INFO : min_count=5 leaves 970554 word corpus (94%
of original 1026764, drops 56210)
2018-04-26 01:31:10,738 : INFO : deleting the raw counts dictionary of 4439
9 items
2018-04-26 01:31:10,747 : INFO : sample=0.001 downsamples 31 most-common wo
rds
2018-04-26 01:31:10,748 : INFO : downsampling leaves estimated 777965 word
corpus (80.2% of prior 970554)
2018-04-26 01:31:10,807 : INFO : estimated required memory for 10583 words
and 200 dimensions: 22224300 bytes
2018-04-26 01:31:10,809 : INFO : resetting layer weights
2018-04-26 01:31:11,046 : INFO : training model with 4 workers on 10583 voc
abulary and 200 features, using sg=0 hs=0 sample=0.001 negative=5 window=5
2018-04-26 01:31:12,085 : INFO : EPOCH 1 - PROGRESS: at 67.16% examples, 57
3384 words/s, in_qsize 7, out_qsize 0
2018-04-26 01:31:12,426 : INFO : worker thread finished; awaiting finish of
3 more threads
2018-04-26 01:31:12,446 : INFO : worker thread finished; awaiting finish of
2 more threads
2018-04-26 01:31:12,448 : INFO : worker thread finished; awaiting finish of
1 more threads
2018-04-26 01:31:12,450 : INFO : worker thread finished; awaiting finish of
0 more threads
2018-04-26 01:31:12,452 : INFO : EPOCH - 1 : training on 1026764 raw words
(778132 effective words) took 1.4s, 559593 effective words/s
2018-04-26 01:31:13,461 : INFO : EPOCH 2 - PROGRESS: at 75.69% examples, 69
7724 words/s, in_qsize 7, out_qsize 0
2018-04-26 01:31:13,532 : INFO : worker thread finished; awaiting finish of
3 more threads
2018-04-26 01:31:13,548 : INFO : worker thread finished; awaiting finish of
2 more threads
2018-04-26 01:31:13,551 : INFO : worker thread finished; awaiting finish of
1 more threads
```

```
2018-04-26 01:31:13,558 : INFO : worker thread finished; awaiting finish of
0 more threads
2018-04-26 01:31:13,559 : INFO : EPOCH - 2 : training on 1026764 raw words
(778252 effective words) took 1.1s, 705425 effective words/s
2018-04-26 01:31:14,515 : INFO : worker thread finished; awaiting finish of
3 more threads
2018-04-26 01:31:14,520 : INFO : worker thread finished; awaiting finish of
2 more threads
2018-04-26 01:31:14,528 : INFO : worker thread finished; awaiting finish of
1 more threads
2018-04-26 01:31:14,534 : INFO : worker thread finished; awaiting finish of
0 more threads
2018-04-26 01:31:14,535 : INFO : EPOCH - 3 : training on 1026764 raw words
(778235 effective words) took 1.0s, 799565 effective words/s
2018-04-26 01:31:15,443 : INFO : worker thread finished; awaiting finish of
3 more threads
2018-04-26 01:31:15,456 : INFO : worker thread finished; awaiting finish of
2 more threads
2018-04-26 01:31:15,471 : INFO : worker thread finished; awaiting finish of
1 more threads
2018-04-26 01:31:15,475 : INFO : worker thread finished; awaiting finish of
0 more threads
2018-04-26 01:31:15,477 : INFO : EPOCH - 4 : training on 1026764 raw words
(778105 effective words) took 0.9s, 829361 effective words/s
2018-04-26 01:31:16,506 : INFO : EPOCH 5 - PROGRESS: at 68.15% examples, 58
6670 words/s, in_qsize 7, out_qsize 0
2018-04-26 01:31:16,775 : INFO : worker thread finished; awaiting finish of
3 more threads
2018-04-26 01:31:16,786 : INFO : worker thread finished; awaiting finish of
2 more threads
2018-04-26 01:31:16,789 : INFO : worker thread finished; awaiting finish of
1 more threads
2018-04-26 01:31:16,803 : INFO : worker thread finished; awaiting finish of
0 more threads
2018-04-26 01:31:16,805 : INFO : EPOCH - 5 : training on 1026764 raw words
(777897 effective words) took 1.3s, 588703 effective words/s
2018-04-26 01:31:16,807 : INFO : training on a 5133820 raw words (3890621 e
ffective words) took 5.8s, 675609 effective words/s
```

In [14]:

```
from gensim.models.doc2vec import TaggedDocument

docs=[TaggedDocument(sentences[i], [str(i)]) for i in range(len(sentences))
]
```

CNN training and testing

In [21]:

```
from keras.layers import Embedding, Dense, Conv1D, MaxPooling1D, \
Dropout, Activation, Input, Flatten, Concatenate
from keras.models import Model
from keras.regularizers import l2
from keras.callbacks import EarlyStopping, ModelCheckpoint

def cnn_model(FILTER_SIZES, \
              # filter sizes as a list
              MAX_NB_WORDS, \
```

```
                # total number of words
                MAX_DOC_LEN, \
                # max words in a doc
                EMBEDDING_DIM=200, \
                # word vector dimension
                NUM_FILTERS=64, \
                # number of filters for all size
                DROP_OUT=0.5, \
                # dropout rate
                NUM_OUTPUT_UNITS=1, \
                # number of output units
                NUM_DENSE_UNITS=100,\
                # number of units in dense layer
                PRETRAINED_WORD_VECTOR=None,\
                # Whether to use pretrained word vectors
                LAM=0.0):
                # regularization coefficient

    main_input = Input(shape=(MAX_DOC_LEN,), \
                       dtype='int32', name='main_input')

    if PRETRAINED_WORD_VECTOR is not None:
        embed_1 = Embedding(input_dim=MAX_NB_WORDS+1, \
                        output_dim=EMBEDDING_DIM, \
                        input_length=MAX_DOC_LEN, \
                        weights=[PRETRAINED_WORD_VECTOR],\
                        trainable=False,\
                        name='embedding')(main_input)
    else:
        embed_1 = Embedding(input_dim=MAX_NB_WORDS+1, \
                        output_dim=EMBEDDING_DIM, \
                        input_length=MAX_DOC_LEN, \
                        name='embedding')(main_input)
    # add convolution-pooling-flat block
    conv_blocks = []
    for f in FILTER_SIZES:
        conv = Conv1D(filters=NUM_FILTERS, kernel_size=f, \
                      activation='relu', name='conv_'+str(f))(embed_1)
        conv = MaxPooling1D(MAX_DOC_LEN-f+1, name='max_'+str(f))(conv)
        conv = Flatten(name='flat_'+str(f))(conv)
        conv_blocks.append(conv)

    if len(conv_blocks)>1:
        z=Concatenate(name='concate')(conv_blocks)
    else:
        z=conv_blocks[0]

    drop=Dropout(rate=DROP_OUT, name='dropout')(z)

    dense = Dense(NUM_DENSE_UNITS, activation='relu',\
                    kernel_regularizer=l2(LAM),name='dense')(drop)
    preds = Dense(NUM_OUTPUT_UNITS, activation='sigmoid', name='output')(de
nse)
    model = Model(inputs=main_input, outputs=preds)

    model.compile(loss="binary_crossentropy", \
              optimizer="adam", metrics=["accuracy"])

    return model
```

CNN model checkpoint

In [23]:

```
from keras.callbacks import EarlyStopping, ModelCheckpoint

# the file path to save best model
BEST_MODEL_FILEPATH="best_model"

# define early stopping based on validation loss
# if validation loss is not improved in
# an iteration compared with the previous one,
# stop training (i.e. patience=0).
# mode='min' indicate the loss needs to decrease
earlyStopping=EarlyStopping(monitor='val_loss', \
                            patience=0, verbose=2, \
                            mode='min')

# define checkpoint to save best model
# which has max. validation acc
checkpoint = ModelCheckpoint(BEST_MODEL_FILEPATH, \
                             monitor='val_acc', \
                             verbose=2, \
                             save_best_only=True, \
                             mode='max')

# compile model
model.compile(loss="binary_crossentropy", \
              optimizer="adam", metrics=["accuracy"])
```

Multilabel Binarizer for predicting label

In [18]:

```
from sklearn.preprocessing import MultiLabelBinarizer
from numpy.random import shuffle

mlb = MultiLabelBinarizer()
Y=mlb.fit_transform(target)  # instead of label it is target
# check size of indicator matrix
Y.shape
# check classes
mlb.classes_
# check # of samples in each class
np.sum(Y, axis=0)
```

Out[18]:

```
(1818, 3)
```

Out[18]:

```
array(['1', '2', '3'], dtype=object)
```

Out[18]:

```
array([923, 454, 441])
```

In [19]:

```python
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

import numpy as np
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

import numpy as np

# get a Keras tokenizer

MAX_NB_WORDS=8000
# documents are quite long in the dataset
MAX_DOC_LEN=1000

tokenizer = Tokenizer(num_words=MAX_NB_WORDS)
tokenizer.fit_on_texts(text)
voc=tokenizer.word_index
# convert each document to a list of word index as a sequence
sequences = tokenizer.texts_to_sequences(text)
# get the mapping between words to word index

# pad all sequences into the same length (the longest)
padded_sequences = pad_sequences(sequences, \
                                 maxlen=MAX_DOC_LEN, \
                                 padding='post', truncating='post')

#print(padded_sequences[0])


EMBEDDING_DIM=200
# get a Keras tokenizer

MAX_NB_WORDS=8000
# documents are quite long in the dataset
MAX_DOC_LEN=1000

tokenizer = Tokenizer(num_words=MAX_NB_WORDS)
tokenizer.fit_on_texts(text)
# tokenizer.word_index provides the mapping
# between a word and word index for all words
NUM_WORDS = min(MAX_NB_WORDS, len(tokenizer.word_index))

# "+1" is for padding symbol
embedding_matrix = np.zeros((NUM_WORDS+1, EMBEDDING_DIM))

for word, i in tokenizer.word_index.items():
    # if word_index is above the max number of words, ignore it
    if i >= NUM_WORDS:
        continue
    if word in wv_model.wv:
        embedding_matrix[i]=wv_model.wv[word]
```

Training Model

In [28]:

```python
from sklearn.model_selection import train_test_split
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MultiLabelBinarizer
from numpy.random import shuffle
EMBEDDING_DIM=200
FILTER_SIZES=[2,3,4]

# set the number of output units
# as the number of classes
#mlb = MultiLabelBinarizer()
output_units_num=len(mlb.classes_)

#Number of filters for each size
num_filters=64

# set the dense units
dense_units_num= num_filters*len(FILTER_SIZES)

BTACH_SIZE = 32
NUM_EPOCHES = 100

# With well trained word vectors, sample size can be reduced
# Assume we only have 500 labeled data
# split dataset into train (70%) and test sets (20%)

padded_sequences.shape
Y.shape

X_train, X_test, Y_train, Y_test = train_test_split(\
                padded_sequences[0:500], Y[0:500], \
                test_size=0.2, random_state=0, shuffle=True)


# create the model with embedding matrix
model=cnn_model(FILTER_SIZES, MAX_NB_WORDS, \
                MAX_DOC_LEN, \
                NUM_FILTERS=num_filters,\
                NUM_OUTPUT_UNITS=output_units_num, \
                NUM_DENSE_UNITS=dense_units_num,\
                PRETRAINED_WORD_VECTOR=embedding_matrix)

earlyStopping=EarlyStopping(monitor='val_loss', patience=1, verbose=2, mode
='min')
checkpoint = ModelCheckpoint(BEST_MODEL_FILEPATH, monitor='val_loss', \
                             verbose=2, save_best_only=True, mode='min')

training=model.fit(X_train, Y_train, \
          batch_size=BTACH_SIZE, epochs=NUM_EPOCHES, \
          callbacks=[earlyStopping, checkpoint],\
          validation_data=[X_test, Y_test], verbose=2)
```

Out[28]:

```
(1818, 1000)
```

Out[28]:

```
(1818, 3)
```

```
Train on 400 samples, validate on 100 samples
```

Model Summary

In [29]:

```
model.summary()
```

Classification Report

In [31]:

```
from sklearn.metrics import classification_report
pred=model.predict(padded_sequences[500:1000])

Y_pred=np.copy(pred)
Y_pred=np.where(Y_pred>0.5,1,0)

Y_pred[0:10]
Y[500:510]

print(classification_report(Y[500:1000], Y_pred, target_names=mlb.classes_)
)
```

Out[31]:

```
array([[1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0]])
```

Out[31]:

```
array([[1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0]])
```

```
             precision    recall  f1-score   support

          1       0.85      1.00      0.92       401
          2       0.90      0.97      0.88        77
          3       0.83      0.23      0.34        22

avg / total       0.86      0.73      0.71       500
```

```
/anaconda3/lib/python3.6/site-
packages/sklearn/metrics/classification.py:1135: UndefinedMetricWarning: Pr
ecision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples.
  'precision', 'predicted', average, warn_for)
```

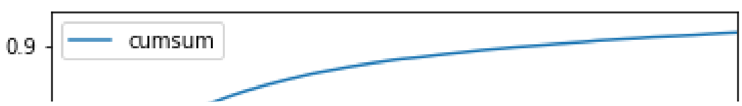
```
/anaconda3/lib/python3.6/site-packages/sklearn/metrics/classification.py:1137: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in labels with no true samples.
  'recall', 'true', average, warn_for)
```

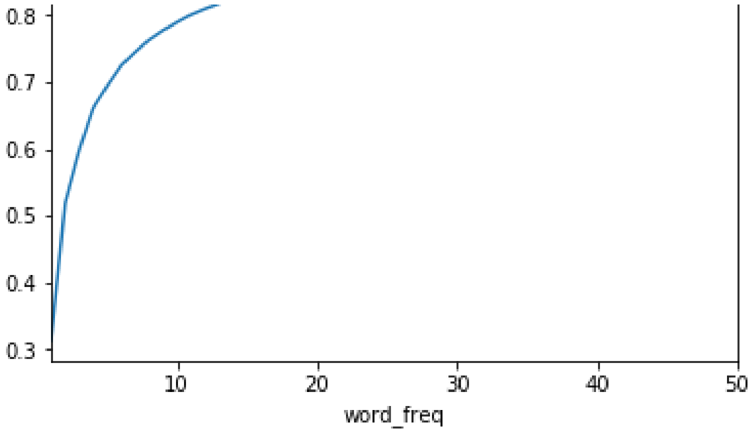
Cumsum

In [32]:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# get count of each word
df=pd.DataFrame.from_dict(tokenizer.word_counts, orient="index")
df.columns=['freq']
print(df.head())

# get histogram of word count
df=df['freq'].value_counts().reset_index()
df.columns=['word_freq','count']

# sort by word_freq
df=df.sort_values(by='word_freq')

# convert absolute counts to precentage
df['percent']=df['count']/len(tokenizer.word_counts)
# get cumulative percentage
df['cumsum']=df['percent'].cumsum()

print(df.head())

df.iloc[0:50].plot(x='word_freq', y='cumsum');

plt.show();

# if set min count for word to 10,
# what % of words can be included?
# how many words will be included?
# This is the parameter MAX_NB_WORDS
# tokenizer = Tokenizer(num_words=MAX_NB_WORDS)
```

```
              freq
description    439
fis             24
provides       478
financial      974
software      3322
   word_freq  count   percent    cumsum
0          1   9602  0.313228  0.313228
1          2   6318  0.206100  0.519328
2          3   2402  0.078356  0.597684
3          4   1953  0.063709  0.661393
4          5   1008  0.032882  0.694275
```





In [33]:

```
sen_len=pd.Series([len(item) for item in sequences])

# create histogram of sentence length
# the "index" is the sentence length
# "counts" is the count of sentences at a length
df=sen_len.value_counts().reset_index().sort_values(by='index')
df.columns=['sent_length','counts']

# sort by sentence length
# get percentage and cumulative percentage

df['percent']=df['counts']/len(sen_len)
df['cumsum']=df['percent'].cumsum()
print(df.head(3))

# From the plot, 90% sentences have length<500
# so it makes sense to set MAX_DOC_LEN=4~500
df.plot(x="sent_length", y='cumsum');
plt.show();
```

```
     sent_length  counts   percent    cumsum
347            0       2  0.001100  0.001100
0              1      49  0.026953  0.028053
1              2      13  0.007151  0.035204
```