Machine Learning for Big Data Project CSC6515

Team members:

- Rajesh Kumar Gupta Lakshminarayan Gupta (B00791207)
- Vismay Revankar (B00813441)

In [7]:
```
# Generalized imports
import pandas as pd
import re, datetime
import numpy as np
import warnings, logging
import re
from datetime import datetime
import logging
import json
import os
import pickle as pkl
from wordcloud import WordCloud
from textblob import TextBlob
import logging, string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
import nltk
nltk.download('stopwords')


warnings.filterwarnings("ignore")
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

In [10]:
```
from google.colab import drive
drive.mount('/content/drive')
import os
os.chdir("/content/drive/My Drive/Colab Notebooks/ML for BD proj")
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly
(https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly)

Enter your authorization code:
··········
Mounted at /content/drive
```

In [0]:
```
# Visualization imports
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

In [14]:
```
# ML package imports
import sklearn
import keras
import tensorflow as tf
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from nmf import NMF
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.svm import SVC, LinearSVC
from sklearn.multiclass import OneVsRestClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.kernel_approximation import Nystroem
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import cross_val_score, KFold
from xgboost import XGBClassifier
from keras.models import Sequential
from keras.layers import Dense
from keras import layers
from sklearn.metrics import silhouette_samples, silhouette_score
```

```
Using TensorFlow backend.
```

The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you [upgrade (https://www.tensorflow.org/guide/migrate)](https://www.tensorflow.org/guide/migrate) now or ensure your notebook will continue to use TensorFlow 1.x via the `%tensorflow_version 1.x` magic: [more info (https://colab.research.google.com/notebooks/tensorflow_version.ipynb)](https://colab.research.google.com/notebooks/tensorflow_version.ipynb).

Utility functions

Some of the utility functions that will be used across the project are written in a separate file called "utility_functions.py" for reusability. In this notebook, however, running this cell should be enough to use them inside the project.

In [0]:
```
#===========================================
#           UTILITY FUNCTIONS              #
#===========================================
# Configuring the logger to record logs across the project
def configure_logger():
    current_datetime = datetime.now().strftime("%d-%m-%Y %H-%M-%S")
    fpath = os.path.join(os.getcwd(), os.path.join("logs", "multi-label-c
    log_file = fpath
    log_stream = 'sys.stdout'
    log_level = logging.DEBUG
    log_format = "%(asctime)s %(levelname)s: %(message)s"
    log_date_format = "%m/%d/%Y %H:%M:%S"
    logging.basicConfig(filename=log_file, level=log_level, format=log_fo
```

```
# Simple function to write to a text file
def write_file(text, filename):
    with open(filename,"w") as f:
        f.write(text)

# Simple function read a text file
def read_file(filename):
    with open(filename, "r") as f:
        data = f.read()
    return data

# Standard function to pickle .py objects
def pickle(obj, filename, foldername):
    PICKLE_PATH = os.path.join(os.getcwd(), foldername, filename + ".pkl
    with open(PICKLE_PATH, "wb") as f:
        pkl.dump(obj, f)

# Standard function to unpickle files to .py objects
def unpickle(filename, foldername):
    UNPICKLE_PATH = os.path.join(os.getcwd(), foldername, filename + ".pl
    with open(UNPICKLE_PATH, "rb") as f:
        data = pkl.load(f)
    return data

# Writing to a JSON file
def write_JSON(obj, filename, foldername):
    WRITE_PATH = os.path.join(os.getcwd(), foldername, filename + ".json
    with open(WRITE_PATH, "w") as f:
        json.dump(obj, f)

# Reading from a JSON file
def read_JSON(filename, foldername):
    READ_PATH = os.path.join(os.getcwd(), foldername, filename + ".json"
    with open(READ_PATH, "r") as f:
        data = json.load(f)
    return data

# Generating word cloud for a given dataframe
def generate_word_cloud(obj, column_name=None):
    if type(obj)==pd.core.frame.DataFrame or column_name:
        text = " ".join(obj[column_name].tolist())
    else:
        text = obj
    word_cloud = WordCloud().generate(text)
    return word_cloud

# Mention type of record
def get_label_type(obj):
    label_type = "multi-label" if len(obj)>1 else "uni-label"
    return label_type
```

```
# Return the polarity of text
def get_polarity(obj):
    blob = TextBlob(obj)
    return blob.sentiment.polarity

# Return the sentiment of text
def get_sentiment(obj):
    sentiment = "positive" if obj>0 else "negative"
    return sentiment

# Get all possible topics -> fn specific to Reuters
def get_unique_values(df, column_name="bip:topics"):
     return {each_value for each_row in df[column_name].values for each_
```

Data cleaning functions

Custom functions that help in cleaning of a dataframe have been written inside this file as a class for better readability and reusability.

In [0]:
```
class DataCleaning:

    stop = stopwords.words('english')
    ps = PorterStemmer()
    wnl = WordNetLemmatizer()

    def __init__(self):
        self.js = re.compile(r'<script.*?>.*?</script>')
        self.css = re.compile(r'<style.*?>.*?</style>')
        self.html = re.compile(r'<.*?>')
        self.braces = re.compile(r'{.*?}')
        self.spl_symbols = re.compile(r'&\S*?;|[\\\/\_\(\)\|\>\<\%]|\.as
        self.spaces = re.compile(r'\s+')
        self.urls = re.compile(r'https?\:\/\/.*?\s')
        self.non_alpha = re.compile(r'[^a-zA-Z\s]')
        self.extra_spaces = re.compile(r'\s+')

    def clean_metadata(self, df, columns=None):
        logging.info("="*15+"Cleaning metadata"+"="*15)
        if columns is not None:
            df = df[columns].to_frame().applymap(lambda x: self.clean_me
            return df.T.iloc[0,:] # df.squeeze() will also do
        return self.meta_data.sub(" ", str(df))

    def clean_extra_spaces(self, df, columns=None):
        logging.info("="*15+"Cleaning extra whitespaces"+"="*15)
        if columns is not None:
            df = df[columns].to_frame().applymap(lambda x: self.clean_ex
            return df.T.iloc[0,:] # df.squeeze() will also do
```

```
            return df.T.iloc[0,:] # df.squeeze() will also do
        return self.extra_spaces.sub(" ", str(df)).strip()

    def clean_js(self, df, columns=None):
        """
        Aliter : PLEASE NOTE THAT THIS CAN BE AN ALITER TO ALL FUNCTIONS
        if columns is not None:
            y=[]
            for x in columns:
                y.append(self.js.sub(" ", str(df[x])))
            return pd.Series(y)
        """
        logging.info("="*15+"Cleaning JS"+"="*15)
        if columns is not None:
            df = df[columns].to_frame().applymap(lambda x: self.clean_js
            return df.T.iloc[0,:] # df.squeeze() will also do
        return self.js.sub(" ", str(df))

    def clean_non_alpha(self, df, columns=None):
        logging.info("="*15+"Cleaning Non-alphabet characters"+"="*15)
        if columns is not None:
            df = df[columns].to_frame().applymap(lambda x: self.clean_no
            return df.T.iloc[0,:] # df.squeeze() will also do
        return self.non_alpha.sub("", str(df))

    def clean_html(self, df, columns=None):
        logging.info("="*15+"Cleaning HTML"+"="*15)
        if columns is not None:
            df = df[columns].to_frame().applymap(lambda x: self.clean_ht
            return df.T.iloc[0,:] # df.squeeze() will also do
        return self.html.sub(" ", str(df))

    def clean_css(self, df, columns=None):
        logging.info("="*15+"Cleaning CSS"+"="*15)
        if columns is not None:
            df = df[columns].to_frame().applymap(lambda x: self.clean_cs
            return df.T.iloc[0,:] # df.squeeze() will also do
        return self.css.sub(" ", str(df))

    def clean_braces(self, df, columns=None):
        logging.info("="*15+"Cleaning braces"+"="*15)
        if columns is not None:
            df = df[columns].to_frame().applymap(lambda x: self.clean_br
            return df.T.iloc[0,:] # df.squeeze() will also do
        return self.braces.sub(" ", str(df))

    def clean_special_symbols(self, df, columns=None):
        logging.info("="*15+"Cleaning special symbols"+"="*15)
        if columns is not None:
            df = df[columns].to_frame().applymap(lambda x: self.clean_sp
            return df.T.iloc[0,:] # df.squeeze() will also do
```

```
        return self.spl_symbols.sub(" ", str(df))

    def clean_spaces(self, df, columns=None):
        logging.info("="*15+"Cleaning spaces"+"="*15)
        if columns is not None:
            df = df[columns].to_frame().applymap(lambda x: self.clean_spa
            return df.T.iloc[0,:] # df.squeeze() will also do
        return self.spaces.sub(" ", str(df))

    def clean_urls(self, df, columns=None):
        logging.info("="*15+"Cleaning URLs"+"="*15)
        if columns is not None:
            df = df[columns].to_frame().applymap(lambda x: self.clean_url
            return df.T.iloc[0,:] # df.squeeze() will also do
        return self.urls.sub(" ", str(df))

    @classmethod
    def clean_stopwords(cls, df, columns=None):
        logging.info("="*15+"Cleaning stopwords"+"="*15)
        if columns is not None:
            df = df[columns].to_frame().applymap(lambda x: cls.clean_stop
            return df.T.iloc[0,:] # df.squeeze() will also do
        return " ".join([token for token in str(df).split() if token not

    @classmethod
    def stem_tokens(cls, df, columns=None):
        logging.info("="*15+"Stemming words"+"="*15)
        if columns is not None:
            df = df[columns].to_frame().applymap(lambda x: cls.stem_token
            return df.T.iloc[0,:] # df.squeeze() will also do
        return " ".join([cls.ps.stem(token) for token in str(df).split()

    @classmethod
    def lemmatize_tokens(cls, df, columns=None):
        logging.info("="*15+"Lemmatizing words"+"="*15)
        if columns is not None:
            df = df[columns].to_frame().applymap(lambda x: cls.lemmatize_
            return df.T.iloc[0,:] # df.squeeze() will also do
        return " ".join([cls.wnl.lemmatize(token) for token in str(df).sp

    @classmethod
    def preprocessing(cls, data):
        try:
            # Remove punctuations
            data = [str(_char) for _char in data if _char not in string.p
            # Changing back to text
            data = "".join(data)
        except Exception:
            data = str(0)
        return data
```

