Topic Categorization

A documentation Submitted
in Complete Fulfillment of
the Requirements for the
Course of

Summer Internship in Celebal Technologies

In Third year – Sixth Semester ,
B-Tech CSE with specialization in
Artificial Intelligence and
Machine Learning

Under

Dr. Gopal Singh Phartiyal (Faculty Mentor)

Mr. Anurag Sharma (Team Lead)

By

500068187	**R177218004**	**Abhishek Gupta**
500068627	**R177218022**	**Archit Jain**
500067784	**R177218030**	**Ayush Joshi**
500065661	**R177218038**	**Gaurica Puri**

DEPARTMENT OF INFORMATICS
SCHOOL OF COMPUTER SCIENCE
UNIVERSITY OF PETROLEUM AND ENERGY STUDIES,

BIDHOLI, DEHRADUN, UTTRAKHAND, INDIA

7th June - 22nd July, 2021

Table of Contents

Acknowledgement

We would like to express our special thanks of gratitude to our UPES faculty mentor Dr. Gopal Singh Phartiyal and Celebal mentor Mr. Anurag Sharma for guiding us during the entire course of our internship and for providing the necessary information regarding the project.

We would also like to thank the supervisors from Celebal Mr. Sharthak Acharjee and Ms. Anjali Arora who time to time helped in solving our doubts and queries through virtual meetings and gave their full efforts in guiding the team in achieving the goal as well as provided encouragement to maintain our progress in track.

Last but not least we extend our gratitude to the Company Celebal Technologies for giving us this opportunity.

Company Overview

Company Name : Celebal Technologies

Celebal Technologies is a premier software services company in the field of Data Science, Big Data and Enterprise Cloud. Celebal Technologies helps you to discover the competitive advantage by employing intelligent data solutions using cutting-edge technology solutions that can bring massive value to your organization. The core offerings are around "Data to Intelligence", wherein company leverage data to extract intelligence and patterns thereby facilitating smarter and quicker decision making for clients.

Celebal Technologies solutions are powered by Robotics, Artificial Intelligence and Machine Learning algorithms which offer improved business efficiency in the interconnected world. Company's clients are Product ISVs and delivery organizations across the globe who use company's niche expertise in product development as well as Enterprise project implementations. Company have adopted the highest standards of service quality and operational excellence, enabling its clients across a wide range of industries to transform into a data-driven enterprise. Company's tailor-made solutions help enterprises maximize productivity, improve speed and accuracy.

With Celebal Technologies, who understands the core value of modern analytics over the enterprise, we help the business in improving business intelligence and more data-driven in architecting solutions. Company's analytics team caters to ad-hoc and define business analytics.

Project Title : Topic Categorization
Internship Duration : 45 days

Project Scope

In this project we had to Identify the topic of the given paragraph or article by picking up the keywords.

For that we required a dataset which had a collection of paragraphs.

Applications and use cases of topic categorization:

- Tagging content or products using categories as a way to improve browsing or to identify related content on your website. Platforms such as E-commerce, news agencies, content curators, blogs, directories, and likes can use automated technologies to classify and tag content and products.
- Text classification can also be used to automate CRM tasks. The text classifier is highly customizable and can be trained accordingly. The CRM tasks can directly be assigned and analysed based on importance and relevance. It reduces manual work and thus is high time efficient.
- A faster emergency response system can be made by classifying panic conversation on social media. Authorities can monitor and classify emergency situation to make a quick response if any such situation arises. This is a case of very selective classification. You can check out this study to read about an elaborated post on one such emergency response system.

Limitation: Since we will be training our model on few selected keywords/topics It may not give correct result if paragraph is of some different topic.

Project Description

Title: Topic Categorization

What is Text Classification?

Text classification is a machine learning technique that assigns a set of predefined categories to open-ended text. Text classifiers can be used to organize, structure, and categorize pretty much any kind of text – from documents, medical studies and files, and all over the web.

For example, new articles can be organized by topics; support tickets can be organized by urgency; chat conversations can be organized by language; brand mentions can be organized by sentiment; and so on.

Text classification is one of the fundamental tasks in natural language processing with broad applications such as sentiment analysis, topic labelling, spam detection, and intent detection.

Here's an example of how it works:

"The user interface is quite straightforward and easy to use."

A text classifier can take this phrase as an input, analyze its content, and then automatically assign relevant tags, such as UI and Easy To Use.

Input text is processed by a text classification model and delivers output tags.

Why is Text Classification Important?

It’s estimated that around 80% of all information is unstructured, with text being one of the most common types of unstructured data. Because of the messy nature of text, analyzing, understanding, organizing, and sorting through text data is hard and time-consuming, so most companies fail to use it to its full potential.

