### **CHAPTER 11: Discovering Topics in Text Collection using Amazon SageMaker**

### **- 26 pages**

One of the most useful ways to comprehend text is through topics.

The process of learning, recognizing and extracting these topics is called topic modelling.

Understanding broad topics in texts has several applications. It can be used in the legal industry to surface themes from the contracts. Rather than manually reviewing mountains of contracts for certain provisions, through unsupervised learnings, themes or topics can be surfaced. Furthermore, it can be used in the retail industry to identify broad trends in the social media conversations. These broad trends can then be used for product innovation – introduce new merchandise into online and physical stores, informing product assortment.

What we will cover in this chapter:

* Overview of various approaches to discovering topics in text collection
* Topic modeling in SageMaker
* Neural Topic model training and inference

In this chapter, we are going to learn to synthesize topics from long form text (longer than 140 characters per text). There are number of methods to uncover topics from text collection. These methods can be broadly categorized into linear learning and non-linear learning. All topic models are based on the premise that:

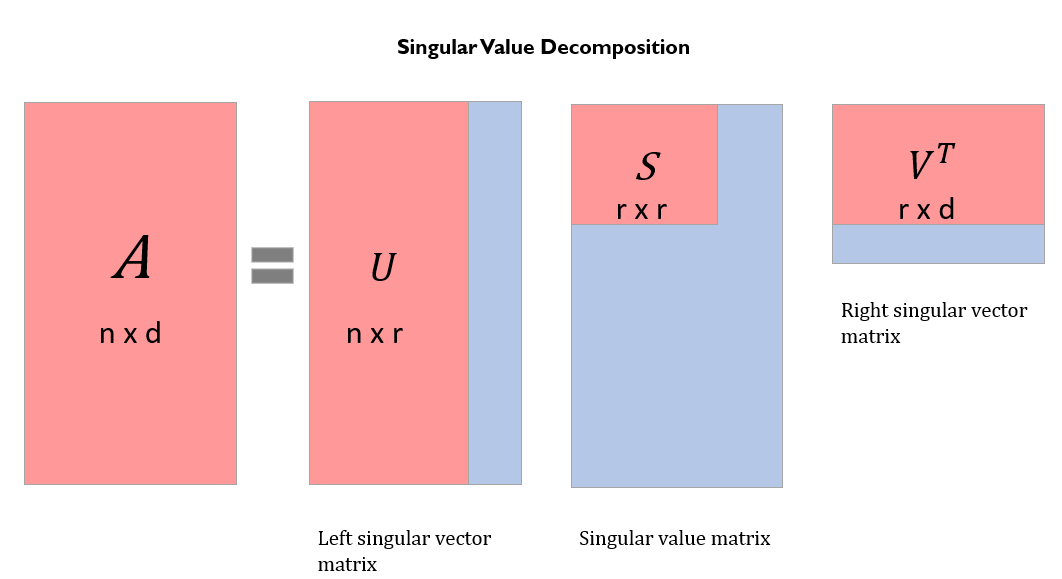
* All documents contain a collection of topics
* All topics contain a collection of words

Topics here are the hidden or latent variables that we do not necessarily observe. These variables reflect the meaning of the document.

*Linear Learning* – some of the techniques include --

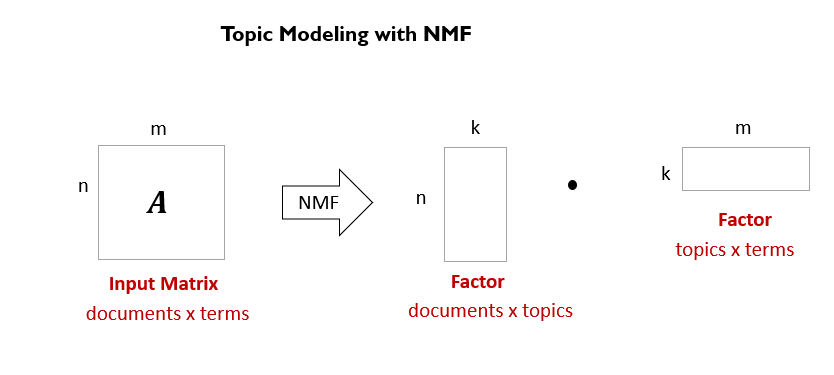
**Latent Semantic Analysis (LSA):** The matrix of documents and words is broken down into document-topic matrix and topic-word matrix. In the simplest form, the document word matrix consists of raw counts, which is frequency with which a given word occurs in a given document. Since this approach doesn’t account for significance of each word in the document, we replace raw counts with tf-idf (term frequency-inverse document frequency) score. Through Tf-idf, words that occur frequently within the document in question, but less frequently across all the other documents will get higher weight. Given that the matrix of documents and words is sparse and noisy, dimensionality must be reduced to obtain meaningful relationships between documents and words via topics. This is done through truncated SVD (Singular Value Decomposition), where document-word matrix is broken down into 3 different matrices of document topic (*U*), word-topic (*V*), singular values matrix (*S*), where singular values represent strength of the topics. The decomposition is also unique. To reduce dimensionality, only t largest singular values are chosen, and only the first t columns of U and V are retained. t is a hyperparameter and can be adjusted to reflect the number of topics we want to find. In linear algebra, any m x n matrix A can be decomposed as

, where U is left singular vector, and V is right singular vector. As noted, this decomposition is unique



The second way to conduct matrix factorization is through NMF.

Non-negative matrix factorization (NMF) – NMF belongs to linear algebra algorithms for identifying latent structure in the data. Two non-negative matrices are used to approximate the document-term matrix.



The difference between NMF and SVD is that with SVD, we can end up with negative component (left and/or right) matrices, which is not natural for interpreting textual representation. NMF, on the other hand, generates non-negative representations for performing LSA

The drawback of LSA is that it has less interpretable topics and it has less efficient representation. Additionally, it is a linear model, which cannot be used to model non-linear dependencies. The number of latent topics is limited by the rank of the matrix.

**Probabilistic LSA (pLSA)** – The whole idea of pLSA is to find a probabilistic model of latent topics that can generate documents and words we observe. Therefore, the joint probability – probability of finding a combination of document and word – can be written as:

We can see how pLSA is similar to LSA, in that corresponds to singular value matrix, corresponds to left singular vector, and corresponds to right singular vector from SVD.

The number one disadvantage with this approach is that for new documents, we do not know probability of finding topics. LDA addresses this issue.