Reading reuters data from JSON

In [0]:
```
# Reading the reuters data from a JSON into a Pandas Dataframe
reuters_raw_df = pd.read_json("input_data/xmldata.json")
```

In [0]:
```
# Checking the dimensions of reuters data
reuters_raw_df.shape
```

Out[8]:
```
(48257, 6)
```

In [0]:
```
# Checking the first 5 rows of reuters data
reuters_raw_df.head(5)
```

Out[17]:

	headline	text	bip:topics	dc.date.pubished	itemid	XMLfilename
0	RTRS-NAB jumps 2.3 pct on buy-back plan.	\nShares in National Australia Bank Ltd jumped...	[C15, C152, C17, C171, C18, C181, CCAT]	1997-03-24	464661	464661newsML.xml
1	Care Group Inc Q4 shr loss vs profit.	\n(000's Omitted)\n\t\t\t\t\t YEAR END DECEM...	[C15, C151, C1511, CCAT]	1997-03-31	476242	476242newsML.xml
2	France urges Israel to stick by Oslo accords.	\nFrance urged Israel on Monday to stick to th...	[GCAT, GDIP]	1997-03-24	464382	464382newsML.xml
3	Former Mexican official denies taking drug bri...	\nA former Mexican deputy attorney general on ...	[GCAT, GCRIM]	1997-03-15	445205	445205newsML.xml
4	Krupp says Dortmund furnace won't be closed.	\nKrupp will not close at least part of its Do...	[C18, C181, C24, CCAT]	1997-03-20	457626	457626newsML.xml

In [0]:
```
reuters_raw_df.columns
```

Out[19]:
```
Index(['headline', 'text', 'bip:topics', 'dc.date.pubished', 'itemid',
       'XMLfilename'],
      dtype='object')
```

Cleaning the reuters data

In [0]:
```
# Creating an instance of the custom DataCleaning class
cleaner = DataCleaning()
```

In [0]:
```
columns_to_be_cleaned = ["text"]
```

In [0]:
```
# Remove the stopwords
reuters_raw_df[["stopwords_removed"]] = reuters_raw_df[columns_to_be_clea
```

In [0]:
```
# Lemmatize the tokens
reuters_raw_df[["lemmatized_version"]] = reuters_raw_df[["stopwords_remo
```

Reasons for lemmatizing tokens instead of stemming:

- Lemmatization produces meaningful tokens whereas stemming can convert appropriate words into meaningless words.
- Due to stemming, we lose meaning and we run the risk of losing the context as well. This can be harmful when we use deep learning models that try to make sense out of the context.
- The interpretability of a model is reduced.

In [0]:
```
# Punctuations are removed
reuters_raw_df[["punctuation_removed"]] = reuters_raw_df[["lemmatized_ve
```

The numerical data does not seem to add much value and this can greatly affect the dimensions, and hence the numerical data has been dropped.

In [0]:
```
# Non-alphabets dropped
reuters_raw_df[["non_alpha_removed"]] = reuters_raw_df[["punctuation_rem
```

In [0]:
```
reuters_raw_df[["extra_spaces_removed"]] = reuters_raw_df[["non_alpha_re
```

In [0]:
```
reuters_raw_df[["refined_text"]] = reuters_raw_df[["extra_spaces_removed
```

In [0]:
```
reuters_clean_df = pd.DataFrame(columns = ["input", "target"])
```

In [0]:
```
reuters_clean_df[["input", "target"]] = reuters_raw_df[["refined_text",
```

In [0]:
```
reuters_clean_df["input"][0]
```

Out[39]:
```
'shares national australia bank ltd jumped cent percent a early monday
afternoon trade announcing plan buy back percent million ordinary shar
es nab cent percent day risen cent earlier trade a however share price
dipped back a australian stock exchange announced correct figure milli
on shares earlier announced million share buyback analysts broker said
market rallied immediate prospect buyback given prospect buyback alrea
dy announced november last year nab said november annual result planne
d buy back million shares equal number share issued dividend reinvestm
ent bonus share share topup plans it said november planned buyback ext
ra million share time raise us million issue exchange tier two capital
nab conducted issue socalled exchangeable capital units excaps success
fully early month ended raising us billion strong demand nab said anno
unced detail buyback excap issue complete it said million buyback comp
rised million announced november plus million share offset dividend re
investment plan share topup bonus share issues the groups strong capit
al position combined flexibility provided excap issue continued genera
tion substantial capital reserve enables group deliver value sharehold
er implementation buyback program nab managing director don argus said
statement he said group would continue pursue development capital mana
gement initiatives the buyback scheduled begin april end october whene
ver million share bought back whichever first sydney newsroom'
```

In [0]:
```
# Pickling the clean data for future use
pickle(reuters_clean_df, "reuters_clean_df", "clean_data")
```

In [0]:
```
# Loading the clean data from the pickled file
reuters_clean_df = unpickle("reuters_clean_df", "clean_data")
```

In [19]:
```
reuters_clean_df.shape
```

Out[19]:
```
(48257, 2)
```

EDA - Exploratory Data Analysis

This is done post cleaning because we did not want to repeat the analysis before and after cleaning. And the cleaning steps that we've done are standard. Thus, the exploratory analysis post cleaning gives us better insight into the data.

Frequently occuring words in a word cloud

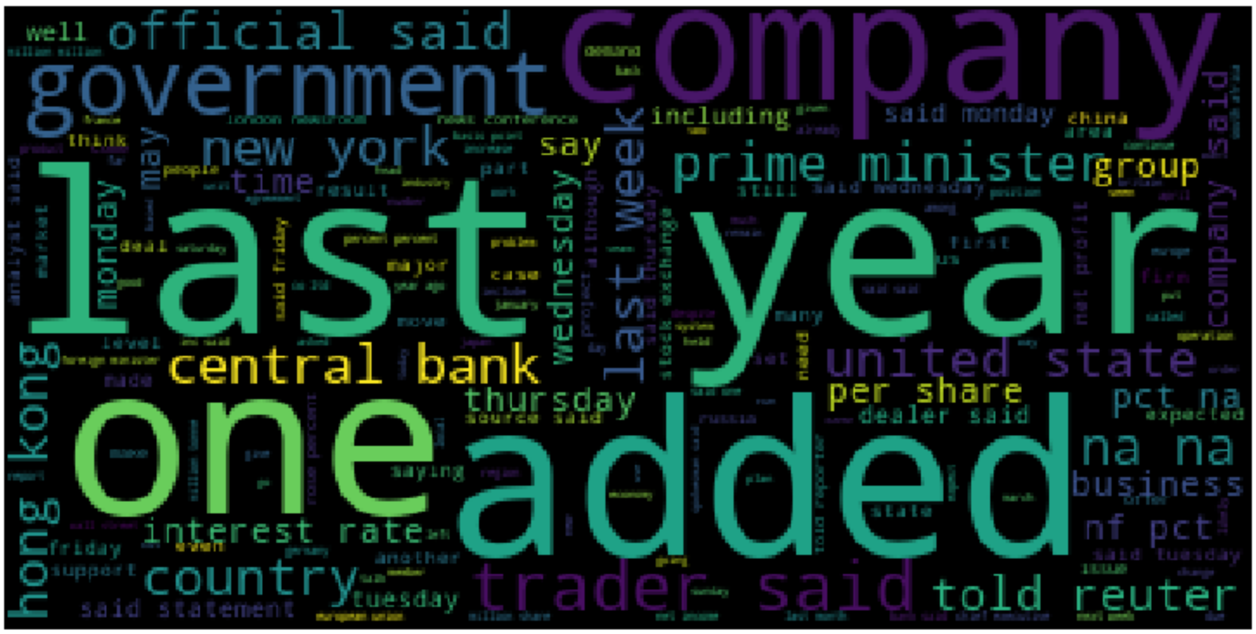
In [0]:
```
word_cloud_obj = generate_word_cloud(reuters_clean_df, column_name="input
```