This is where text classification with machine learning comes in. Using text classifiers, companies can automatically structure all manner of relevant text, from emails, legal documents, social media, chatbots, surveys, and more in a fast and cost-effective way. This allows companies to save time analyzing text data, automate business processes, and make data-driven business decisions.

Why use machine learning text classification? Some of the top reasons:

Scalability

Manually analyzing and organizing is slow and much less accurate.. Machine learning can automatically analyze millions of surveys, comments, emails, etc., at a fraction of the cost, often in just a few minutes. Text classification tools are scalable to any business needs, large or small.

Real-time analysis

There are critical situations that companies need to identify as soon as possible and take immediate action (e.g., PR crises on social media). Machine learning text classification can follow your brand mentions constantly and in real time, so you'll identify critical information and be able to take action right away.

Consistent criteria

Human annotators make mistakes when classifying text data due to distractions, fatigue, and boredom, and human subjectivity creates inconsistent criteria. Machine learning, on the other hand, applies the same lens and criteria to all data and results. Once a text classification model is properly trained it performs with unsurpassed accuracy.

How Does Text Classification Work?

You can perform text classification in two ways: manual or automatic.

Manual text classification involves a human annotator, who interprets the content of text and categorizes it accordingly. This method can deliver good results but it's time-consuming and expensive.

Automatic text classification applies machine learning, natural language processing (NLP), and other AI-guided techniques to automatically classify text in a faster, more cost-effective, and more accurate manner.

In this guide, we're going to focus on automatic text classification.

There are many approaches to automatic text classification, but they all fall under three types of systems:

Rule-based systems

Machine learning-based systems

Hybrid systems

Rule-based systems

Rule-based approaches classify text into organized groups by using a set of handcrafted linguistic rules. These rules instruct the system to use semantically relevant elements of a text to identify relevant categories based on its content. Each rule consists of an antecedent or pattern and a predicted category.

Say that you want to classify news articles into two groups: Sports and Politics. First, you'll need to define two lists of words that characterize each group (e.g., words related to sports such as football, basketball, LeBron James, etc., and words related to politics, such as Donald Trump, Hillary Clinton, Putin, etc.).

Next, when you want to classify a new incoming text, you'll need to count the number of sport-related words that appear in the text and do the same for politics-related words. If the number of sports-related word appearances is greater than the politics-related word count, then the text is classified as Sports and vice versa.

For example, this rule-based system will classify the headline "When is LeBron James' first game with the Lakers?" as Sports because it counted one sports-related term (LeBron James) and it didn't count any politics-related terms.

Rule-based systems are human comprehensible and can be improved over time. But this approach has some disadvantages. For starters, these systems require deep knowledge of the domain. They are also time-consuming, since generating rules for a complex system can be quite challenging and usually requires a lot of analysis and testing. Rule-based systems are also difficult to maintain and don't scale well given that adding new rules can affect the results of the pre-existing rules.

Machine learning based systems

Instead of relying on manually crafted rules, machine learning text classification learns to make classifications based on past observations. By using pre-labeled examples as training data, machine learning algorithms can learn the different associations between pieces of text, and that a particular output (i.e., tags) is expected for a particular input (i.e., text). A "tag" is the pre-determined classification or category that any given text could fall into.

The first step towards training a machine learning NLP classifier is feature extraction: a method is used to transform each text into a numerical representation in the form of a vector. One of the most frequently used

approaches is bag of words, where a vector represents the frequency of a word in a predefined dictionary of words.

For example, if we have defined our dictionary to have the following words {This, is, the, not, awesome, bad, basketball}, and we wanted to vectorize the text “This is awesome,” we would have the following vector representation of that text: (1, 1, 0, 0, 1, 0, 0).

Then, the machine learning algorithm is fed with training data that consists of pairs of feature sets (vectors for each text example) and tags (e.g. sports, politics) to produce a classification model:

Training process in Text Classification
Once it’s trained with enough training samples, the machine learning model can begin to make accurate predictions. The same feature extractor is used to transform unseen text to feature sets, which can be fed into the classification model to get predictions on tags (e.g., sports, politics):

Prediction process in Text Classification
Text classification with machine learning is usually much more accurate than human-crafted rule systems, especially on complex NLP classification tasks. Also, classifiers with machine learning are easier to maintain and you can always tag new examples to learn new tasks.

Machine Learning Text Classification Algorithms

Some of the most popular text classification algorithms include the Naive Bayes family of algorithms, support vector machines (SVM), and deep learning.

Naive Bayes

The Naive Bayes family of statistical algorithms are some of the most used algorithms in text classification and text analysis, overall.

One of the members of that family is Multinomial Naive Bayes (MNB) with a huge advantage, that you can get really good results even when your dataset isn't very large (~ a couple of thousand tagged samples) and computational resources are scarce.