**Latent Dirichlet Allocation(LDA)** – It is the most common type of topic modeling and is much more generalized than LSA and pLSA (probabilistic LSA). It is a Bayesian version of pLSA. Given the Dirichlet (aka distribution of distributions) priors – distribution of topic mixture and word mixture -- topic distribution in documents and word distributions in topics are derived. Topic mixture distribution can generate any type of mixture of topics. For example, given 3 topics, topic 1 can be 70%, topic 2 can be 20%, and topic 3 can be 10%. Likewise, any type of word mixture can be sampled from the distribution.

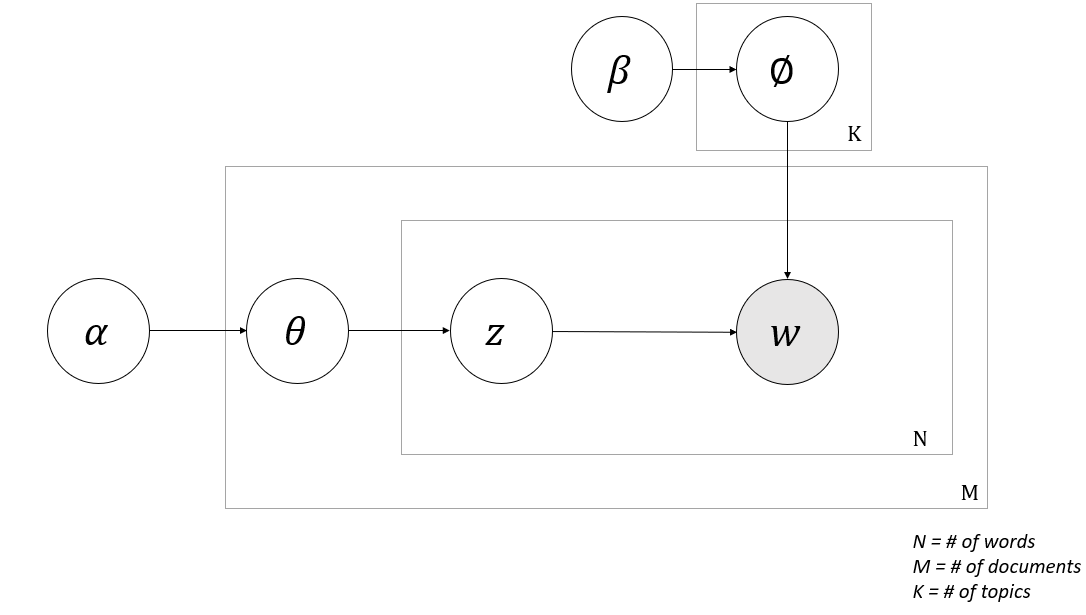
Let’s understand the parameters of the mixture distributions:

Alpha is Dirichlet prior on per document. High alpha, , indicates that each document contains a mixture of most of the topics, while low alpha means each document is a combination of fewer topics. Beta, , is Dirichlet prior on per topic. High beta reflects each topic is a mixture of most of all words. While low beta indicates each topic is a mixture of fewer words. Theta, , is the probability distribution of topics in documents, given a type of topic mixture (topic 1- 70%, topic 2 – 20%, & topic 3 – 10%). Z is used to annotate each topic, which is assigned to each word, W. Phi, is probability distribution of words in topics, given a type of word mixture. Therefore, we can say probability of a word given a document is:

which is for each topic.

We decompose the probability distribution matrix of words in document into two matrices consisting of distribution of topics ( in documents and of words in topics (. We will start with randomly initializing weights to both the matrices. We assume that the documents are generated by:

1. Randomly choose a topic from the distribution of topics in a document based on their assigned weights
2. Next, based on the distribution of words for the chosen topic, we select a word at random and put it in the document
3. The above process is repeated for all the topics in the document



If the document generated through the process detailed above does not align with the actual document, then we know that the weights in the matrices need to be adjusted. To optimize the weights, we will use a technique called Gibbs sampling.

With Gibbs sampling, we will maximize the likelihood of data given the two matrices – topic distribution in documents and word distribution in topics:

* We start with suboptimal topics representation of all the documents and word distributions of all topics
* For optimal allocation, for each document d
  + And for each topic, we compute

p(topic t | document d), which is the proportion of the words in document d belonging to topic t. This is nothing but weight of the topic in this document

p(word w | topic t) is the proportion of the assignments, that come from word w, to topic t across all the documents. In other words, to what extent is this word defining this topic across all the documents.

* + Re-assign w a new topic t, where we choose topic t with probability v, where v is p(topic t | document d) \* p(word w | topic t). We will assign a new topic t to the word w if this topic is better defined by the word across all the documents
  + What we are assuming in this step is except for the current assignment all the other topic assignments are accurate

After the repeating the above process enough number of times, we will eventually reach an optimal state with good assignments. We can use these assignments to estimate the topic distribution for each of the documents and words associated with each topic (word distribution for each topic)

The advantage of this approach is that topics are very interpretable relative to all the other methods discussed above.

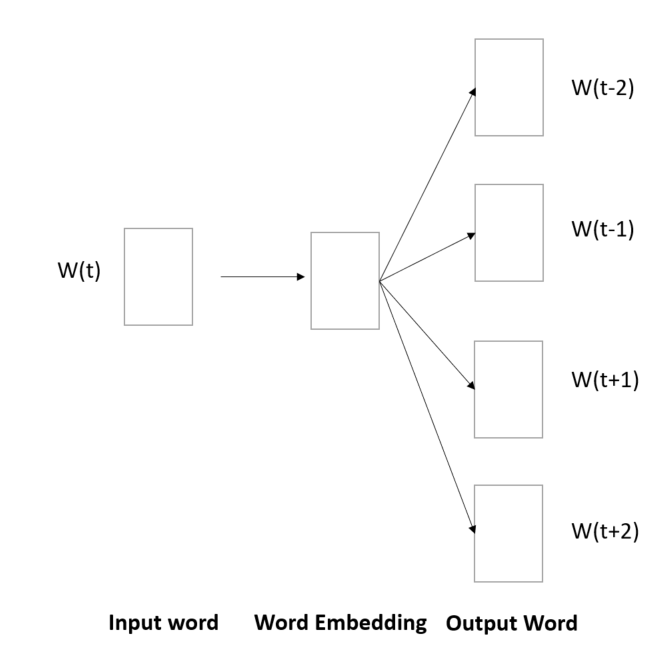
*Non-Linear Learning*

Neural network models for topic modeling are much more flexible and new capabilities can be easily added (for ex: creating contextual words for an input/target word)

**Lda2vec** is a superset of word2vec and LDA models. The unique attribute of this model is that it learns word, document and topic vectors. It uses both word and document vectors to predict words next to the target word. Document vector consists of two components – topic matrix or topic embedding and topic weights for each document.