In [0]:
```
pickle(word_cloud_obj, "word_cloud_obj", "utility_objects")
```

In [0]:
```
word_cloud_obj = unpickle("word_cloud_obj", "utility_objects")
```

In [0]:
```
# Visualizing the word cloud
plt.figure(figsize = (10, 10))
plt.axis("off")
plt.tight_layout(pad=0)
plt.imshow(word_cloud_obj)
```

Out[19]: <matplotlib.image.AxesImage at 0x1496d4ef0>



Inference

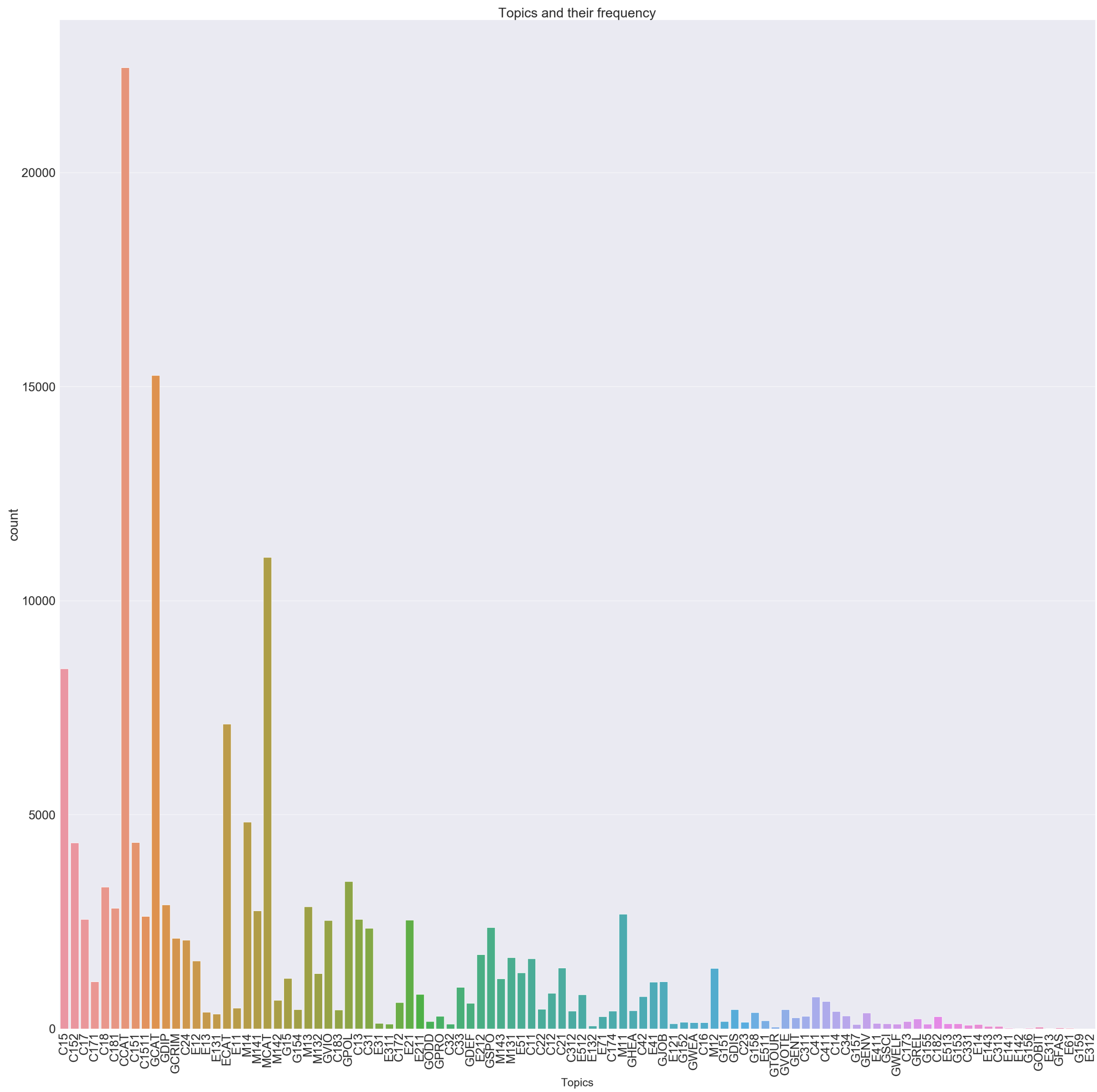
- Clearly, as we can see, the terms like "last year", "added", "company", "hong kong" are the frequently occuring unigrams and bigrams in the corpus after cleaning.

Frequent topics in a bar graph

In [0]:
```
# Getting the redundant topics from the corpus to calculate frequency
topics = [each_topic for each_topic_list in reuters_clean_df["target"].to
```

In [0]:
```
topic_df = pd.DataFrame(topics, columns=["topic"])
```

In [0]:
```
# Visualizing the word cloud
plt.figure(figsize = (50, 50))
sns.set(font_scale=3)
ax = sns.countplot(x="topic", data=topic_df)
ax.set_xlabel("Topics", fontsize=30)
plt.title("Topics and their frequency")
plt.xticks(rotation=90)
plt.show()
```



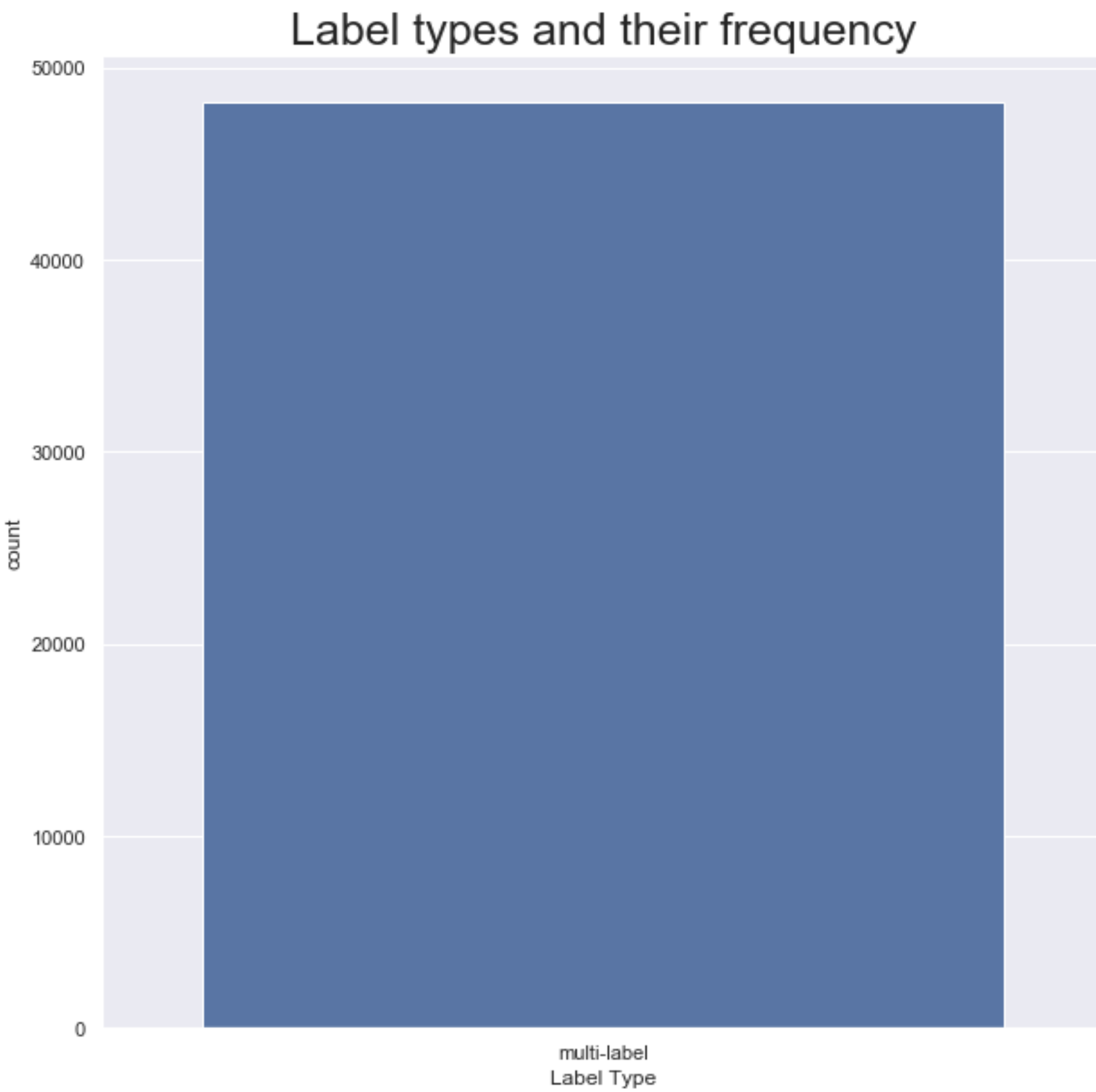
Inference:

- The most frequently occuring topics are CCAT, GCAT.
- This is significant because out of ~48000 records, ~24000 records seem to have CCAT in them. This means that every alternate record belongs to CCAT topic.

Number of records with single and multiple labels

In [0]:
```
# Getting the label type: uni-label or multi-label
reuters_clean_df["label_type"] = reuters_clean_df["input"].apply(get_labe
```

In [0]:
```
# Visualizing the word cloud
plt.figure(figsize = (10, 10))
sns.set(font_scale=1)
ax = sns.countplot(x="label_type", data=reuters_clean_df)
ax.set_xlabel("Label Type")
plt.title("Label types and their frequency", fontsize=25)
plt.show()
```



Inference:

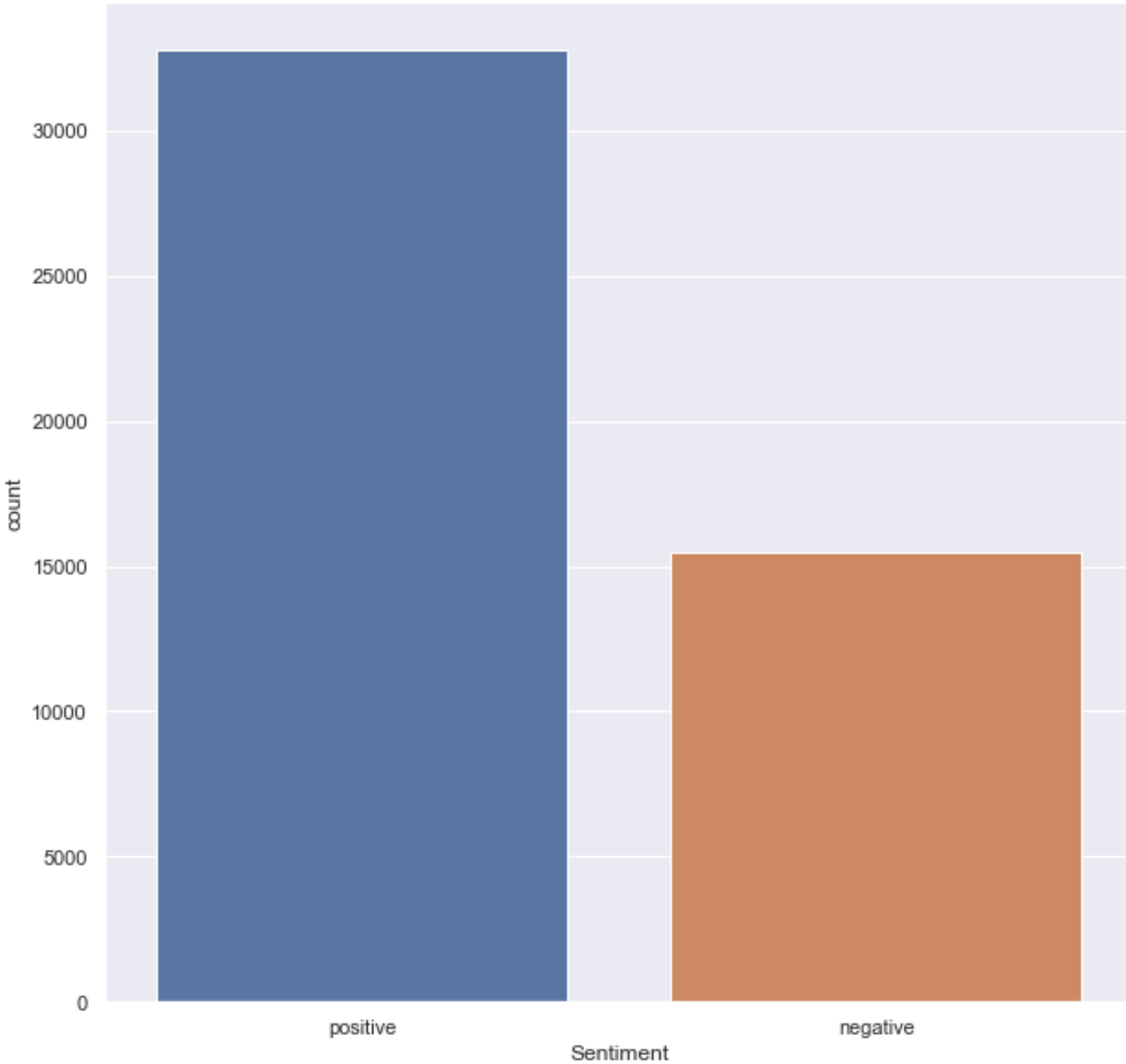
- The corpus given to us does not have uni-label records.
- This analysis was done to hold the records with uni-labels separately, if there were any. Then, we could have created a simple classifier that performs multi-class classification.
- Now, it is clear from this analysis that this is a multi-label classification problem.

A simple sentiment analysis to understand the corpus a little better!

In [0]:
```
# Getting the polarity score of input text
reuters_clean_df["polarity_score"] = reuters_clean_df["input"].apply(get_
```

In [0]:
```
# Getting the sentiment of the input text
reuters_clean_df["sentiment"] = reuters_clean_df["polarity_score"].apply
```

In [0]:
```
# Visualizing the word cloud
plt.figure(figsize = (10, 10))
sns.set(font_scale=1)
ax = sns.countplot(x="sentiment", data=reuters_clean_df)
ax.set_xlabel("Sentiment")
plt.show()
```



Inference:

- The number of records that carry a positive sentiment is ~32000 whereas the ones with a negative sentiment is ~16000.
- This analysis could have helped with a deeper understanding of the corpus and could be used to tune the model to predict the labels.
- However, there is not enough domain knowledge with us to use this as a part of our solution as of now and has been done only for exploratory purposes.