Naive Bayes is based on Bayes's Theorem, which helps us compute the conditional probabilities of the occurrence of two events, based on the probabilities of the occurrence of each individual event. So we are calculating the probability of each tag for a given text, and then outputting the tag with the highest probability.

The probability of A, if B is true, is equal to the probability of B, if A is true, times the probability of A being true, divided by the probability of B being true.

This means that any vector that represents a text will have to contain information about the probabilities of the appearance of certain words within the texts of a given category, so that the algorithm can compute the likelihood of that text belonging to the category.

Take a look at this blog post to learn more about Naive Bayes.

Support Vector Machines

Support Vector Machines (SVM) is another powerful text classification machine learning algorithm, because like Naive Bayes, SVM doesn't need much training data to start providing accurate results. SVM does, however, require more computational resources than Naive Bayes, but the results are even faster and more accurate.

In short, SVM draws a line or "hyperplane" that divides a space into two subspaces. One subspace contains vectors (tags) that belong to a group, and another subspace contains vectors that do not belong to that group.

The optimal hyperplane is the one with the largest distance between each tag. In two dimensions it looks like this:

Those vectors are representations of your training texts, and a group is a tag you have tagged your texts with.

As data gets more complex, it may not be possible to classify vectors/tags into only two categories. So, it looks like this:

But that's the great thing about SVM algorithms – they're "multi-dimensional." So, the more complex the data, the more accurate the results will be. Imagine the above in three dimensions, with an added Z-axis, to create a circle.

Mapped back to two dimensions the ideal hyperplane looks like this:

Deep Learning

Deep learning is a set of algorithms and techniques inspired by how the human brain works, called neural networks. Deep learning architectures offer huge benefits for text classification because they perform at super high accuracy with lower-level engineering and computation.

The two main deep learning architectures for text classification are Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN).

Deep learning is hierarchical machine learning, using multiple algorithms in a progressive chain of events. It's similar to how the human brain works when making decisions, using different techniques simultaneously to process huge amounts of data.

Deep learning algorithms do require much more training data than traditional machine learning algorithms (at least millions of tagged examples). However, they don't have a threshold for learning from training data, like traditional machine learning algorithms, such as SVM and NBeep learning classifiers continue to get better the more data you feed them with.

Dataset Description

Dataset Name: 20 Newsgroups Dataset

The 20 Newsgroups data set is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups. To the best of my knowledge, it was originally collected by Ken Lang, probably for his Newsweeder: Learning to filter netnews paper, though he does not explicitly mention this collection. The 20 newsgroups collection has become a popular data set for experiments in text applications of machine learning techniques, such as text classification and text clustering.

The whole dataset is organized into 20 newsgroups:

```
['alt.atheism',
 'comp.graphics',
 'comp.os.ms-windows.misc',
 'comp.sys.ibm.pc.hardware',
 'comp.sys.mac.hardware',
 'comp.windows.x',
 'misc.forsale',
 'rec.autos',
 'rec.motorcycles',
 'rec.sport.baseball',
 'rec.sport.hockey',
 'sci.crypt',
 'sci.electronics',
 'sci.med',
 'sci.space',
 'soc.religion.christian',
 'talk.politics.guns',
 'talk.politics.mideast',
 'talk.politics.misc',
 'talk.religion.misc']
```

Each record in the dataset is actually a text file (a series of words), for example, the first file looks like this:

```
From: lerxst@wam.umd.edu (where's my thing)
Subject: WHAT car is this!?
Nntp-Posting-Host: rac3.wam.umd.edu
Organization: University of Maryland, College Park
Lines: 15

 I was wondering if anyone out there could enlighten me on this car I saw
the other day. It was a 2-door sports car, looked to be from the late 60s/
early 70s. It was called a Bricklin. The doors were really small. In addition,
the front bumper was separate from the rest of the body. This is
all I know. If anyone can tellme a model name, engine specs, years
of production, where this car is made, history, or whatever info you
have on this funky looking car, please e-mail.

Thanks,
- IL
   ---- brought to you by your neighborhood Lerxst ----
```

Implementation And Working

(1) First, we will import the necessary libraries.

Code Part:

Importing Libraries

In [1]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation, Dense, Dropout , LSTM
from sklearn.preprocessing import LabelBinarizer
import sklearn.datasets as skds
from pathlib import Path
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

(2) The dataset comprises of training and testing data. We will first read the training dataset.