Word2vec creates word embeddings by training on co-occurring words (for a given window size). The trained word2vec takes an input word and generates probability for words in the vocabulary – probability that these words occur in the context of input word.

LDA, as detailed earlier, generates topic embeddings.



Lda2vec is similar to Neural Variational Document Models (NVDM) in terms of learning topic embedding and assigning words to define each topic. However, NVDM adopt a much cleaner and flexible approach for topic modeling by creating document vectors using neural network; word to word relationship is completely disregarded.

**NVDM (Neural Variational Document Model)** is flexible generative document modeling process, which uses variational inference framework. Inference network is constructed by passing discrete text as input, producing variational distribution of a document. It is variational or stochastic because we want to learn multiple representations of documents through topics.

NVDM combines continuous stochastic document representation with bag-of-words generative model, producing accurate representation of documents relative to other models. Variational Autoencoder (uses a neural network to encode the dataset and a second neural network to decode the compressed representation of dataset) infrastructure is used to look at the best way to approximate information in a data set. The variational autoencoder is optimized by minimizing the loss in a) reconstructing the original document from topic embeddings and b) building a stochastic representation of the input document (aka Kullback Leibler Divergence), which are nothing but topic embeddings. Note that we are approximating each bag-of-word representation of document using a gaussian distribution. KL divergence measures how much information is lost when approximating documents with gaussian distribution.

In summary, the inference network (first part) + generative network (second part) are both parametrized by neural networks. The generative network outputs parameters for probability distribution of words.

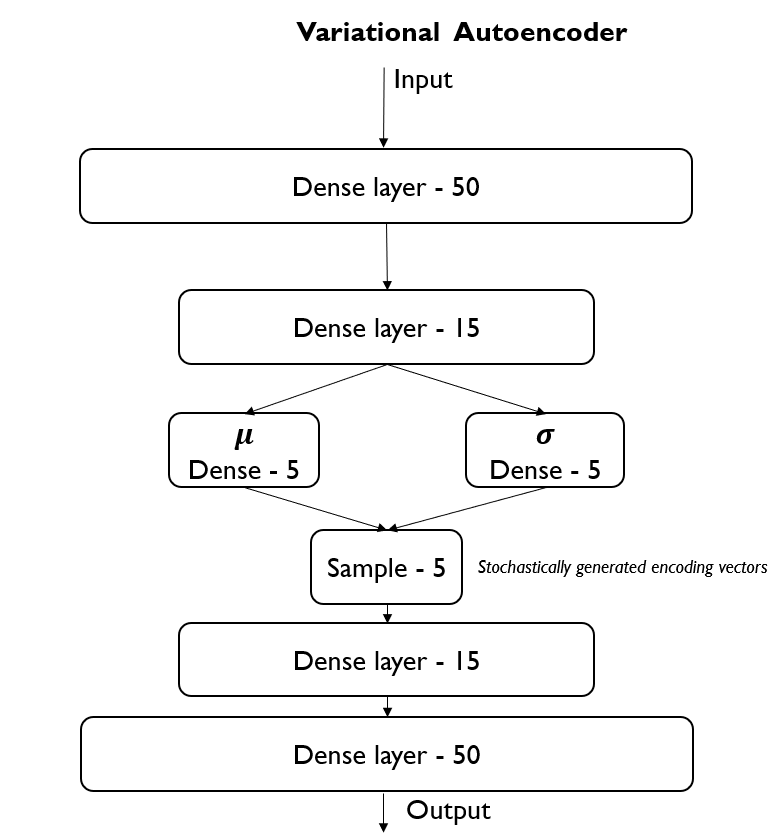
*Let’s do a deep dive on Neural Topic Model (NTM), an implementation of NVDM from AWS*

**Amazon SageMaker and Topic Modeling**

As detailed earlier, topic modeling has several applications, ranging from document classification to information retrieval to document summarization. Given the applications of topic modeling, AWS offers both readily consumable API, AWS Comprehend, and built-in algorithms, LDA (Latent Dirichlet Allocation) and NTM (Neural Topic Modeling). As detailed in earlier chapters, AWS Comprehend works well for most of the scenarios, such as synthesizing social media conversations and helpdesk tickets. Built-in algorithms, on the other hand, provide fine-grained control and flexibility to uncover topics from long form text belonging to a specific domain. Additionally, the Hosting service from SageMaker enables seamless integration of the trained model into target web or mobile applications. The algorithm training can be optimized for a specific use case at hand. LDA applies linear learning while NTM employs multilayer perceptron (MLP) to learn latent representations of documents.

**How Neural Topic Model (NTM) Works**

The NTM implements the Neural Variational Document Model (NVDM). NVDM is like Variational Autoencoder (VAE). VAE are powerful generative models. Autoencoders are used to solve complex problems in a variety of areas, including Fraud Detection and Natural Language Processing (sentiment analysis, document clustering, paraphrase detection). These neural networks, essentially, learn compressed representation of data, also known as dimensionality reduction. Autoencoders constitute a pair of connected networks, an encoder and decoder. An encoder takes an input and converts it into a smaller, dense representation. While a decoder network can convert it back to input. Variational part of the document model – it is the part that lends them well for generative models. Rather than outputting an encoding vector of size less than input size, it generates two vectors, a vector of means ( and a vector of standard deviation (. Intuitively, the mean vector controls where encoding of an input should be centered around, while the standard deviation controls the area around the center. Because the sample generated each time is going to vary, the decoder will learn to reconstruct different latent encodings of input.



*Variational Autoencoder = Encoder + Decoder + Loss function*

The basic idea is to use a Multiple Layer Perceptron (MLP) to encode the input, which is bag of words, to produce latent representation of documents – , which is a gaussian probability density function. This lower dimensional space is stochastic in nature. The Neural Network has weights and bias . The semantic meaning of the topics can be determined by top-ranking words in each topic. The latent representation learned by the model corresponds to topic mixture weights per document. A Softmax decoder reconstructs the document by independently generating the words. The decoder has weights and bias . The decoder gets an input the latent representation of documents (h) and reconstructs topic words matrix. Each column in the matrix represents a topic, while each row represents a word. The matrix values, for a given column, represents probability distribution of words for the topic.

MLP is a class of feedforward Artificial Neuron Network (ANN). It consists of at least three layers of nodes: input, output and a hidden layer. Except for the input node, each node is a neuron that uses a nonlinear activation function.

Training objective: Information is lost during the dimensionality reduction. We measure this using reconstruction log-likelihood log . The goal is to minimize the reconstruction error and Kullback-Leiber Divergence

Loss Function = - Reconstruction Likelihood + Regularization Term

The loss function for document is:

The first term is the reconstruction loss or expected negative log-likelihood of the ith document. The expectation (average) is taken with respect to encoder’s distribution over the latent representation. Remember that the encodings are stochastic in nature. This term encourages the decoder to reconstruct the data as close to the original input as possible.