Words that makes up the positive corpus

In [0]:
```
positive_df = reuters_clean_df[reuters_clean_df["sentiment"]=="positive"
```

In [0]:
```
positive_word_cloud_obj = generate_word_cloud(positive_df, column_name="
```

In [0]:
```
# Visualizing the word cloud
plt.figure(figsize = (10, 10))
plt.axis### Word cloud that makes up the positive corpus("off")
plt.tight_layout(pad=0)
plt.imshow(positive_word_cloud_obj)
```

Out[27]: <matplotlib.image.AxesImage at 0x148bfb358>

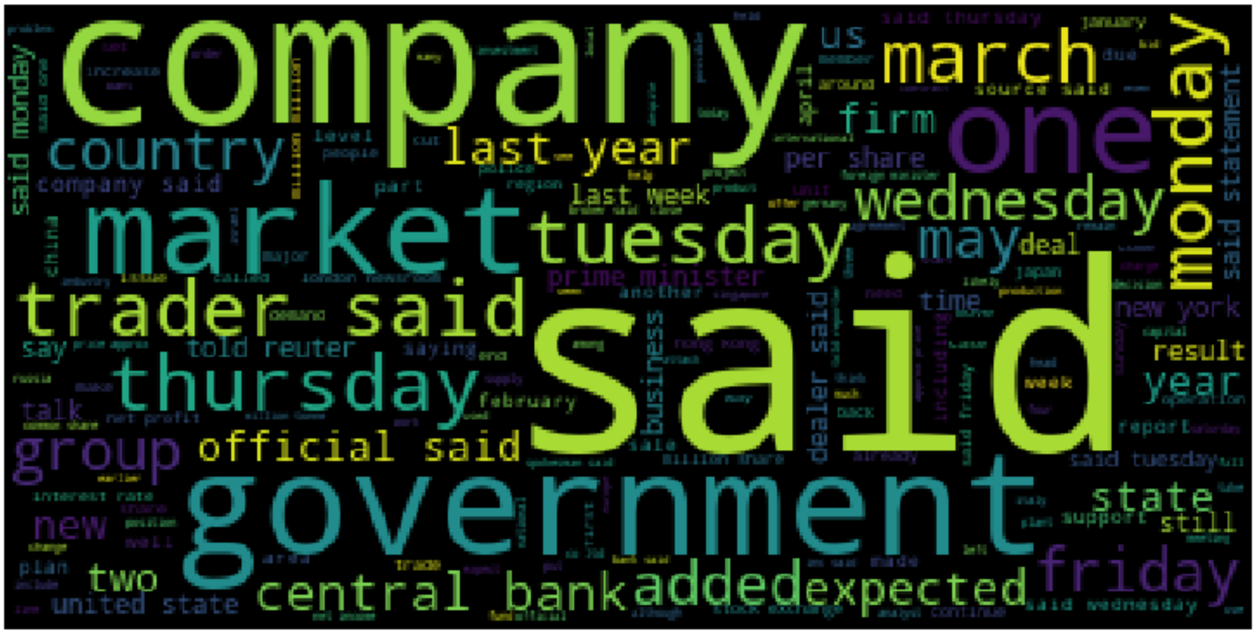
Words that makes up the negative corpus

In [0]:
```
negative_df = reuters_clean_df[reuters_clean_df["sentiment"]=="negative"
```

In [0]:
```
negative_word_cloud_obj = generate_word_cloud(negative_df, column_name=":
```

In [0]:
```
# Visualizing the word cloud
plt.figure(figsize = (10, 10))
plt.axis("off")
plt.tight_layout(pad=0)
plt.imshow(negative_word_cloud_obj)
```

Out[33]: <matplotlib.image.AxesImage at 0x148b34a90>



In [0]:
```
pickle(reuters_clean_df, "reuters_clean_eda_df", "clean_data")
```

In [0]:
```
reuters_clean_df = unpickle("reuters_clean_eda_df", "clean_data")
```

Feature Extraction

Firstly, a bag of words model is created from the cleaned dataframe. Following this, a TF-IDF vectorizer is used.

TF-IDF is a technique that gives a statistical measure of how important a term is to the corpus. While computing the TF, each term is given equal importance whereas in the IDF computation, the rare ones across the corpus are given more importance.

[https://www.cs.toronto.edu/~hinton/science.pdf (https://www.cs.toronto.edu/~hinton/science.pdf)](https://www.cs.toronto.edu/~hinton/science.pdf) -> 2000 features

In [0]:

```
%%time
# Combination of CountVectorizer and TfidfTransformer
tf_idf_vectorizer = sklearn.feature_extraction.text.TfidfVectorizer(max_
tf_idf_matrix = tf_idf_vectorizer.fit_transform(reuters_clean_df['input'
print(f"The feature names are:\n {tf_idf_vectorizer.get_feature_names()}

# Save the tf-idf matrix to a file
pickle(tf_idf_matrix, "tf_idf_2000_features", "tf_idf_models")
```

```
The feature names are:
 ['aa', 'aaa', 'ability', 'able', 'about', 'abroad', 'accept', 'accept
ed', 'access', 'accident', 'accord', 'according', 'account', 'accounti
ng', 'accuracy', 'accused', 'achieve', 'acquire', 'acquired', 'acquisi
tion', 'across', 'act', 'action', 'active', 'activity', 'actual', 'add
', 'added', 'adding', 'addition', 'additional', 'address', 'administra
tion', 'administrative', 'advance', 'advantage', 'advertising', 'affai
r', 'affairs', 'affect', 'affected', 'africa', 'african', 'after', 'af
ternoon', 'ag', 'again', 'age', 'agency', 'agenda', 'agent', 'ago', 'a
gree', 'agreed', 'agreement', 'agricultural', 'agriculture', 'ahead',
'aid', 'aim', 'aimed', 'air', 'aircraft', 'airline', 'airlines', 'airp
ort', 'alan', 'albania', 'albanian', 'albanians', 'all', 'allegation',
'alleged', 'alliance', 'allow', 'allowed', 'allowing', 'ally', 'almost
', 'along', 'already', 'also', 'alternative', 'although', 'aluminium',
'always', 'am', 'ambassador', 'america', 'american', 'americans', 'ami
d', 'among', 'amount', 'amsterdam', 'an', 'analyst', 'analysts', 'and'
, 'angeles', 'announce', 'announced', 'announcement', 'annual', 'anoth
er', 'anything', 'appeal', 'appeared', 'application', 'appointed', 'ap
proach', 'approval', 'approved', 'approx', 'apr', 'april', 'arab', 'ar
```

Q2) Clustering using KMeans ++

In [0]:
```
def run_kmeans(k, tf_idf_matrix):
    kmeans = KMeans(init='k-means++', n_clusters=k, verbose=True, n_init
    return kmeans
```

Running Kmeans for k=3,4,5..102 (no of unique topics)

For technical showcasing, we will test out various values of k starting from 3 and plot an elbow graph.

[http://www.cs.toronto.edu/~hinton/absps/science_som.pdf (http://www.cs.toronto.edu/~hinton/absps/science_som.pdf)](http://www.cs.toronto.edu/~hinton/absps/science_som.pdf) \n [http://www.cs.toronto.edu/~hinton/science.pdf (http://www.cs.toronto.edu/~hinton/science.pdf)](http://www.cs.toronto.edu/~hinton/science.pdf)

In [0]:
```
tf_idf_matrix = unpickle("tf_idf_2000_features", "tf_idf_models")
```

In [0]:
```
kvalues = range(3, 103, 10)
kmeans_meta = {}
for k in kvalues:
    kmeans = run_kmeans(k, tf_idf_matrix)
    kmeans_meta[f"kmeans_{k}"] = kmeans
    pickle(kmeans, f"kmeans_{k}_rev", "k_means_models")
```

```
Initialization complete
Iteration  0, inertia 83046.467
Iteration  1, inertia 44810.459
Iteration  2, inertia 44402.757
Iteration  3, inertia 44244.467
Iteration  4, inertia 44194.623
Iteration  5, inertia 44169.592
Iteration  6, inertia 44151.370
Iteration  7, inertia 44134.832
Iteration  8, inertia 44115.705
Iteration  9, inertia 44098.194
Iteration 10, inertia 44086.325
Iteration 11, inertia 44078.047
```

In [0]:
```
kvalues = range(3, 103)
kmeans_meta = {}
for k in kvalues:
    try:
        kmeans_meta[f"kmeans_{k}"] = unpickle(f"kmeans_{k}_rev", "k_mean
    except:
        continue
```