Code Part:

Reading Train Dataset Files

In [2]:
```
# For reproducibility
np.random.seed(1237)

# Source file directory
path_train = "C://Users//hp-pc//Documents//Celebal Work//20news-bydate-train"

files_train = skds.load_files(path_train,load_content=False)

label_index = files_train.target
label_names = files_train.target_names
labelled_files = files_train.filenames

data_tags = ["filename","category","news"]
data_list = []

# Read and add data from file to a list
i=0
for f in labelled_files:
    data_list.append((f,label_names[label_index[i]],Path(f).read_text(encoding='latin1')))
    i += 1

# We have training data available as dictionary filename, category, data
data = pd.DataFrame.from_records(data_list, columns=data_tags)
```

Creation of the initial dataset

The aim of this step is to get a dataset with the following structure:

Code Part:

```
In [3]: data
```

Out[3]:

	filename	category	news
0	C://Users//hp-pc//Documents//Celebal Work//20n...	rec.sport.baseball	From: cubbie@garnet.berkeley.edu (...
1	C://Users//hp-pc//Documents//Celebal Work//20n...	comp.sys.mac.hardware	From: gnelson@pion.rutgers.edu (Gregory Nelson...
2	C://Users//hp-pc//Documents//Celebal Work//20n...	sci.crypt	From: crypt-comments@math.ncsu.edu\nSubject: C...
3	C://Users//hp-pc//Documents//Celebal Work//20n...	comp.sys.mac.hardware	From: ()\nSubject: Re: Quadra SCSI Problems??...
4	C://Users//hp-pc//Documents//Celebal Work//20n...	alt.atheism	From: keith@cco.caltech.edu (Keith Allan Schne...
...	...	...	...
11309	C://Users//hp-pc//Documents//Celebal Work//20n...	rec.motorcycles	From: rbemben@timewarp.prime.com (Rich Bemben)...
11310	C://Users//hp-pc//Documents//Celebal Work//20n...	comp.windows.x	From: stevedav@netcom.com (Steve Davidson)\nSu...
11311	C://Users//hp-pc//Documents//Celebal Work//20n...	talk.politics.guns	From: 0005111312@mcimail.com (Peter Nesbitt)\n...
11312	C://Users//hp-pc//Documents//Celebal Work//20n...	talk.politics.misc	From: eck@panix.com (Mark Eckenwiler)\nSubject...
11313	C://Users//hp-pc//Documents//Celebal Work//20n...	comp.sys.mac.hardware	From: lau@aerospace.aero.org (David Lau)\nSubj...

11314 rows × 3 columns

Filename : File Path
Category : Label/Topic
News : Text for corresponding label

(3) Split 80% of data as train data and 20% as validation data.

Code Part:

Splitting 80% Train Data as training data and 20% for validation

```
In [4]: # lets take 80% data as training and remaining 20% for validation
        train_size = int(len(data) * .8)

        train_posts = data['news'][:train_size]
        train_tags = data['category'][:train_size]
        train_files_names = data['filename'][:train_size]

        validation_test_posts = data['news'][train_size:]
        validation_test_tags = data['category'][train_size:]
        validation_test_files_names = data['filename'][train_size:]
```

(4) In the pre-processing first, we need to tokenize the texts and then convert it into a TF – IDF matrix. Also, label binarize the train tags/y_train (labels/topics) , since our model will deal with numerical data.

Why is Tokenization required in NLP?

Before processing a natural language, we need to identify the words that constitute a string of characters. That's why tokenization is the most basic step to proceed with NLP (text data). This is important because the meaning of the text could easily be interpreted by analysing the words present in the text.

Tokenization is the foremost step while modelling text data. Tokenization is performed on the corpus to obtain tokens. The following tokens are then used to prepare a vocabulary. Vocabulary refers to the set of unique tokens in the corpus. Remember that vocabulary can be constructed by considering each unique token in the corpus or by considering the top K Frequently Occurring Words.

Code Part:

Tokenizing and Applying TF-IDF

In [5]:
```
# 20 news groups
num_labels = 20
vocab_size = 15000
batch_size = 100

# define Tokenizer with Vocab Size
tokenizer = Tokenizer(num_words=vocab_size)
tokenizer.fit_on_texts(train_posts)

x_train = tokenizer.texts_to_matrix(train_posts, mode='tfidf')
validation_x_test = tokenizer.texts_to_matrix(validation_test_posts, mode='tfidf')

encoder = LabelBinarizer()
encoder.fit(train_tags)
y_train = encoder.transform(train_tags)
validation_y_test = encoder.transform(validation_test_tags)
```

TF-IDF is a score that represents the relative importance of a term in the document and the entire corpus. TF stands for Term Frequency, and IDF stands for Inverse Document Frequency:

$$TFIDF(t,d) = TF(t,d) \times \log\left(\frac{N}{DF(t)}\right)$$

Being:

- t: term (i.e. a word in a document)
- d: document
- $TF(t)$: term frequency (i.e. how many times the term t appears in the document d)
- N: number of documents in the corpus
- $DF(t)$: number of documents in the corpus containing the term t

The TF-IDF value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general**.**

It also takes into account the fact that some documents may be larger than others by normalizing the TF term (expressing instead relative term frequencies).