The second term is for regularization, technique used to avoid overfitting. This is Kullback-Leibler divergence between , probability of latent representation of document given , and , probability of hidden representation. This divergence measures how much information is lost when using p to represent q.

In variational autoencoder, p follows standard normal distribution with mean zero and variance one. If encoder output representations h that are different than those from a normal distribution, it will receive a penalty.

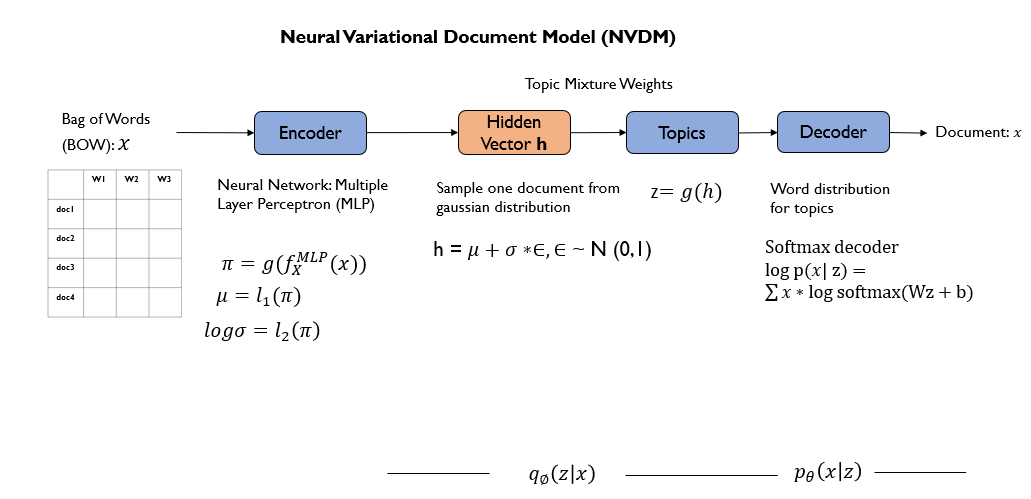
We train the variational autoencoder using gradient descent to minimize the loss function with respect to the parameters and . For SGD with step size , the encoder parameters are updated using 🡨

*Meaning of variables*

h: hidden variable, compressed representation of bag-of-words

X = bag-of-words representation of documents

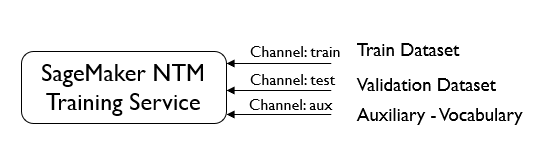
Different variations of encoding are generated, which are then decoded to generate words, generating documents



Softmax decoder is multinomial logistic regression, where we account for conditional probability of different topics. The transformation is effectively normalized exponential function used to highlight the largest values and suppress values which are significantly below the max value

For training the NTM algorithm, Amazon SageMaker can provision multiple virtual machines with either CPUs or GPUs for processing. You can specify the type of instances (memory or compute), number of instances, IAM role used to access data in S3, and the Uri of the training NTM image hosted by Elastic Container Registry (ECR) in your target region.

When fitting the NTM algorithm to the input dataset, you can optionally provide validation dataset and auxiliary input, which is a text file listing words in the vocabulary (vocab.txt). The auxiliary input is used built semantic meaning of the topics. Rather than listing word distribution of latent topics, auxiliary input will help list actual words associated with the topics.



Unlike supervised learning algorithms, there is no straight forward metric to measure accuracy of the algorithm. As indicated earlier, the main goal of training objective is to decrease the negative log likelihood of data and KL-Divergence between . Validation dataset is used to stop the training early and avoid overfitting. As the training is in progress for the specified number of epochs, the performance of the model is evaluated against validation dataset to see if the loss is decreasing. If the validation loss does not decrease for a specified number of epochs, then the training stops early.

There are several hyperparameters to control the training of NTM. These hyperparameters, which are external to the model, will control how the model is trained. Below is a listing of some of the hyperparameters. For a complete listing of NTM hyperparameters, check [here](https://docs.aws.amazon.com/sagemaker/latest/dg/ntm_hyperparameters.html)

**feature\_dim** – the dimension of the features, which is typically set to the vocabulary size

**num\_topics** – the number of topics to extract

**mini\_batch\_size** – the batch size for each worker instance. Note that in multi-GPU instances, this number will be further divided by the number of GPUs. Therefore, for example, if we plan to train on an 8-GPU machine (such as ml.p2.8xlarge) and want each GPU to have 1024 training examples per batch, mini\_batch\_size should be set to 8196.

**epochs** – the maximal number of epochs to train for, training may stop early

**num\_patience\_epochs** and **tolerance** control the early stopping behavior. Roughly speaking, the algorithm will stop training if within the last *num\_patience\_epochs* epochs there have not been improvements on validation loss. Improvements smaller than *tolerance* will be considered non-improvement.

**optimizer and learning\_rate** – by default we use the adadelta optimizer, and learning\_rate does not need to be set. The optimizer is used to optimize the NTM neural network ( ). For other optimizers, the choice of an appropriate learning rate may require experimentation.

The SageMaker also provides the model hosting service, providing a RESTful API endpoint for model inferences. The hosting service takes the number and type of instances to deploy the trained model.

Once the trained model is deployed as an endpoint, test data can be submitted in one of two formats – csv or recordio-protobuf.

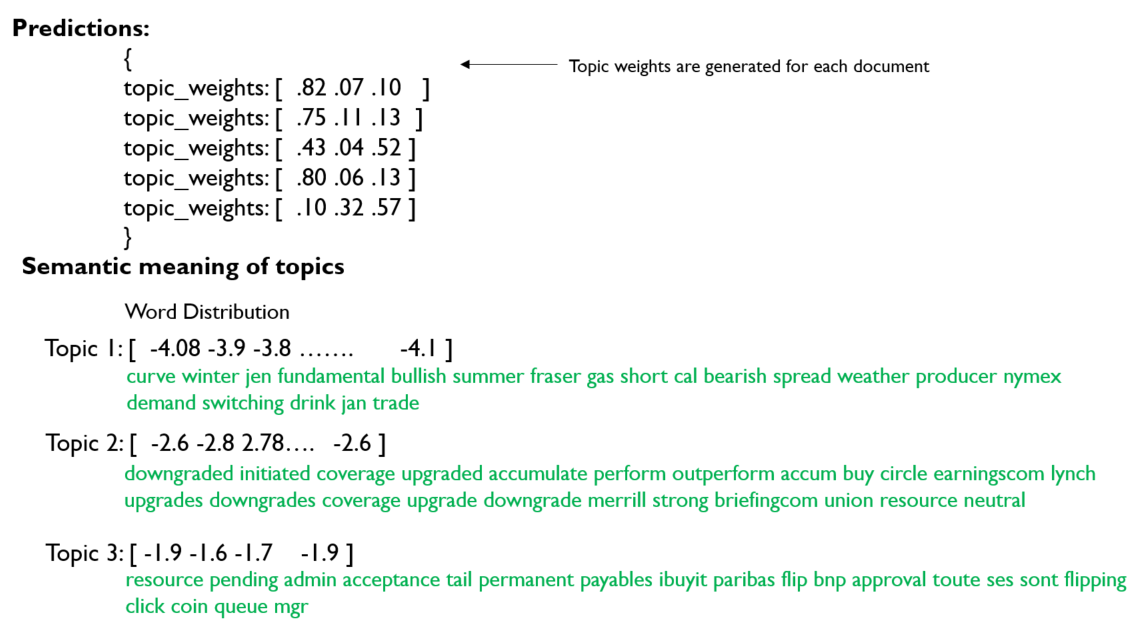
RecordIO wrapped Protobuf -- RecordIO is a binary data exchange format, where the data is divided into subsets, called records. Each record is prefixed with its length in bytes. Protobuf, protocol buffers, is like XML and JSON, in that it is used to serialize structured data. The binary serialization format is introduced by Google, with libraries in most languages such as Javascript and Python. It is faster (serialization performance) and strongly typed relative to JSON and XML.

Amazon SageMaker converts each record in the dataset into a binary representation of 4-byte floats.

For more information, refer to amazon docs on [*Common data formats - Training*](https://docs.aws.amazon.com/sagemaker/latest/dg/cdf-training.html)

Once the test data is submitted in either csv or recordio protobuf format, the NTM endpoint predicts topic mixture weights for each of the documents submitted to endpoint. For example, row 1 in the “predictions” dictionary corresponds to document 1. The document 1 has 3 topics, with weights .82, .07 and .10 (note that all the probabilities or weights add up to 1)

To develop semantic meaning for the topics, we retrieve word distribution for each of the topics. In the logs from running the training NTM job, in both Jupyter notebook and cloud watch, you will note that the words corresponding to the topic will be displayed. These words are retrieved from the vocabulary list (vocab.txt) provided while training.



**NTM Model Training using Enron Emails Dataset**

In the section below, we will look for topics in Enron emails. We’ve downloaded the data from University of California Irvine Machine Learning Repository. These emails are exchanged between Enron, an American energy company that ceased its operations in 2007 due to financial losses, and other parties that did business with the company.

After downloading the dataset from [here](https://archive.ics.uci.edu/ml/datasets/bag+of+words), we need to prepare it, so it can be consumed by Neural Topic Model (NTM). The dataset contains document ID, word ID, and count, which is the number of times that the word (identified through word ID) occurs in the document (identified through document ID).

Let’s begin with preparing the dataset for modeling:

* Create module named ‘bowemails.py’ to create bag-of-words representation of emails. We will transform the raw counts of word occurrence in emails to produce tf-idf score. Also, we will filter the number of emails and words we want process.

**Ingest bag-of-words dataset and store it on S3**

The dataset contains 39,861 emails and 28,101 unique words. We will work with subset of the emails – 3986 emails and 17,524 unique words. Additionally, we create a text file, vocab.txt, so the NTM model can report word distribution of a topic, including the words in the distribution instead of just IDs.

We will pivot the data frame containing emails to create a matrix, with rows representing emails and columns representing words. Count represents the scalar value for email and word combination – the frequency with which a word occurs in the email.

We will consolidate all the commonly used functions – bowemails.py

Let’s create a data frame containing ordered list of words in vocabulary. This function will be used to create vocabulary for the subset of emails we’ve selected.

import pandas as pd

import numpy as np

from sklearn.preprocessing import normalize

from scipy.sparse import csr\_matrix

import os

import boto3

import sagemaker

import io

import sagemaker.amazon.common as smac

# def create\_vocab(vocab\_fn):

vocab = pd.read\_table(vocab\_fn, header=None)

vocab.columns = ['word']

# sort words first

vocab.sort\_values(by=['word'])

# assign index

vocab['word\_ID'] = range(1, len(vocab)+1)

return vocab

We will now select subset of emails from original dataset and create pivot table, with each row representing an email and each column representing a word in the vocabulary.

# Create bow representation of emails and persist vocab file (filtered)

def prepare\_bow\_vocab(input\_fn, percent\_emails, vocab\_ip\_fn, vocab\_op\_fn):

df\_emails = pd.read\_table(input\_fn

, compression='gzip'

, header=None

, sep=' '

, skiprows = 3)

df\_emails.columns = ['email\_ID','word\_ID','count']

# Filter emails to reduce data size

len\_emails = len(df\_emails['email\_ID'].unique())

fil\_len\_emails = int(len\_emails \* percent\_emails)

# Get list of all email IDs

emailID\_list = df\_emails['email\_ID'].unique()

# Get a % of the original emails to avoid memory errors

emailID\_list = emailID\_list[0:fil\_len\_emails]

# Filter the original dataframe to only include filtered email\_id and word\_id combinations

df\_emails = df\_emails[df\_emails['email\_ID'].isin(emailID\_list)]

# Pivot email and word ID combinations

pvt\_emails = pd.pivot\_table(df\_emails, values='count', index='email\_ID', columns=['word\_ID'], fill\_value=0)

Then, we will create vocabulary (list of words) for the subset of emails selected.

# Create filtered list of vocabulary

df\_vocab = create\_vocab(vocab\_ip\_fn)

#Only retrieve words that are part of filtered emails dataframe

vocab\_fil = pd.merge(df\_vocab, df\_emails, on='word\_ID', how='inner' )

words = vocab\_fil['word'].unique()

#Create vocabulary for the filtered dataset

with open(vocab\_op\_fn, 'w') as f:

for item in words:

f.write("%s\n" % item)

return pvt\_emails

Our assumption is that words that occur frequently in an email and less frequently in other emails are the ones that are important for discovering topics. While words the occur frequently in all the emails may not be important for discovering topics. To account for this, we multiply the count value with a weight, TF-IDF (term frequency – inverse document frequency)

Bag\_of\_words is pivoted dataframe.

# Convert simple word counts per email to weighted word counts

# Weight is determined by term freq and inverse document freq

# Words that appear frequently within an email but less frequently across all emails are indicative of topics unique to the email in question

def TF\_IDF(bag\_of\_words):

no\_emails = len(bag\_of\_words)

dict\_IDF = {name: np.log(float(no\_emails) / (1+len(bag\_of\_words[bag\_of\_words[name] > 0]))) for name in bag\_of\_words.columns}

new\_BOW = pd.DataFrame()

for name in bag\_of\_words.columns:

new\_BOW[name] = bag\_of\_words[name] \* dict\_IDF[name]

return(new\_BOW)

In addition, we will partition the training data for SageMaker to parallel process the parts of training data. In the case of distributed training, we have multiple EC2 instances training on shards of training data, increasing training speed.

We will use write\_spmatrix\_to\_sparse\_tensor function from SageMaker Python SDK to convert a sparse matrix into a protobuf-encoded (mechanism for serializing structured data) multi-dimensional sparse array, an efficient format required by NTM (Neural Topic Model)

# Partition NTM compatible dataset for training

def split\_convert\_upload(sparray, bucket, prefix, fname\_template='data\_part{}.pbr', n\_parts=2):

chunk\_size = sparray.shape[0] // n\_parts

for i in range(n\_parts):

# Calculate start and end indices

start = i\*chunk\_size

end = (i+1)\*chunk\_size

if i+1 == n\_parts:

end = sparray.shape[0]

# Convert sparse matrix to sparse tensor (record io protobuf) - a format required by NTM algorithm

# pbr - Amazon Record Protobuf format

buf = io.BytesIO()

smac.write\_spmatrix\_to\_sparse\_tensor(array=sparray[start:end], file=buf, labels=None)

buf.seek(0)

# Upload to s3 location specified by bucket and prefix

fname = os.path.join(prefix, fname\_template.format(i))

boto3.resource('s3').Bucket(bucket).Object(fname).upload\_fileobj(buf)

***Bringing it all together***

### *Prepare training, validation and test dataset:*

Prepare bag-of-words representation of emails.

ip\_fn = 'data/docword.enron.txt.gz'

percent\_emails = .10 # get only a x% of emails to avoid memory errors

vocab\_ip\_fn = 'data/vocab.enron.txt'

vocab\_op\_fn = 'data/vocab.txt'

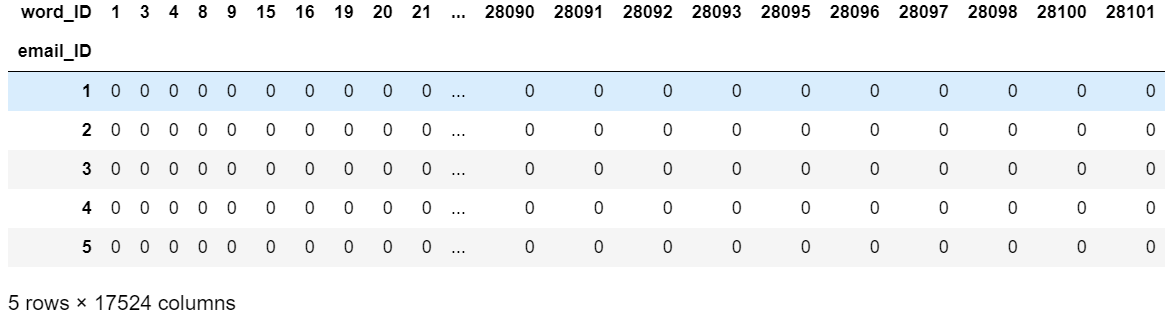
#Get bag-of-words from input of enron emails

# We will filter emails to reduce data size

# Create vocabulary based on the subset of emails that will be sent to training

pvt\_emails = prepare\_bow\_vocab(ip\_fn, percent\_emails, vocab\_ip\_fn, vocab\_op\_fn)

pvt\_emails.head()



*Term Frequency Inverse Document Frequency*

tfidf\_emails = TF\_IDF(pvt\_emails)

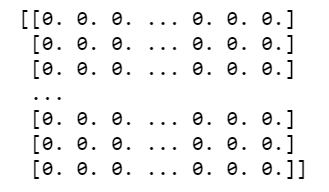
Create efficient representation (compressed sparse row) of bag of words

# convert pivoted dataframe to compressed sparse row matrix

# compressed sparse row matrix contains row pointer, column index and values

sparse\_emails = csr\_matrix(pvt\_emails, dtype=np.float32)

print(sparse\_emails[:16].toarray())



We partition the compressed sparse row (CSR) matrix into training, validation and test datasets.

The NTM algorithm, as well as other first-party SageMaker algorithms, accepts data in RecordIO Protobuf format as detailed above.

n\_train = int(0.8 \* sparse\_emails.shape[0])

# split train and test

train\_vectors = sparse\_emails[:n\_train, :]

test\_vectors = sparse\_emails[n\_train:, :]

# further split test set into validation set and test set

n\_test = test\_vectors.shape[0]

val\_vectors = test\_vectors[:n\_test//2, :]

test\_vectors = test\_vectors[n\_test//2:, :]

print(train\_vectors.shape, test\_vectors.shape, val\_vectors.shape)



The partitioned data is now stored into an S3 bucket. A folder called *EnronEmails* is created, within which *train, val, output, and aux* sub-folders are created.

role = get\_execution\_role()

# provide your bucket name here

#bucket = '<bucket-name>'

bucket = 'ai-in-aws'

prefix = 'enronemails'

train\_prefix = os.path.join(prefix, 'train')

val\_prefix = os.path.join(prefix, 'val')

output\_prefix = os.path.join(prefix, 'output')

aux\_prefix = os.path.join(prefix, 'aux')

s3\_train\_data = os.path.join('s3://', bucket, train\_prefix)

s3\_val\_data = os.path.join('s3://', bucket, val\_prefix)

s3\_aux\_data = os.path.join('s3://', bucket, aux\_prefix)

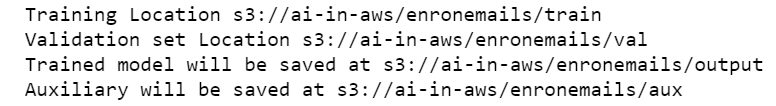
output\_path = os.path.join('s3://', bucket, output\_prefix)

print('Training Location', s3\_train\_data)

print('Validation set Location', s3\_val\_data)

print('Trained model will be saved at', output\_path)

print('Auxiliary will be saved at', s3\_aux\_data)



We partition the list of emails into chunks, so the Neural Topic Model can run training on multiple chunks of emails in parallel. *write\_spmatrix\_to\_sparse\_tensor* from SageMaker API is used to convert sparse matrix into RecordIO Protobuf format.

[*Protocol buffers format*:](https://docs.aws.amazon.com/sagemaker/latest/dg/cdf-inference.html) An array containing numeric values is treated as an instance containing single dense vector. dataElement = [1.5, 16.0, 14.0, 23.0]

It will be converted to the following representation by the SDK. *converted* = { "features": { "values": dataElement } }. Make 3 parts of training dataset before uploading the same to s3 bucket. Since the validation dataset is not huge, it is uploaded as is to the corresponding bucket.

# Convert compressed sparse row matrix to recordio-wrapped-protobuf format

# RecordIO is used to efficiently load large datasets (data can be read continuously and stored in a compressed format)

split\_convert\_upload(train\_vectors, bucket=bucket, prefix=train\_prefix, fname\_template='data\_part{}.pbr', n\_parts=3)

split\_convert\_upload(val\_vectors, bucket=bucket, prefix=val\_prefix, fname\_template='val\_part{}.pbr', n\_parts=1)

**Train Neural Topic Model in SageMaker**

We use SageMaker Python SDK will be used to train and deploy NTM. The SDK is used to convert scipy sparse matrix to RecordIO Protobuf format. The NTM algorithm, as well as other built-in algorithms, accept data in RecordIO Protobuf format.

SageMaker uses Amazon Elastic Container Registry (ECR) docker container to host the NTM training image. The ECR containers are currently available for SageMaker NTM training in different regions

Here, we get the container URI (uniform resource identifier) of the NTM docker image.

container = get\_image\_uri(boto3.Session().region\_name, 'ntm')

We will provision 2 training instances of type ml.c4.xlarge to run NTM on the Enron emails dataset.

sess = sagemaker.Session()

ntm = sagemaker.estimator.Estimator(container,

role,

train\_instance\_count=2,

train\_instance\_type='ml.c4.xlarge',

output\_path=output\_path,

sagemaker\_session=sess)

**Set Hyperparameters**

**feature\_dim** – the feature dimension, it should be set to the vocabulary size

**num\_topics** – the number of topics to extract

**mini\_batch\_size** – the batch size for each worker instance. Note that in multi-GPU instances, this number will be further divided by the number of GPUs. Therefore, for example, if we plan to train on an 8-GPU machine (such as ml.p2.8xlarge) and want each GPU to have 1024 training examples per batch, mini\_batch\_size should be set to 8196.

epochs – the maximal number of epochs to train for, training may stop early

**num\_patience\_epochs** and tolerance control the early stopping behavior. Roughly speaking, the algorithm will stop training if within the last num\_patience\_epochs epochs there have not been improvements on validation loss. Improvements smaller than `tolerance` will be considered non-improvement.

**optimizer and learning\_rate** – by default we use the adadelta optimizer, and learning\_rate is not required for this optimizer. For other optimizers, the choice of an appropriate learning rate may require experimentation.

num\_topics = 3

vocab\_size = 17524 # from shape from pivoted emails dataframe

ntm.set\_hyperparameters(num\_topics=num\_topics, feature\_dim=vocab\_size, mini\_batch\_size=30, epochs=100, num\_patience\_epochs=5, tolerance=.001)

# Upload vocabulary file to auxiliary folder on S3 bucket -- this is used to identify words associated with latent topics

aux\_path = s3\_aux\_data + "/"

!aws s3 cp $vocab\_op\_fn $aux\_path

s3\_train = s3\_input(s3\_train\_data, distribution='ShardedByS3Key', content\_type='application/x-recordio-protobuf')

s3\_val = s3\_input(s3\_val\_data, distribution='FullyReplicated',

content\_type='application/x-recordio-protobuf')

s3\_aux = s3\_input(s3\_aux\_data, distribution='FullyReplicated', content\_type='text/plain')

Train the NTM by sending data through 3 channels: train, validation and auxiliary

ntm.fit({'train': s3\_train, 'validation': s3\_val, 'auxiliary': s3\_aux})

*Metrics from training data are listed below:*

The Neural topic model optimizes across several epochs by minimizing loss in:

1. Building stochastic representation of emails (topic embeddings)
   1. This loss is measured with Kullback-Leibler Divergence – which is relative entropy, a measure of how one probability distribution is different from a second, a proxy probability distribution)
2. Reconstructing original emails from topic embeddings
   1. This loss is called *reconstruction loss*

Total Loss = Reconstruction Loss + KL Divergence Loss

8.41 = 8.24 + 0.16

*Model Performance Evaluation Metrics*

To evaluate the performance of Neural Topic model, we will look at 3 metrics:

1. Word Embedding Topic Coherence Metric (WETC) – measure semantic similarity of top words in each topic. A good quality model will have top words that are located close to each other in lower dimensional space. To locate words in lower dimensional space, pre-trained word embeddings from GloVe (Global Vectors) are used.
2. Topic Uniqueness (TU) – measures uniqueness of the topics generated. The measure is inversely proportional to the number of times a word appears across all the topics. For example, if a word appears in only one topic, then the uniqueness of the topic is high (which is 1). However, if a word appears across, say, 5 topics, then uniqueness measure is .2 (1 divide by 5). To calculate, topic uniqueness across all the topics, we average TU measures across all topics
3. Perplexity is a statistical measure of how well a probability model predicts a sample (validation dataset). After training, perplexity of the trained model is computed on validation dataset (performance of the trained model on validation dataset)
   1. The lower the perplexity the better, maximizing the accuracy in the validation dataset

The highlighted section reflects the 3 topics uncovered and their corresponding words

Loss (name: value) total: 8.40675995656

Loss (name: value) kld: 0.160669060242

Loss (name: value) recons: 8.24609097212

Loss (name: value) logppx: 8.40675995656

#quality\_metric: host=algo-2, epoch=56, validation total\_loss <loss>=8.40675995656

Loss of server-side model: 8.40675995656

Best model based on early stopping at epoch 56. Best loss: 8.40675995656

For the best model, *Word Embedding Topic Coherence (WETC)* is .34 and *Topic* *Uniqueness (TU) is 1*

Topics from epoch:final (num\_topics:3) [wetc 0.34, tu 1.00]:

Topic # 1 is measuring high on both semantic similarity of top words identified (0.61) and topic uniqueness (1)

[0.61, 1.00] sont ses ciapres lexpediteur destinataires etablis lintention erreur toutes recevez aurait titre ete jointes declinent lhypothese linternet lintegrite dassurer modifie

Topic # 2 – top words identified seem to be repeating across three topics.

[0.27, 1.00] clicking web chart represent affiliates contained sources prohibited accurate solicitation based officer instruction delete corp reliable offer immediately andor link

Topic # 3 – this topic uniqueness and semantic similarity among top words is less relative to those of topics 1 and 2.

[0.15, 1.00] tail flip head coin flipping bidoffer resource pending pao eva yippee request approval playing create sooo hookie double walk volunteer

**Deploy the NTM Model and Review Inference**

The deploy function of SageMaker Python SDK Estimator will provide instances to host the trained model as an endpoint.

ntm\_predictor = ntm.deploy(initial\_instance\_count=1, instance\_type='ml.m4.xlarge')

**Inference with CSV**

The NTM endpoint takes input either in csv or Record IO Protobuf format. Here we will pass first 5 emails from test dataset to the endpoint.

ntm\_predictor.content\_type = 'text/csv'

ntm\_predictor.serializer = csv\_serializer

ntm\_predictor.deserializer = json\_deserializer

test\_data = np.array(test\_vectors.todense())

test\_data[:5]

The predictor returned from the deploy function is used to obtain topic mixture weights for each of the five emails submitted

results = ntm\_predictor.predict(test\_data[1:5])

print(results)

predictions = np.array([prediction['topic\_weights'] for prediction in results['predictions']])

print(predictions)

[[0.07802384 0.51283914 0.40913707]

[0.06476374 0.55751461 0.37772167]

[0.07701793 0.24741225 0.67556983]

[0.0907187 0.66870773 0.24057359]]

Here, we will graph the learned topic probabilities per each email (of the 5 emails selected). Each topic’s weight (probability) in a given email is graphed.

import matplotlib.pyplot as plt

%matplotlib inline

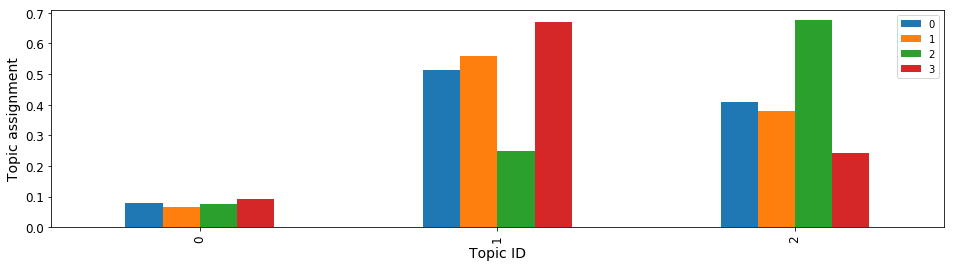
fs=12

df=pd.DataFrame(predictions.T)

df.plot(kind='bar', figsize=(16,4), fontsize=fs)

plt.ylabel('Topic assignment', fontsize=fs+2)

plt.xlabel('Topic ID', fontsize=fs+2)



As is evident from the results above, topic 1 is pretty dominant in three emails, while topic 2 is dominant in email #3. On the other hand, topic 0 is dormant across all the emails selected.

**Exploring the train model**

In the section below, we will unpack the trained model that we saved to the output path

import mxnet as mx

import os

import boto3

import matplotlib.pyplot as plt

%matplotlib inline

Obtain the latest trained job name

current\_job\_name = ntm.latest\_training\_job.job\_name

model\_path = os.path.join(output\_prefix, current\_job\_name, 'output/model.tar.gz')

model\_path

Download the trained model

boto3.resource('s3').Bucket(bucket).download\_file(model\_path, 'downloaded\_model.tar.gz')

Unzip the serialized model

!tar -xzvf 'downloaded\_model.tar.gz'

!unzip -o model\_algo-1

Load parameters of the trained model

model = mx.ndarray.load('params')

Retrieving word distributions for each of the latent topics

W = model['arg:projection\_weight']

[[-3.9556365 -2.319879 -1.8877467]

[-3.960328 -2.4875867 -1.6623772]

[-3.9587383 -2.4153535 -1.744642 ]

...

[-3.968517 -2.3106682 -1.9013762]

[-3.9398718 -2.3236659 -1.9011943]

[-3.9703078 -2.3161588 -1.893697 ]]

<NDArray 14142x3 @cpu(0)>

Now, let’s create a word cloud for each of the topics

Iterate through the vocabulary list to create dictionary of key (wordID) value (word) pairs

!pip install wordcloud

import wordcloud as wc

# Create vocabulary list

vocab\_list = pd.read\_table('vocab.txt', header=None)

vocab\_list = vocab\_list[0].tolist()

len(vocab\_list)

word\_to\_id = dict()

for i, v in enumerate(vocab\_list):

#print("Index and Value", i, v)

word\_to\_id[v] = i

limit = 24

n\_col = 4

counter = 0

For each of the topics synthesized, get the word distribution. Use softmax function to assign probability for each of the word associated with the topic

The sum of all the probabilities of words associated with each topic should add up to 1

plt.figure(figsize=(20,16))

for ind in range(num\_topics):

if counter >= limit:

break

title\_str = 'Topic{}'.format(ind)

pvals = mx.nd.softmax(mx.nd.array(W[:, ind])).asnumpy()

print("Printing pvals: ", len(pvals))

word\_freq = dict()

for k in word\_to\_id.keys():

i = word\_to\_id[k]

word\_freq[k] =pvals[i]

wordcloud = wc.WordCloud(background\_color='white').fit\_words(word\_freq)

plt.subplot(limit // n\_col, n\_col, counter+1)

plt.imshow(wordcloud, interpolation='bilinear')

plt.axis("off")

plt.title(title\_str)

#plt.close()

counter +=1



Based on the top-ranking words in each of the topic, we can determine the topic being discussed in the emails

**All words are in french -- topic**

Topic 1: sont ses ciapres lexpediteur destinataires etablis lintention erreur toutes recevez aurait titre ete jointes declinent lhypothese linternet lintegrite dassurer modifie

**Perhaps related to Analysis of data**

Topic 2: clicking web chart represent affiliates contained sources prohibited accurate solicitation based officer instruction delete corp reliable offer immediately andor link

**Perhaps related to Risk and Probability**

Topic 3: tail flip head coin flipping bidoffer resource pending pao eva yippee request approval playing create sooo hookie double walk volunteer

**References**

<https://aws.amazon.com/blogs/machine-learning/introduction-to-the-amazon-sagemaker-neural-topic-model/>

<https://arxiv.org/pdf/1511.06038.pdf>

<https://www.doc.ic.ac.uk/~js4416/163/website/nlp/>

<https://jaan.io/what-is-variational-autoencoder-vae-tutorial/>

<https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf>

<http://blog.echen.me/2011/08/22/introduction-to-latent-dirichlet-allocation/>