In [0]:
```
kmeans_meta
```

Out[25]:
```
{'kmeans_3': KMeans(algorithm='auto', copy_x=True, init='k-means++', m
ax_iter=300,
        n_clusters=3, n_init=10, n_jobs=None, precompute_distances='au
to',
        random_state=None, tol=0.0001, verbose=True),
 'kmeans_4': KMeans(algorithm='auto', copy_x=True, init='k-means++', m
ax_iter=300,
        n_clusters=4, n_init=3, n_jobs=None, precompute_distances='aut
o',
        random_state=None, tol=0.0001, verbose=True),
 'kmeans_5': KMeans(algorithm='auto', copy_x=True, init='k-means++', m
ax_iter=300,
        n_clusters=5, n_init=3, n_jobs=None, precompute_distances='aut
o',
        random_state=None, tol=0.0001, verbose=True),
 'kmeans_6': KMeans(algorithm='auto', copy_x=True, init='k-means++', m
ax_iter=300,
        n_clusters=6, n_init=3, n_jobs=None, precompute_distances='aut
o',
        random_state=None, tol=0.0001, verbose=True),
 'kmeans_7': KMeans(algorithm='auto', copy_x=True, init='k-means++', m
ax_iter=300,
        n_clusters=7, n_init=1, n_jobs=None, precompute_distances='aut
o',
        random_state=None, tol=0.0001, verbose=True),
 'kmeans_17': KMeans(algorithm='auto', copy_x=True, init='k-means++',
max_iter=300,
        n_clusters=17, n_init=1, n_jobs=None, precompute_distances='au
to',
        random_state=None, tol=0.0001, verbose=True),
 'kmeans_27': KMeans(algorithm='auto', copy_x=True, init='k-means++',
max_iter=300,
        n_clusters=27, n_init=1, n_jobs=None, precompute_distances='au
to',
        random_state=None, tol=0.0001, verbose=True),
 'kmeans_37': KMeans(algorithm='auto', copy_x=True, init='k-means++',
max_iter=300,
        n_clusters=37, n_init=1, n_jobs=None, precompute_distances='au
to',
        random_state=None, tol=0.0001, verbose=True),
 'kmeans_47': KMeans(algorithm='auto', copy_x=True, init='k-means++',
max_iter=300,
        n_clusters=47, n_init=1, n_jobs=None, precompute_distances='au
to',
        random_state=None, tol=0.0001, verbose=True),
 'kmeans_57': KMeans(algorithm='auto', copy_x=True, init='k-means++',
max_iter=300,
        n_clusters=57, n_init=1, n_jobs=None, precompute_distances='au
to',
        random_state=None, tol=0.0001, verbose=True),
```

```
 'kmeans_67': KMeans(algorithm='auto', copy_x=True, init='k-means++',
max_iter=300,
        n_clusters=67, n_init=1, n_jobs=None, precompute_distances='au
to',
        random_state=None, tol=0.0001, verbose=True),
 'kmeans_77': KMeans(algorithm='auto', copy_x=True, init='k-means++',
max_iter=300,
        n_clusters=77, n_init=1, n_jobs=None, precompute_distances='au
to',
        random_state=None, tol=0.0001, verbose=True),
 'kmeans_87': KMeans(algorithm='auto', copy_x=True, init='k-means++',
max_iter=300,
        n_clusters=87, n_init=1, n_jobs=None, precompute_distances='au
to',
        random_state=None, tol=0.0001, verbose=True),
 'kmeans_97': KMeans(algorithm='auto', copy_x=True, init='k-means++',
max_iter=300,
        n_clusters=97, n_init=1, n_jobs=None, precompute_distances='au
to',
        random_state=None, tol=0.0001, verbose=True)}
```

In [0]:

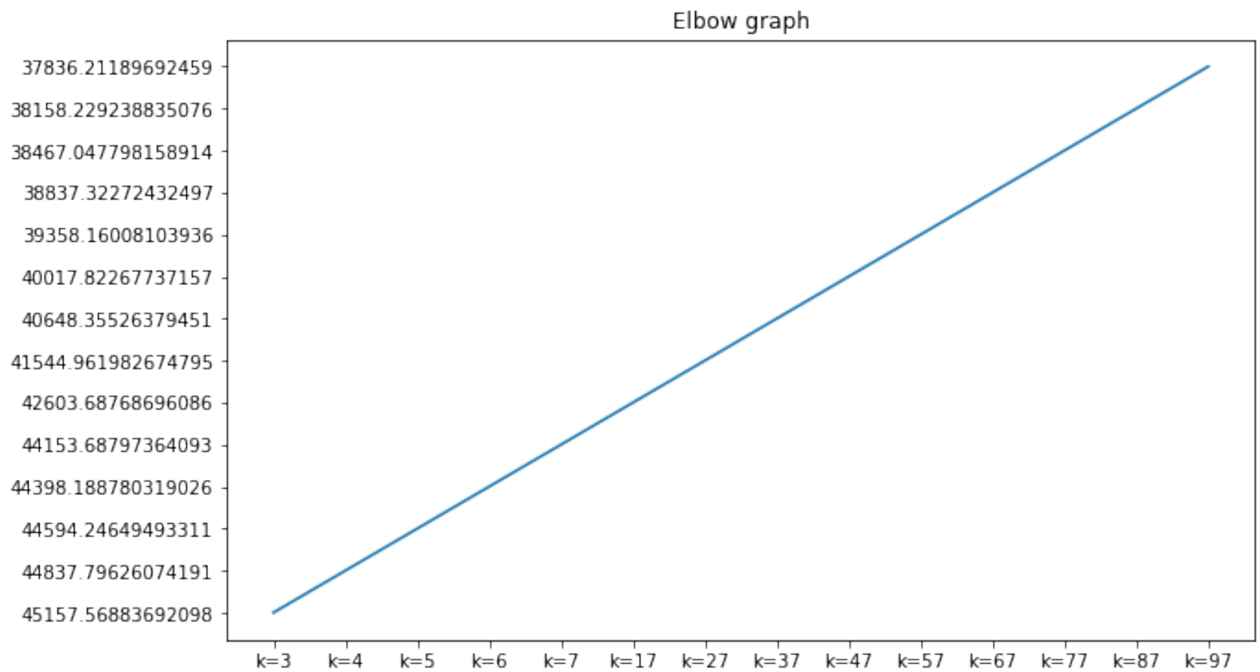
```
kvalues = [(int(re.sub("\D", "", i)), kmeans_meta[i].inertia_) for i in l
kvalues
```

Out[118]:

```
[(3, 45157.56883692098),
 (4, 44837.79626074191),
 (5, 44594.24649493311),
 (6, 44398.188780319026),
 (7, 44153.68797364093),
 (17, 42603.68768696086),
 (27, 41544.961982674795),
 (37, 40648.35526379451),
 (47, 40017.82267737157),
 (57, 39358.16008103936),
 (67, 38837.32272432497),
 (77, 38467.047798158914),
 (87, 38158.229238835076),
 (97, 37836.21189692459)]
```

Q3) Evaluation of Kmeans++ using Elbow Graph

In [0]:
```
x_axis = [f"k={k[0]}" for k in kvalues]
y_axis = [f"{k[1]}" for k in kvalues]
plt.figure(figsize=(10,6))
plt.plot(x_axis, y_axis)
plt.title("Elbow graph")
plt.show()
```



There is no elbow point in the above elbow graph. And this is understandable because most of the real world use cases will not provide an elbow point. Here, we choose to go with a value of 4.

The reason why we are going with a value of 4 is because according to Geoffrey Hinton's supporting material for the research paper [1] states that the reuters corpus fundamentally consists of 4 topics (4 clusters). Even generally, in kmeans, whenever the k value is unknown, one of the best way to find it would be to keep the use case in mind. For example, a T-shirt manufacturer using kmeans need not run an elbow method to find k. His use case is to group his customers into people wearning small, medium and large t-shirt sizes. He can directly go with a value of 3. Since, the project requires there to be a technical standpoint for the chosen value of k, elbow graph is drawn.

Q4) NMF - Feature Extraction [2][3]

Firstly, a brief introduction about NMF:

- Non-negative Matrix Factorization is a matrix decomposition technique used in topic modeling.
- In topic modeling, each document in a corpus is considered to be a mixture of several topics where each document is represented by the probability distribution of various topics that come together to form that document. And also, each topic is considered to be a mixture of various words in the corpus.
- NMF is constrained by the fact that the probability distribution or the W and H matrix is always non-negative and this makes perfect sense as even the worst tf-idf score of a word that isn't present in a document is non-negative.
- In short, the tf-idf matrix A(m rows x n cols) is broken down into a multiplication of W(m rows x r cols) and H(r rows x n cols) where there is some reconstruction error on multiplying W and H to get the resulting A.
- Since this is a NLP task, the use case of topic modeling was much more intuitive and we assumed it would provide us better results as far as dimensionality reduction is concerned.
- One of the reasons why we didn't go with autoencoders is because THEY ARE SIMPLY NOT USED IN THE INDUSTRY as other effective compression methods like JPEG are already present and needless to say, they are computationally expensive.
- We have not used the sklearn implementation of NMF as it does not use the GPU resources allotted by Google Colab. We went ahead and used a custom NMF written in Tensorflow.

Implementation details:

- As far as 2000 epochs are being iterated through to find W and H, we can see that the cost decreases very slowly post this. Hence, we stop and use the W as our input matrix to the text classifier.
- The reason why we are restricting the W matrix to **102 features** is because we assume each document is a mixture of probability distribution of the 102 unique classes present in the corpus.

What advantages does dimensionality reduction give?

- It greatly reduces the computation time.
- It reduces the memory load on the machine.
- It conveys substantial information in a compressed format.
- It also retains the most important features in a way.