Example:

Word	TF		IDF	TF*IDF	
	A	B		A	B
The	1/7	1/7	log(2/2) = 0	0	0
Car	1/7	0	log(2/1) = 0.3	0.043	0
Truck	0	1/7	log(2/1) = 0.3	0	0.043
Is	1/7	1/7	log(2/2) = 0	0	0
Driven	1/7	1/7	log(2/2) = 0	0	0
On	1/7	1/7	log(2/2) = 0	0	0
The	1/7	1/7	log(2/2) = 0	0	0
Road	1/7	0	log(2/1) = 0.3	0.043	0
Highway	0	1/7	log(2/1) = 0.3	0	0.043

(5) Since we will be using a LSTM layer in our model and it expects an input in 3D shape, so , we need to reshape our input as shown below.

Code Part:

x_train = np.reshape(x_train, (x_train.shape[0], 1, x_train.shape[1]))

validation_x_test = np.reshape(validation_x_test, (validation_x_test.shape[0], 1, validation_x_test.shape[1]))

```
In [9]: x_train = np.reshape(x_train, (x_train.shape[0], 1, x_train.shape[1]))
        validation_x_test = np.reshape(validation_x_test, (validation_x_test.shape[0], 1, validation_x_test.shape[1]))
```

(6) Now, we have to build the model for our TF-IDF input matrix.

-> Taking our matrix as input and defining the first layer which is an LSTM layer.

Code Part:
model.add(LSTM(512,input_shape=(1,vocab_size)))

-> Defining the 2nd layer of our model as activation layer using the relu function.

Code Part:
model.add(Activation('relu'))

-> Defining the 3rd layer as dropout to avoid overfitting of model.

Code Part:
model.add(Dropout(0.3))

-> Next is the dense layer.

Code Part:
model.add(Dense(512))

-> Now, follow the same steps from step 2nd for further layers.

Code Part:

```
model.add(Activation('relu'))
model.add(Dropout(0.3))
model.add(Dense(num_labels))
```

-> Final layer is the softmax layer (used for multi class classification problems).

Code Part:

```
model.add(Activation('softmax'))
```

Loss Function : categorical_crossentropy
It is a loss function which is used in multi class classification.

Optimizer : Adam Optimizer
We used this because it is efficient when working with large problems involving a lot of data and parameters. It requires less memory to function.

Metrics : Accuracy
It is used to find the accuracy of the model.

Model Building

In [10]:

```
model = Sequential()
model.add(LSTM(512,input_shape=(1,vocab_size)))
model.add(Activation('relu'))
model.add(Dropout(0.3))
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.3))
model.add(Dense(num_labels))
model.add(Activation('softmax'))
model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=10,
                    verbose=1,
                    validation_data=(validation_x_test, validation_y_test))
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 512)               31770624
_________________________________________________________________
activation (Activation)      (None, 512)               0
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 512)               262656
_________________________________________________________________
activation_1 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 20)                10260
_________________________________________________________________
activation_2 (Activation)    (None, 20)                0
=================================================================
Total params: 32,043,540
Trainable params: 32,043,540
Non-trainable params: 0
```

(7) After training the model we got following results.

Training Accuracy = 99.82% , Validation Accuracy = 92.0%

(8) Now, we can test our model in the test dataset. We will read the test dataset in the same way and apply TF-IDF tokenizer and reshape it for our model.

Code Part:

Reading Test Dataset Files

In [12]:
```
# Source file directory
path_test = "C://Users//hp-pc//Documents//Celebal Work//20news-bydate-test"

files_test = skds.load_files(path_test,load_content=False)

label_index = files_test.target
label_names = files_test.target_names
labelled_files = files_test.filenames

data_tags = ["filename","category","news"]
data_list = []

# Read and add data from file to a list
i=0
for f in labelled_files:
    data_list.append((f,label_names[label_index[i]],Path(f).read_text(encoding='latin1')))
    i += 1

# We have testing data available as dictionary filename, category, data
data = pd.DataFrame.from_records(data_list, columns=data_tags)
```

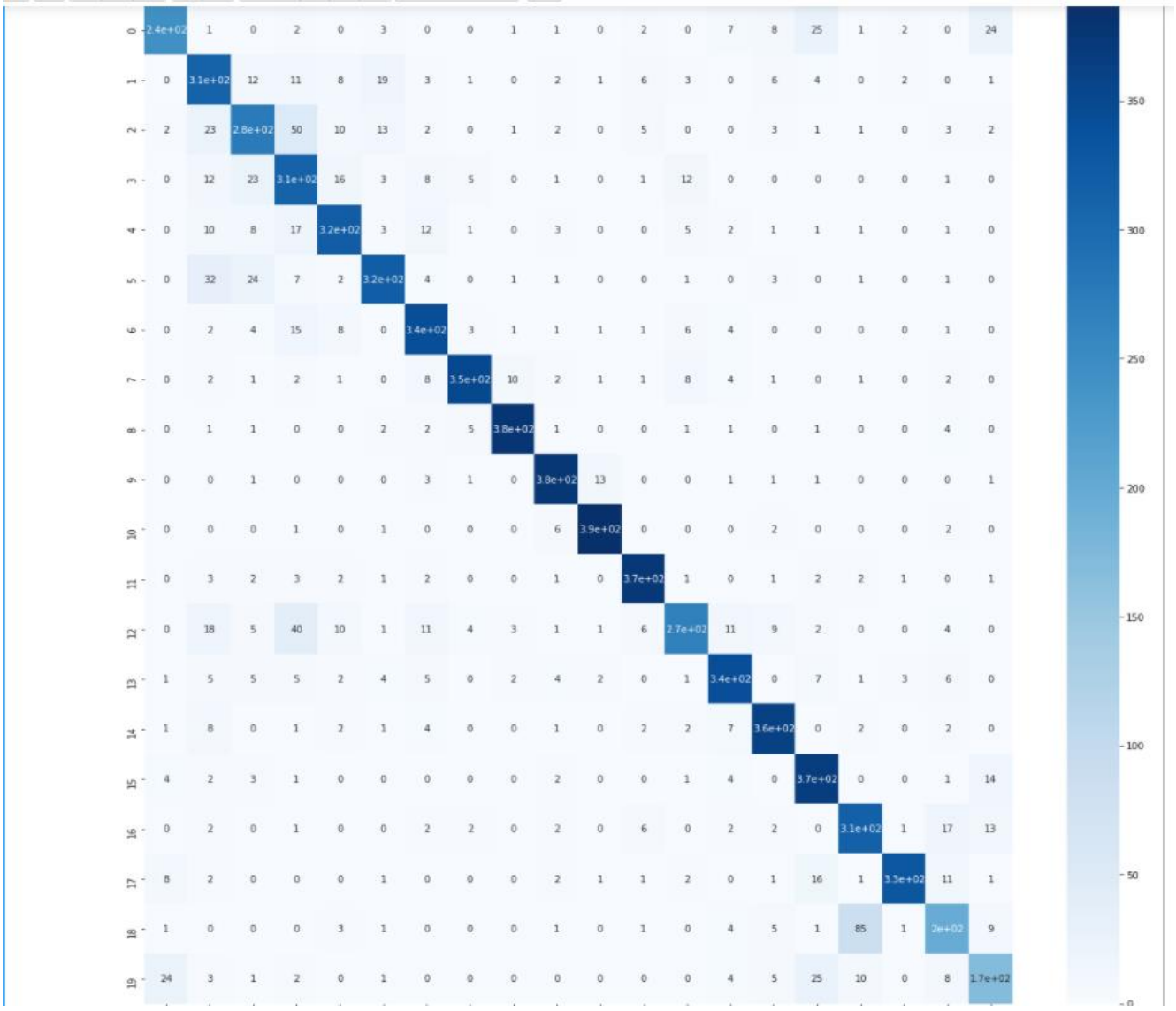
(8) After, running model on test dataset we get following results:

Test Accuracy = 84.06 %

(9) Now, plot the confusion matrix

Code Part:

```
cm = confusion_matrix(test_tags, labels[y_pred])
plt.figure(figsize=(20, 20))
sns.heatmap(cm, annot=True, cmap='Blues')
```



After evaluating the confusion matrix, the model performance was found good.

Why we use confusion matrix ?

The main problem with classification accuracy is that it hides the detail you need to better understand the performance of your classification model. There are two examples where you are most likely to encounter this problem:

1. When your data has more than 2 classes. With 3 or more classes you may get a classification accuracy of 80%, but you don't know if that is because all classes are being predicted equally well or whether one or two classes are being neglected by the model.
2. When your data does not have an even number of classes. You may achieve accuracy of 90% or more, but this is not a good score if 90 records for every 100 belong to one class and you can achieve this score by always predicting the most common class value.

The confusion matrix shows the ways in which your classification model is confused when it makes predictions.

It gives insight not only into the errors being made by our classifier but more importantly the types of errors that are being made.

It is this breakdown that overcomes the limitation of using classification accuracy alone.

Accuracy: the percentage of texts that were categorized with the correct tag.

Precision: the percentage of examples the classifier got right out of the total number of examples that it predicted for a given tag.

Recall: the percentage of examples the classifier predicted for a given tag out of the total number of examples it should have predicted for that given tag.

F1 Score: the harmonic mean of precision and recall.

(10) Now, we can further test our model by taking some random paragraphs from Wikipedia having labels/topics on which our model was trained.

Code Part:

Testing Model on random paragraphs from wikipedia with same topics/labels as used on training data

In [24]:
```
# source: https://en.wikipedia.org/wiki/History_of_Christianity
paragraph_1 = '''The history of Christianity concerns the Christian religion, Christian countries,\
    and the Church with its various denominations, from the 1st century to the present.\
    Christianity originated with the ministry of Jesus, a Jewish teacher and healer who \
    proclaimed the imminent kingdom of God and was crucified c. AD 30–33 in Jerusalem in \
    the Roman province of Judea. His followers believe that, according to the Gospels, he \
    was the Son of God and that he died for the forgiveness of sins and was raised from the\
    dead and exalted by God, and will return soon at the inception of God's kingdom.'''
# source: https://en.wikipedia.org/wiki/Graphics
paragraph_2 = '''Graphics (from Greek γραφικός graphikos, 'belonging to drawing') are\
               visual images or designs on some surface, such as a wall, canvas, screen,\
               paper, or stone to inform, illustrate, or entertain. In contemporary usage,\
               it includes a pictorial representation of data, as in c manufacture, in\
               typesetting and the graphic arts, and in educational and recreational software.\
               Images that are generated by a computer are called computer graphics.'''
x_data = []
x_data.append(paragraph_1)
x_data.append(paragraph_2)
actual_label = ['Christianity','Graphics']
x_data_series = pd.Series(x_data)
x_tokenized = tokenizer.texts_to_matrix(x_data_series, mode='tfidf')
x_tokenized = np.reshape(x_tokenized, (x_tokenized.shape[0], 1, x_tokenized.shape[1]))
i=0
from tabulate import tabulate
list1 =[]
for x_t in x_tokenized:
    prediction = model.predict(np.array([x_t]))
    predicted_label = labels[np.argmax(prediction[0])]
    list1.append([actual_label[i],predicted_label])
    i += 1
print(tabulate(list1, headers=['Actual Label', 'Predicted Label'],tablefmt='fancy_grid'))
```

Results of above testing:

Actual Label	Predicted Label
Christianity	soc.religion.christian
Graphics	comp.graphics

Code

Code:

Importing Libraries

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation, Dense, Dropout , LSTM
from sklearn.preprocessing import LabelBinarizer
import sklearn.datasets as skds
from pathlib import Path
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

Reading Train Dataset Files

```
# For reproducibility
np.random.seed(1237)

# Source file directory
path_train = "C://Users//hp-pc//Documents//Celebal Work//20news-bydate-train"

files_train = skds.load_files(path_train,load_content=False)

label_index = files_train.target
label_names = files_train.target_names
labelled_files = files_train.filenames

data_tags = ["filename","category","news"]
data_list = []

# Read and add data from file to a list
i=0
for f in labelled_files:
```

```
    data_list.append((f,label_names[label_index[i]],Path(f).read_text(encoding='latin1')))
    i += 1

# We have training data available as dictionary filename, category, data
data = pd.DataFrame.from_records(data_list, columns=data_tags)

data
```

Splitting 80% Train Data as training data and 20% for validation

```
# lets take 80% data as training and remaining 20% for validation
train_size = int(len(data) * .8)

train_posts = data['news'][:train_size]
train_tags = data['category'][:train_size]
train_files_names = data['filename'][:train_size]

validation_test_posts = data['news'][train_size:]
validation_test_tags = data['category'][train_size:]
validation_test_files_names = data['filename'][train_size:]
```

Tokenizing and Applying TF-IDF

```
# 20 news groups
num_labels = 20
vocab_size = 15000
batch_size = 100

# define Tokenizer with Vocab Size
tokenizer = Tokenizer(num_words=vocab_size)
tokenizer.fit_on_texts(train_posts)

x_train = tokenizer.texts_to_matrix(train_posts, mode='tfidf')
validation_x_test = tokenizer.texts_to_matrix(validation_test_posts, mode='tfidf')

encoder = LabelBinarizer()
encoder.fit(train_tags)
y_train = encoder.transform(train_tags)
validation_y_test = encoder.transform(validation_test_tags)

train_posts
x_train
y_train

x_train = np.reshape(x_train, (x_train.shape[0], 1, x_train.shape[1]))
validation_x_test = np.reshape(validation_x_test, (validation_x_test.shape[0], 1,
validation_x_test.shape[1]))
```

Model Building

```
model = Sequential()
model.add(LSTM(512,input_shape=(1,vocab_size)))
model.add(Activation('relu'))
model.add(Dropout(0.3))
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.3))
model.add(Dense(num_labels))
model.add(Activation('softmax'))
model.summary()

model.compile(loss='categorical_crossentropy',
          optimizer='adam',
          metrics=['accuracy'])

history = model.fit(x_train, y_train,
                batch_size=batch_size,
                epochs=10,
                verbose=1,
                validation_data=(validation_x_test, validation_y_test))
```

Training Accuracy = 99.82% , Validation Accuracy = 92.0%

```
encoder.classes_
```

Reading Test Dataset Files

```
# Source file directory
path_test = "C://Users//hp-pc//Documents//Celebal Work//20news-bydate-test"

files_test = skds.load_files(path_test,load_content=False)

label_index = files_test.target
label_names = files_test.target_names
labelled_files = files_test.filenames

data_tags = ["filename","category","news"]
data_list = []

# Read and add data from file to a list
i=0
for f in labelled_files:
   data_list.append((f,label_names[label_index[i]],Path(f).read_text(encoding='latin1')))
   i += 1

# We have testing data available as dictionary filename, category, data
data = pd.DataFrame.from_records(data_list, columns=data_tags)
```

```
data
test_posts = data['news'][:]
test_tags = data['category'][:]
test_files_names = data['filename'][:]
x_test = tokenizer.texts_to_matrix(test_posts, mode='tfidf')
x_test.shape
test_tags.shape
print(len(test_tags))
x_test = np.reshape(x_test, (x_test.shape[0], 1, x_test.shape[1]))
```

Topics/Labels

```
labels = np.array(['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc',
 'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware', 'comp.windows.x',
 'misc.forsale', 'rec.autos', 'rec.motorcycles', 'rec.sport.baseball',
 'rec.sport.hockey', 'sci.crypt', 'sci.electronics', 'sci.med', 'sci.space',
 'soc.religion.christian', 'talk.politics.guns', 'talk.politics.mideast',
 'talk.politics.misc', 'talk.religion.misc'])

prediction = model.predict(np.array(x_test))
y_pred = np.argmax(prediction, axis=1)
acc = (labels[y_pred]==test_tags).mean()
print("Test Accuracy:", acc)
```

Test Accuracy = 84.06 %

```
cm = confusion_matrix(test_tags, labels[y_pred])
```

Confusion Matrix

```
plt.figure(figsize=(20, 20))
sns.heatmap(cm, annot=True, cmap='Blues')
```

Testing Model on random paragraphs from wikipedia with same topics/labels as used on training data

```
# source: https://en.wikipedia.org/wiki/History_of_Christianity
paragraph_1 = '''The history of Christianity concerns the Christian religion, Christian countries,\
   and the Church with its various denominations, from the 1st century to the present.\
   Christianity originated with the ministry of Jesus, a Jewish teacher and healer who \
   proclaimed the imminent kingdom of God and was crucified c. AD 30–33 in Jerusalem in \
   the Roman province of Judea. His followers believe that, according to the Gospels, he \
   was the Son of God and that he died for the forgiveness of sins and was raised from the\
   dead and exalted by God, and will return soon at the inception of God's kingdom.'''
# source: https://en.wikipedia.org/wiki/Graphics
paragraph_2 = '''Graphics (from Greek γραφικός graphikos, 'belonging to drawing') are\
         visual images or designs on some surface, such as a wall, canvas, screen,\
```

```
        paper, or stone to inform, illustrate, or entertain. In contemporary usage,\
        it includes a pictorial representation of data, as in c manufacture, in\
        typesetting and the graphic arts, and in educational and recreational software.\
        Images that are generated by a computer are called computer graphics.'''
x_data = []
x_data.append(paragraph_1)
x_data.append(paragraph_2)
actual_label = ['Christianity','Graphics']
x_data_series = pd.Series(x_data)
x_tokenized = tokenizer.texts_to_matrix(x_data_series, mode='tfidf')
x_tokenized = np.reshape(x_tokenized, (x_tokenized.shape[0], 1, x_tokenized.shape[1]))
i=0
from tabulate import tabulate
list1 =[]
for x_t in x_tokenized:
   prediction = model.predict(np.array([x_t]))
   predicted_label = labels[np.argmax(prediction[0])]
   list1.append([actual_label[i],predicted_label])
   i += 1
print(tabulate(list1, headers=['Actual Label', 'Predicted Label'],tablefmt='fancy_grid'))
```

References

- https://medium.com/swlh/text-classification-using-tf-idf-7404e75565b8
- https://hcis-journal.springeropen.com/track/pdf/10.1186/s13673-019-0192-7.pdf
- https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
- https://ieeexplore.ieee.org/document/9139510
- https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/