In [0]:
```
nmf = NMF(max_iter=2000)
```

In [0]:
```
arr = tf_idf_matrix.todense()
```

In [0]:
```
W, H = nmf.fit_transform(arr, r_components=102,initW=False, givenW=False
```

```
Device mapping:
/job:localhost/replica:0/task:0/device:XLA_CPU:0 -> device: XLA_CPU de
vice
/job:localhost/replica:0/task:0/device:XLA_GPU:0 -> device: XLA_GPU de
vice
/job:localhost/replica:0/task:0/device:GPU:0 -> device: 0, name: Tesla
K80, pci bus id: 0000:00:04.0, compute capability: 3.7

|Epoch:    0  Cost= 45935.090
|Epoch:   10  Cost= 37883.430
|Epoch:   20  Cost= 34267.652
|Epoch:   30  Cost= 33436.383
|Epoch:   40  Cost= 33108.078
|Epoch:   50  Cost= 33002.457
|Epoch:   60  Cost= 32948.320
|Epoch:   70  Cost= 32917.164
|Epoch:   80  Cost= 32896.156
|Epoch:   90  Cost= 32880.777
|Epoch:  100  Cost= 32870.277
```

In [0]:
```
weights = {
    "W": W,
    "H": H
}
# pickle weights for future usage
##pickle(weights, "nmf_epoch_2000_weights", "nmf_models")
```

In [0]:
```
weights=unpickle("nmf_epoch_2000_weights", "nmf_models")
```

Q5) Getting the input and target ready for text classification

In [0]:
```
def extract_features(text_df):
    main_df = pd.DataFrame()
    mlb = MultiLabelBinarizer()
    y = mlb.fit_transform(text_df["target"])
    pickle(mlb, "mlb_model_latest", "fe_models")
    X = weights["W"]
    # Pandas Dataframe containing features and labels according to requi
    main_df["features"], main_df["labels"], main_df["cluster_label"] = li
    return main_df, X, y, mlb
```

In [0]:
```
# Get the input data
main_df, X, y, _ = extract_features(reuters_clean_df)
```

Dividing the dataset according to cluster it belongs

In [0]:
```
def create_dataset(main_df):
    datasets = {}
    for cluster_label in range(4):
        temp_df = main_df[main_df["cluster_label"]==cluster_label]
        datasets[f"data_{cluster_label}"] = {}
        datasets[f"data_{cluster_label}"]["X"] = np.array(list(temp_df["
        datasets[f"data_{cluster_label}"]["y"] = np.array(list(temp_df["
        datasets[f"data_{cluster_label}"]["shape"] = datasets[f"data_{clu
    return datasets
```

In [0]:
```
datasets = create_dataset(main_df)
```

In [0]:
```
datasets
```

Out[131]:
```
{'data_0': {'X': array([[0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
..., 0.0000000e+00,
          1.7804676e-05, 0.0000000e+00],
         [0.0000000e+00, 5.6537992e-05, 0.0000000e+00, ..., 0.0000000e
+00,
          2.3725903e-05, 0.0000000e+00],
         [0.0000000e+00, 0.0000000e+00, 0.0000000e+00, ..., 0.0000000e
+00,
          0.0000000e+00, 0.0000000e+00],
         ...,
         [0.0000000e+00, 0.0000000e+00, 0.0000000e+00, ..., 0.0000000e
+00,
          0.0000000e+00, 0.0000000e+00],
         [0.0000000e+00, 0.0000000e+00, 0.0000000e+00, ..., 3.2642843e
-06,
          1.5218067e-06, 0.0000000e+00],
         [4.5827278e-06, 0.0000000e+00, 0.0000000e+00, ..., 1.8159641e
```

```
-25,
          0.0000000e+00, 0.0000000e+00]], dtype=float32),
  'y': array([[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]]),
  'shape': (3199, 102)},
 'data_1': {'X': array([[0.0000000e+00, 0.0000000e+00, 5.4501033e-05,
..., 0.0000000e+00,
          0.0000000e+00, 0.0000000e+00],
         [1.5113309e-05, 0.0000000e+00, 1.1758035e-04, ..., 0.0000000e
+00,
          0.0000000e+00, 2.1797589e-03],
         [0.0000000e+00, 1.5628006e-05, 2.4113435e-05, ..., 4.6488029e
-05,
          0.0000000e+00, 0.0000000e+00],
         ...,
         [0.0000000e+00, 4.3997570e-05, 4.1825743e-04, ..., 0.0000000e
+00,
          0.0000000e+00, 1.6585293e-03],
         [0.0000000e+00, 5.9270013e-05, 0.0000000e+00, ..., 0.0000000e
+00,
          0.0000000e+00, 0.0000000e+00],
         [0.0000000e+00, 0.0000000e+00, 0.0000000e+00, ..., 2.3849750e
-06,
          0.0000000e+00, 0.0000000e+00]], dtype=float32),
  'y': array([[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]]),
  'shape': (15393, 102)},
 'data_2': {'X': array([[2.3999319e-05, 3.8407990e-05, 0.0000000e+00,
..., 1.1217865e-04,
          0.0000000e+00, 6.0658065e-05],
         [0.0000000e+00, 0.0000000e+00, 0.0000000e+00, ..., 0.0000000e
+00,
          0.0000000e+00, 0.0000000e+00],
         [2.9041045e-04, 0.0000000e+00, 2.6254763e-04, ..., 0.0000000e
+00,
          4.3989236e-05, 0.0000000e+00],
         ...,
         [0.0000000e+00, 0.0000000e+00, 1.0140719e-04, ..., 5.8143938e
-05,
          0.0000000e+00, 1.1156135e-04],
```

```
          [0.0000000e+00, 0.0000000e+00, 0.0000000e+00, ..., 0.0000000e
+00,
           1.3771405e-03, 0.0000000e+00],
          [0.0000000e+00, 0.0000000e+00, 0.0000000e+00, ..., 5.3438649e
-05,
           0.0000000e+00, 8.6400214e-06]], dtype=float32),
  'y': array([[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, 1]]),
  'shape': (8827, 102)},
 'data_3': {'X': array([[3.2753253e-10, 0.0000000e+00, 5.1967262e-35,
..., 1.8713799e-04,
           8.3519053e-06, 0.0000000e+00],
          [0.0000000e+00, 0.0000000e+00, 2.9703971e-28, ..., 0.0000000e
+00,
           0.0000000e+00, 0.0000000e+00],
          [0.0000000e+00, 0.0000000e+00, 8.3697505e-06, ..., 0.0000000e
+00,
           0.0000000e+00, 0.0000000e+00],
          ...,
          [0.0000000e+00, 0.0000000e+00, 0.0000000e+00, ..., 0.0000000e
+00,
           0.0000000e+00, 0.0000000e+00],
          [0.0000000e+00, 4.6890412e-04, 0.0000000e+00, ..., 0.0000000e
+00,
           2.2958020e-06, 0.0000000e+00],
          [0.0000000e+00, 0.0000000e+00, 1.9411206e-04, ..., 0.0000000e
+00,
           0.0000000e+00, 0.0000000e+00]], dtype=float32),
  'y': array([[0, 0, 0, ..., 0, 0, 0],
          [0, 0, 0, ..., 0, 0, 0],
          [0, 0, 0, ..., 0, 0, 1],
          ...,
          [0, 0, 1, ..., 0, 0, 0],
          [0, 0, 0, ..., 0, 0, 0],
          [0, 0, 0, ..., 0, 0, 0]]),
  'shape': (20838, 102)}}
```


In [0]:
```
pickle(datasets, "datasets", "input_data")
```

In [12]:
```
datasets
```

Out[12]:
```
{'data_0': {'X': array([[0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
..., 0.0000000e+00,
          1.7804676e-05, 0.0000000e+00],
         [0.0000000e+00, 5.6537992e-05, 0.0000000e+00, ..., 0.0000000e
+00,
          2.3725903e-05, 0.0000000e+00],
         [0.0000000e+00, 0.0000000e+00, 0.0000000e+00, ..., 0.0000000e
+00,
          0.0000000e+00, 0.0000000e+00],
         ...,
         [0.0000000e+00, 0.0000000e+00, 0.0000000e+00, ..., 0.0000000e
+00,
          0.0000000e+00, 0.0000000e+00],
         [0.0000000e+00, 0.0000000e+00, 0.0000000e+00, ..., 3.2642843e
-06,
          1.5218067e-06, 0.0000000e+00],
         [4.5827278e-06, 0.0000000e+00, 0.0000000e+00, ..., 1.8159641e
-25,
          0.0000000e+00, 0.0000000e+00]], dtype=float32),
```

Model building

In the assignment, we had used a Decision Tree classifier to classify the documents. Here, for each and every cluster, we will be building a deep neural network for performing multilabel classification. We are going to use an architecture of 200-200-102 for the given task. The last layer of this neural network will not have softmax activation function as it can only help in multiclass classification with one class outweighing the all other classes. Instead, we use sigmoid to produce the logit probabilities of each and every class for a document.

We will be using the keras library to take advantage of the underlying GPU architecture.

In [0]:
```
datasets = unpickle("datasets", "input_data")
```

In [0]:
```
def create_model(X, y):
  # create model
  model = Sequential()
  model.add(Dense(200, input_dim=datasets["data_0"]["shape"][1], activat
  model.add(Dense(200, activation='relu'))
  model.add(Dense(102, activation='sigmoid'))
  # Compile model
  model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['
  model.summary()
  # Fit the model
  model.fit(X, y, validation_split=0.3, epochs=20, batch_size=10)
  return model
```

In [0]:
```
def perform_cross_validation(datasets, cv=5):
  for i in range(4):
    print(f"for set: {i}")
    try:
      scores = []
      j = 0
      for train, test in KFold(5).split(datasets[f"data_{i}"]["X"]):
        print(f"for set: {i} {j}")
        model = create_model(datasets[f"data_{i}"]["X"][train], datasets
        score = model.evaluate(datasets[f"data_{i}"]["X"][test], dataset
        scores.append(score[1])
        j+=1
      datasets[f"data_{i}"]["cross_val_scores"] = np.array(scores)
      datasets[f"data_{i}"]["cross_val_variance"] = np.array(scores).var
    except Exception as e:
        print(e)
  return datasets
```

In [13]:
```
datasets = perform_cross_validation(datasets)
```

```
for set: 0
for set: 0 0
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/b
ackend/tensorflow_backend.py:66: The name tf.get_default_graph is depr
ecated. Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/b
ackend/tensorflow_backend.py:541: The name tf.placeholder is deprecate
d. Please use tf.compat.v1.placeholder instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/b
ackend/tensorflow_backend.py:4432: The name tf.random_uniform is depre
cated. Please use tf.random.uniform instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/o
ptimizers.py:793: The name tf.train.Optimizer is deprecated. Please us
e tf.compat.v1.train.Optimizer instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/b
```

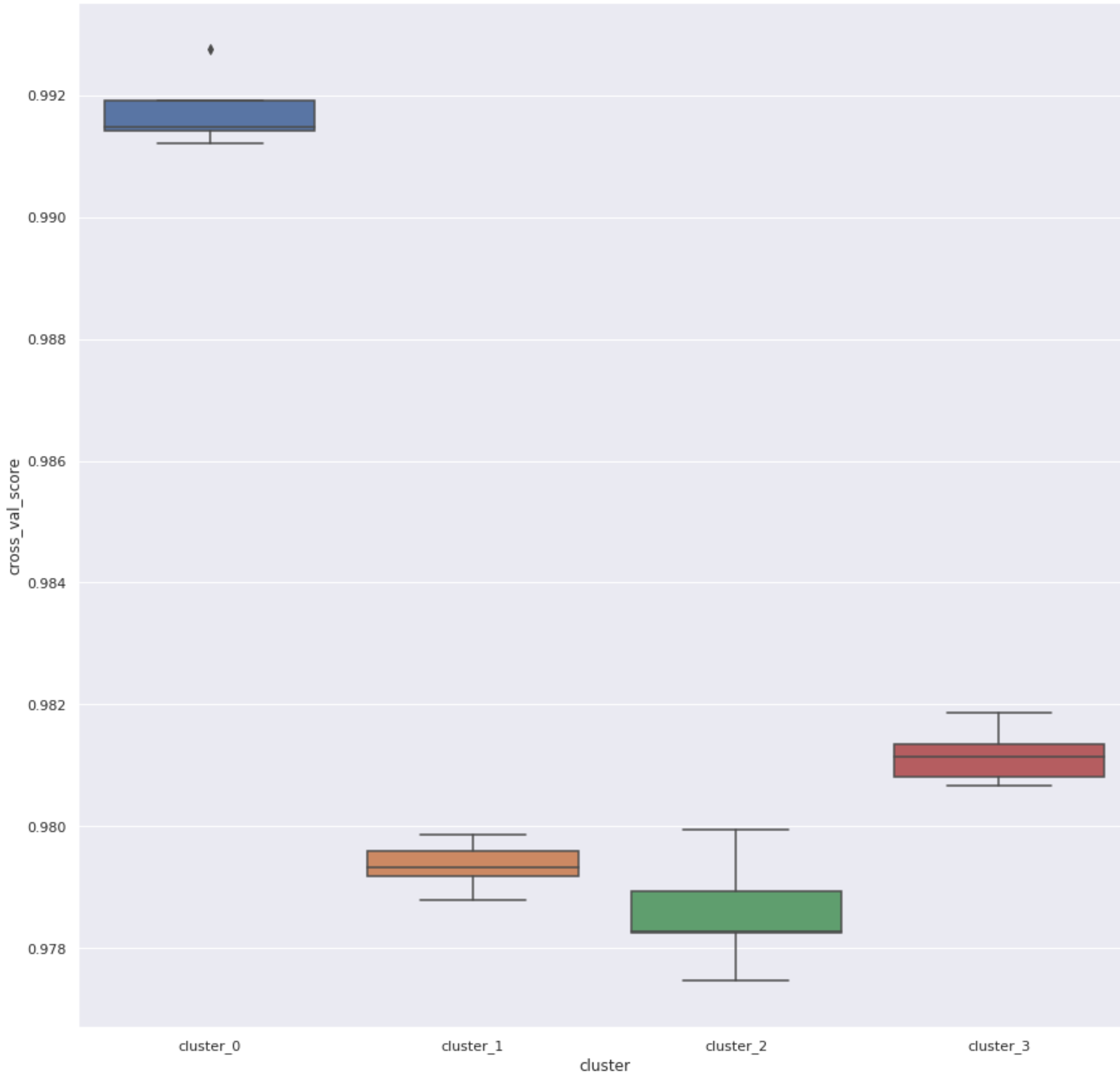
In [0]:
```
for i in range(4):
  datasets[f"cluster_{i}"] = datasets.pop(f"data_{i}")
```

In [0]:
```
df = []
for key, value in datasets.items():
    for x in datasets[key]["cross_val_scores"]:
        df.append([key, x])
df = pd.DataFrame(df, columns=["cluster", "cross_val_score"])
```

Box plot to display the cross val scores of a deep neural network

In [27]:
```
sns.set(rc={'figure.figsize':(15,15)})
sns.boxplot(x="cluster", y="cross_val_score", data=df)
```

Out[27]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f810a421438>
```



In [0]:
```
pickle(datasets, "datasets_final", "input_data")
```

In [0]:
```
final_data = unpickle("datasets_final", "input_data")
```

In [29]:
```
print("The average accuracy of model built for cluster 1 is: {}% with a v
print("The average accuracy of model built for cluster 2 is: {}% with a v
print("The average accuracy of model built for cluster 3 is: {}% with a v
print("The average accuracy of model built for cluster 4 is: {}% with a v
```

```
The average accuracy of model built for cluster 1 is: 99.1755880426166
5% with a variance of 3.014592374470503e-07
The average accuracy of model built for cluster 2 is: 97.9355307294027
1% with a variance of 1.3366198252313804e-07
The average accuracy of model built for cluster 3 is: 97.8575151923350
7% with a variance of 6.863095872869965e-07
The average accuracy of model built for cluster 4 is: 98.1165608630411
2% with a variance of 1.8378359242952318e-07
```

References:

[1] Hinton, Geoffrey E. and Ruslan Salakhutdinov. “Reducing the dimensionality of data with neural networks.” Science 313 5786 (2006): 504-7 .

[2] Tsuge, S. & Shishibori, M. & Kuroiwa, Shingo & Kita, K.. (2001). Dimensionality reduction using non-negative matrix factorization for information retrieval. Proc IEEE Int Conf on Systems, Man and Cybernetics. 2. 960 - 965 vol.2. 10.1109/ICSMC.2001.973042.

[3] "Eesungkim/NMF-Tensorflow". Github, 2019, https://github.com/eesungkim/NMF-Tensorflow/blob/master/nmf.py (https://github.com/eesungkim/NMF-Tensorflow/blob/master/nmf.py).

In [0]: