Notebook Executions

Submission Details

FIT5212 Assignment 1

Student Name - Simran Singh Gulati

Student ID - 31125301

Part 1 : Text Classification

We have 3 binary classification tasks, to predict if an article is labelled as InfoTheory, CompVis or Ma
to any number of classes or even none of them. So basically we have to test each article for all the th

The text to be classified is stored in the column termed **Abstract** and the 3 classes belong to the colu

1. InfoTheory
2. CompVis
3. Math

These 3 tasks will be tested using 2 different algorithms, one based on statistical approach and one I
as:

1. Statistical Classifier - SVM

Considering the algorithms discussed in tutorials, I short listed SVM and Logistics Regression.

As per this blog on Medium, SVM is a better choice than the Logistic Regression for both un-structure
author claims SVM isn't as prone to over-fitting as is Logistic Regression. Lastly, as observed in the tu
couldn't predict very well when exposed to an imbalanced dataset (or skewed distribution).

Thus I decided to go with SVM for the statistical classification.

2. Recurrent Neural Network - Simple RNN

For the RNN based approach I decided to go with Simple RNN.

Additionally the 3 tasks have to be tested against 2 different preprocessing techniques. I am choosin
process the text :

1. WordNetLemmatizer from NLTK
2. Spacy (preprocessing pipeline)

As per this article, Spacy tends to be much faster than NLTK and also supports word vectors. Given tl
sentence (unlike NLTK), it would be interesting to see how these two pipelines compare with each otl

The nlp pipeline from Spacy is a rigorous approach involving POS tagging, dependency labelling as w
test the entire pipeline, the size of our dataset limits us from running the whole pipeline. Therefore I d
restrict it to tokenisation, POS tagging and lemmatisation.

Lastly the 3 tasks have to be tained on two separte datasets. Therefore we first train our model on fir
the entire dataset (approximately 54000 records)

Having briefed you about my plan, we proceed to the coding part wherein each configuration would b
and recall score along with a **precision-recall curve**.

As I am using Google Colab for this assignment, we mount the drive to avoid uploading entire datase

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Statiscal Approach

Importing Libraries

Here we import the python libraies that we use in this notebook. Broadly speaking we have used Num
modules from NLTK, and SkLearn.

Numpy was used for data transofrmation and Pandas for loading data and (selecting rows from the c
processing based on pre-defined functions. SkLEarn was used for model training and evalusation. La
the model performance

```
%matplotlib inline

from nltk.corpus import stopwords
from nltk import word_tokenize
from nltk.tokenize import wordpunct_tokenize
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import precision_score, recall_score, f1_score, average_precisio
from sklearn.svm import LinearSVC, SVC
import pandas as pd
import numpy as np

import spacy
# Initialize spacy 'en' model, keeping only tagger component needed for lemmatization
nlp = spacy.load('en',  disable=['parser', 'ner'])
```

```
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import plot_precision_recall_curve
import matplotlib.pyplot as plt
```

```
/usr/local/lib/python3.7/dist-packages/_pytest/mark/structures.py:426: Deprecatio
  @attr.s(cmp=False, hash=False)
/usr/local/lib/python3.7/dist-packages/jsonschema/compat.py:6: DeprecationWarnin
  from collections import MutableMapping, Sequence  # noqa
/usr/local/lib/python3.7/dist-packages/jsonschema/compat.py:6: DeprecationWarnin
  from collections import MutableMapping, Sequence  # noqa
/usr/local/lib/python3.7/dist-packages/catalogue.py:138: DeprecationWarning: Sele
  for entry_point in AVAILABLE_ENTRY_POINTS.get(self.entry_point_namespace, []):
/usr/local/lib/python3.7/dist-packages/catalogue.py:138: DeprecationWarning: Sele
  for entry_point in AVAILABLE_ENTRY_POINTS.get(self.entry_point_namespace, []):
/usr/local/lib/python3.7/dist-packages/catalogue.py:126: DeprecationWarning: Sele
  for entry_point in AVAILABLE_ENTRY_POINTS.get(self.entry_point_namespace, []):
/usr/local/lib/python3.7/dist-packages/catalogue.py:138: DeprecationWarning: Sele
  for entry_point in AVAILABLE_ENTRY_POINTS.get(self.entry_point_namespace, []):
```

Loading Data

We create 3 dataframes, two for training each - one with first 100 records (df_train_1) and the other w
And one for the test set which again has all the records as in the file.

```
import pandas as pd

# Load the dataset into a pandas dataframe.
df_train_2 = pd.read_csv("/content/drive/Shareddrives/fit5212-s1-2021-tutorials/A1/ax
df_train_1 = df_train_2[:1000]
df_test = pd.read_csv("/content/drive/Shareddrives/fit5212-s1-2021-tutorials/A1/axcs_

# Report the number of sentences.
print('Number of sentences in train-set 1: {:,}\n'.format(df_train_1.shape[0]))
print('Number of sentences in train-set 2: {:,}\n'.format(df_train_2.shape[0]))
print('Number of sentences in test-set: {:,}\n'.format(df_test.shape[0]))

# Report proportions
print('Proportion of 0 class in InfoTheory in train-set 2 : {:.4f}\n'.format(df_train
print('Proportion of 0 class in CompVis in train-set 2 : {:.4f}\n'.format(df_train_2.
print('Proportion of 0 class in Math in train-set 2 : {:.4f}\n'.format(df_train_2.loc
```

```
Number of sentences in train-set 1: 1,000

Number of sentences in train-set 2: 54,731

Number of sentences in test-set: 19,678

Proportion of 0 class in InfoTheory in train-set 2 : 0.8075

Proportion of 0 class in CompVis in train-set 2 : 0.9594
```

```
Proportion of 0 class in Math in train-set 2 : 0.6944
```

Data Extraction

Now, for each dataset (2 train sets and 1 test set) we extract the the columns:

1. Abstract
2. InfoTheory
3. CompVis
4. Math

Wherein the Abstract is used for Docs to be trained and other 3 as labels.

```
# extract Docs and Labels
trainDocs1 = df_train_1.Abstract.tolist()
trainInfoTheoryLabels1 = df_train_1.InfoTheory.tolist()
trainCompVisLabels1 = df_train_1.CompVis.tolist()
trainMathLabels1 = df_train_1.Math.tolist()

trainDocs2 = df_train_2.Abstract.tolist()
trainInfoTheoryLabels2 = df_train_2.InfoTheory.tolist()
trainCompVisLabels2 = df_train_2.CompVis.tolist()
trainMathLabels2 = df_train_2.Math.tolist()

testDocs = df_test.Abstract.tolist()
testInfoTheoryLabels = df_test.InfoTheory.tolist()
testCompVisLabels = df_test.CompVis.tolist()
testMathLabels = df_test.CompVis.tolist()
```

Text Preprocessing

Tokenisation & Lemmatisation

Here I define two LemmaTokenizer, one based on NLTK and the other on Spacy. The idea behind thes already been discussed in the first Markdown Cell. Please refer to that for detailed explanaion. Here v the input and returns the Lemmatised form.

```
class LemmaTokenizerWordnet(object):
    def __init__(self):
        self.wnl=WordNetLemmatizer()
    def __call__(self,doc):
        return [self.wnl.lemmatize(t) for t in word_tokenize(doc)]

class LemmaTokenizerSpacy(object):
```

```
    def __call__(self,doc):
        trydoc = nlp(doc)
        return [token.lemma_ for token in trydoc]
```

Vectorisation

Similarly we created two separate vectorizer, one for NLTK Lematizer and the other for Spacy Lemma
transforms the tokenised texts into vectors.

```
vectorizer1 = TfidfVectorizer(analyzer='word',input='content',
                             lowercase=True,
                             token_pattern='(?u)\\b\\w\\w+\\b',
                             min_df=3,
                             ngram_range=(1,2),
                             tokenizer=LemmaTokenizerWordnet())
```

```
vectorizer2 = TfidfVectorizer(analyzer='word',input='content',
                             lowercase=True,
                             token_pattern='(?u)\\b\\w\\w+\\b',
                             min_df=3,
                             ngram_range=(1,2),
                             tokenizer=LemmaTokenizerSpacy())
```

Model Selection

This code is just for model selection. As already discussed in the first Markdown Cell, we use the Line
for detailed discussion. Focussing on the code here, the variable 'clf' stores the model/classifier nam

```
# this variable stores the model name
clf = LinearSVC()
```

```
import nltk
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

Pipeline

Having defined all the preprocessing tasks, I combine them into a user-defined functions that can als
text-classification.

This function takes 4 input parameters:

1. Training Data (Docs)
2. Training Labels
3. Test Data
4. Test Labels
5. Choice of Vectoriser

Depending upon the inputs, we fit and transform the training data using the vectorizer and then same

Plus the labels are also transformed using numpy arrays.

Then the classifier (LinearSVC) selected here is used to train the model.

Once trained, this model is used to make predictions for the test set and the model performance is pr

As discussed above the metrics chosen are F1, Precision and Recall with a Precision-Recall curve.

```
def text_classification(trainDocs, trainLabels, testDocs, testLabels, vectorizer):
  x_train=vectorizer.fit_transform(trainDocs)
  y_train=np.asarray(trainLabels)
  # Use the same vectorizer to transform the test set
  x_test=vectorizer.transform(testDocs)
  y_test=np.asarray(testLabels)

  clf.fit(x_train, y_train)
  y_predict=clf.predict(x_test)

  recall=recall_score(y_test,y_predict,average='macro')
  precision=precision_score(y_test,y_predict,average='macro')
  f1score=f1_score(y_test,y_predict,average='macro')
  average_precision = average_precision_score(y_test, y_predict)

  print('Macro F1 score:'+ str(f1score))
  print('Macro Precision: '+ str(precision))
  print('Macro Recall: '+ str(recall))

  disp = plot_precision_recall_curve(clf, x_test, y_test)
  disp.ax_.set_title('2-class Precision-Recall curve: '
                     'AP={0:0.2f}'.format(average_precision))
```

Implementation

Now we test all the configurations using the pipeling defined above.

We proceed in the manner:

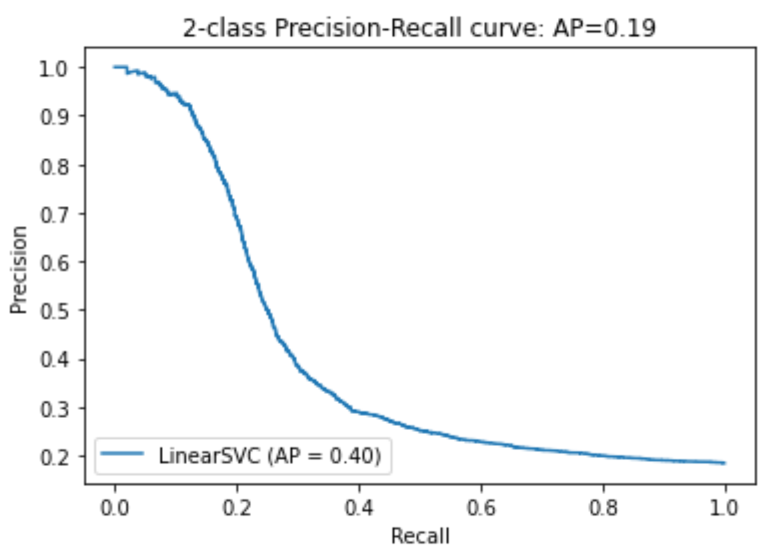
1. InfoTheory Classification
    1. Train Set 1, NLTK
    2. Train Set 1, Spacy
    3. Train set 2, NLTK
    4. Train set 2, Spacy
2. ComVis Classification
    1. Train Set 1, NLTK
    2. Train Set 1, Spacy
    3. Train set 2, NLTK
    4. Train set 2, Spacy
3. Math Classification
    1. Train Set 1, NLTK
    2. Train Set 1, Spacy
    3. Train set 2, NLTK
    4. Train set 2, Spacy

Thereby we begin with

1.1 InfoTheory Classification with training set 1 and NLTK preprocessing.

```
text_classification(trainDocs1, trainInfoTheoryLabels1, testDocs, testInfoTheoryLabel
```

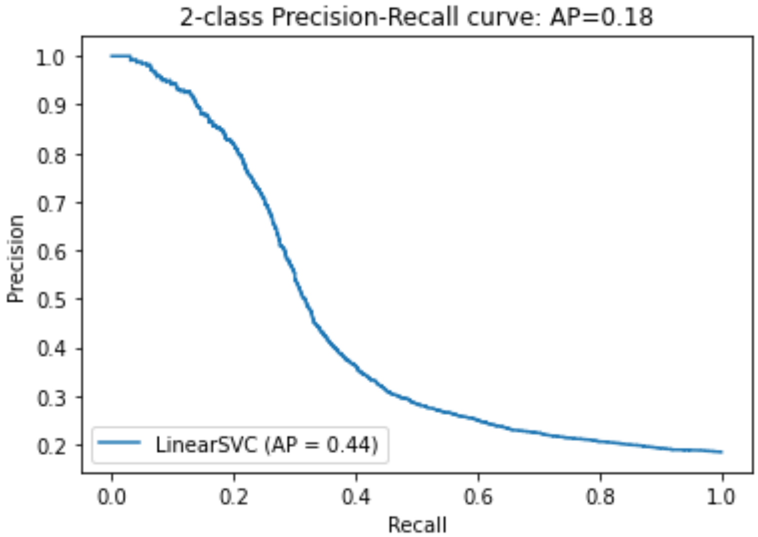
```
Macro F1 score:0.45373221302681077
Macro Precision: 0.908432080557392
Macro Recall: 0.5020741150442478
```



1.2 InfoTheory Classification with training set 1 and Spacy Preprocessing

```
text_classification(trainDocs1, trainInfoTheoryLabels1, testDocs, testInfoTheoryLabel
```

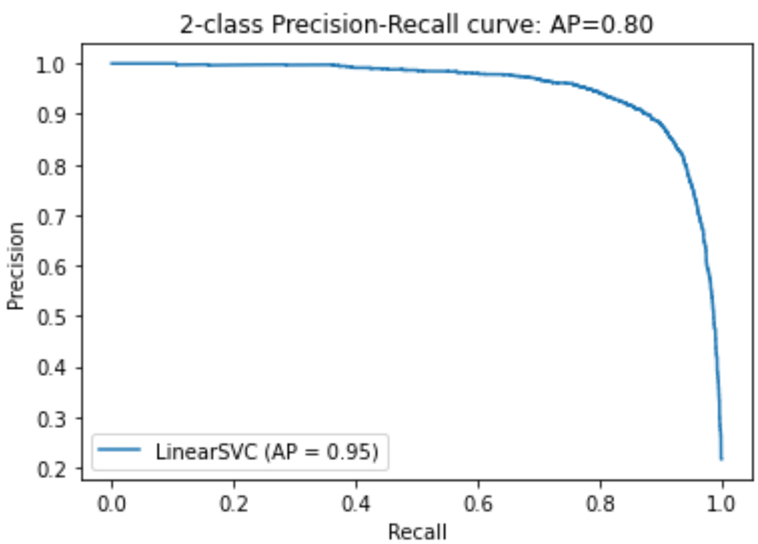
```
Macro F1 score:0.44941242305540013
Macro Precision: 0.40812074397804654
Macro Recall: 0.5
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: U
  _warn_prf(average, modifier, msg_start, len(result))
```



1.3 InfoTheory Classification with training set 2 and NLTK Preprocessing

```
text_classification(trainDocs2, trainInfoTheoryLabels2, testDocs, testInfoTheoryLabel
```

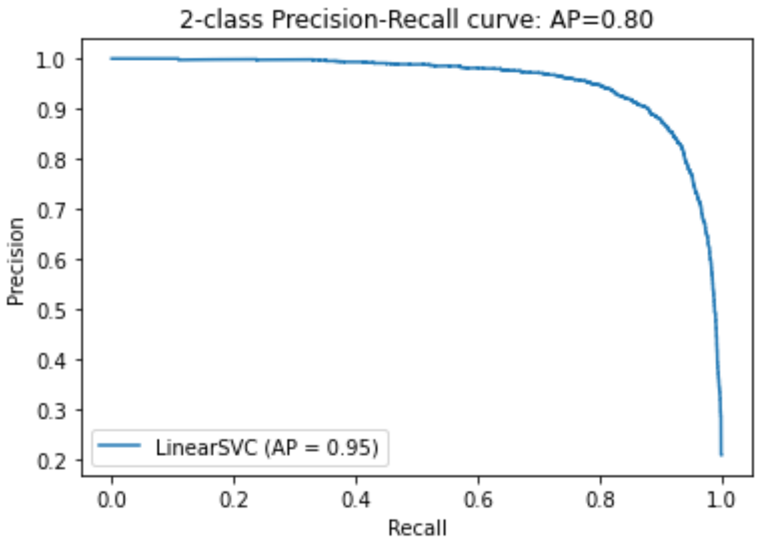
```
Macro F1 score:0.9253837329904628
Macro Precision: 0.9451900599705472
Macro Recall: 0.9083372520531612
```



1.4 InfoTheory Classification with training set 2 and Spacy Preprocessing

```
text_classification(trainDocs2, trainInfoTheoryLabels2, testDocs, testInfoTheoryLabel
```

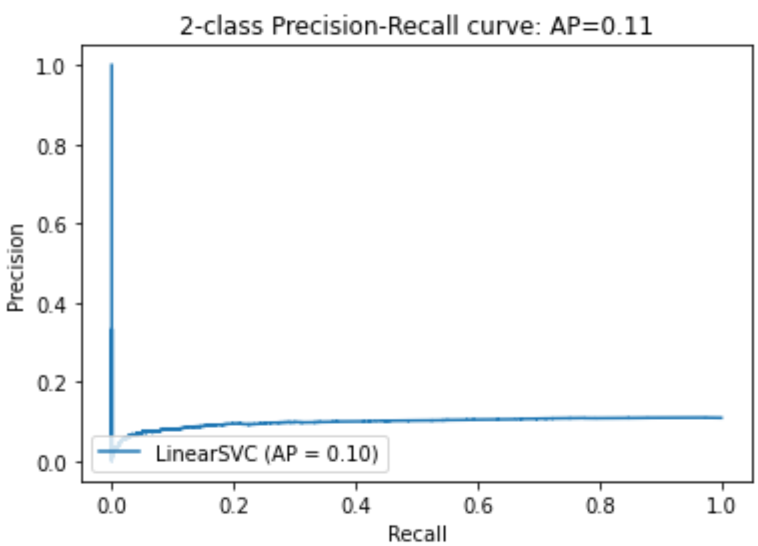
```
Macro F1 score:0.9249162947966643
Macro Precision: 0.9448837857618397
Macro Recall: 0.9077530253343515
```



2.1 CompVis Classification with training set 1 and NLTK Preprocessing

```
text_classification(trainDocs1, trainCompVisLabels1, testDocs, testCompVisLabels, vec
```

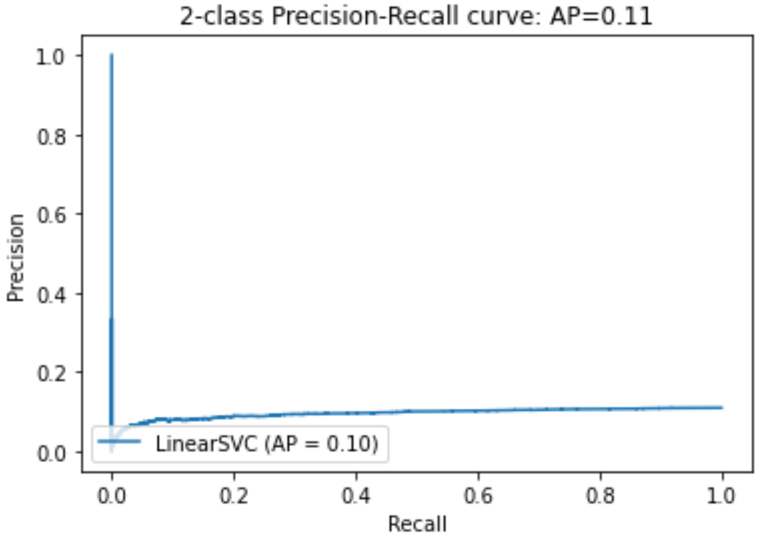
```
Macro F1 score:0.4710783786689603
Macro Precision: 0.44531964630551885
Macro Recall: 0.5
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: U
  _warn_prf(average, modifier, msg_start, len(result))
```



2.2 CompVis Classification with training set 1 and Spacy Preprocessing

```
text_classification(trainDocs1, trainCompVisLabels1, testDocs, testCompVisLabels, vec
```

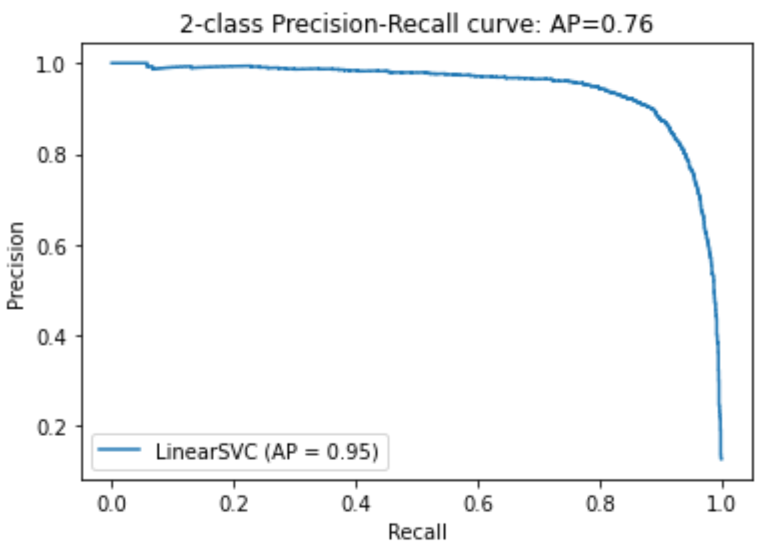
```
Macro F1 score:0.4710783786689603
Macro Precision: 0.44531964630551885
Macro Recall: 0.5
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: U
  _warn_prf(average, modifier, msg_start, len(result))
```



2.3 CompVis Classification with training set 2 and NLTK Preprocessing

```
text_classification(trainDocs2, trainCompVisLabels2, testDocs, testCompVisLabels, vec
```

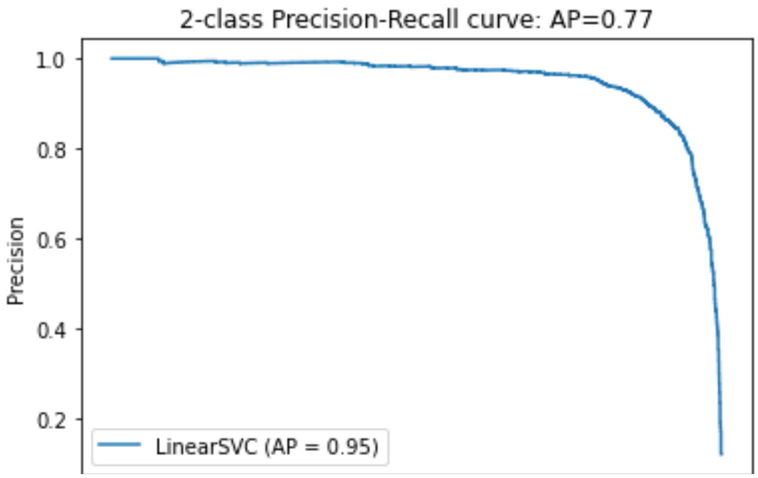
```
Macro F1 score:0.917978604149698
Macro Precision: 0.9641612700015845
Macro Recall: 0.8821254730624326
```



2.4 CompVis Classification with training set 2 and Spacy Preprocessing

```
text_classification(trainDocs2, trainCompVisLabels2, testDocs, testCompVisLabels, vec
```

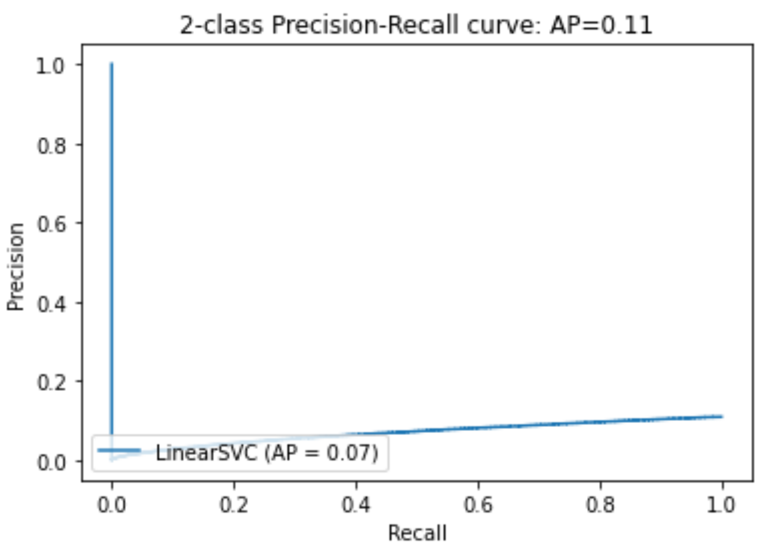
```
Macro F1 score:0.9217483642082678
Macro Precision: 0.9668036385053989
Macro Recall: 0.8865073324942189
```



3.1 Math Classification with training set 1 and NLTK Preprocessing

```
text_classification(trainDocs1, trainMathLabels1, testDocs, testMathLabels, vectorize
```

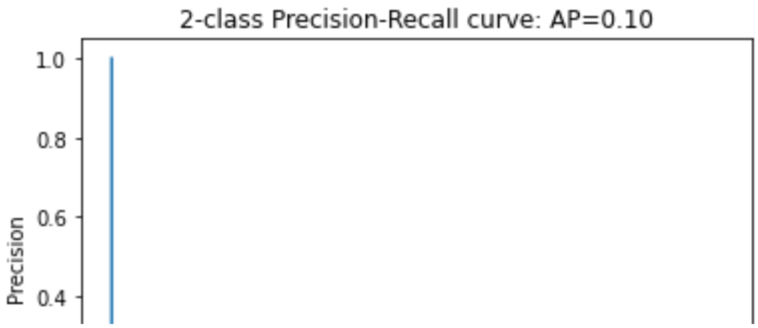
```
Macro F1 score:0.46770179614801993
Macro Precision: 0.4446558995988067
Macro Recall: 0.49326714595458176
```



3.2 Math Classification with training set 1 and Spacy Preprocessing

```
text_classification(trainDocs1, trainMathLabels1, testDocs, testMathLabels, vectorize
```

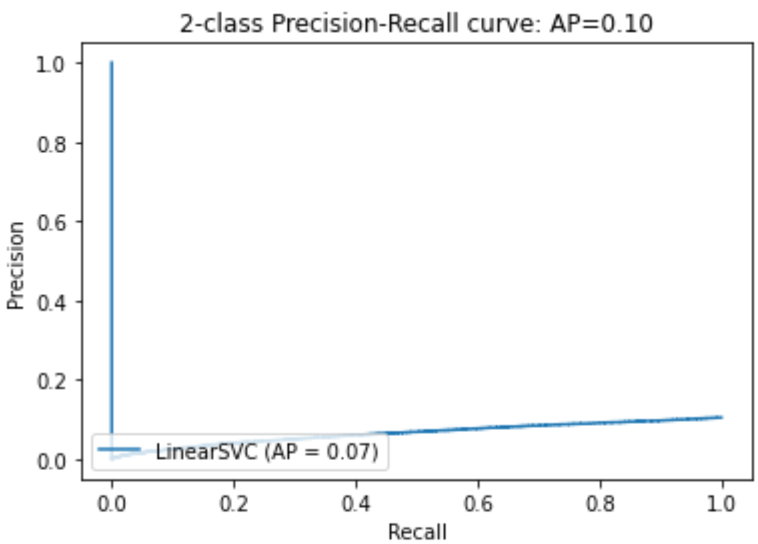
```
Macro F1 score:0.47011445527766005
Macro Precision: 0.44717741935483873
Macro Recall: 0.49553172475424484
```



3.3 Math Classification with training set 2 and NLTK Preprocessing

```
text_classification(trainDocs2, trainMathLabels2, testDocs, testMathLabels, vectorize
```

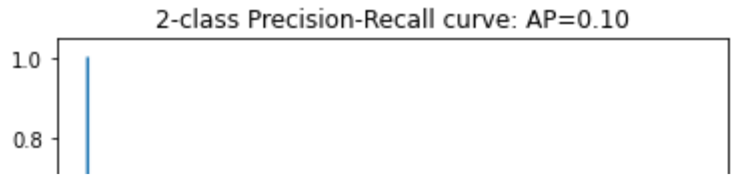
```
Macro F1 score:0.3974580942036528
Macro Precision: 0.4373174079613202
Macro Recall: 0.3695502391038891
```



3.4 Math Classification with training set 2 and Spacy Preprocessing

```
text_classification(trainDocs2, trainMathLabels2, testDocs, testMathLabels, vectorize
```

```
Macro F1 score:0.39886294677979656
Macro Precision: 0.43836588361917445
Macro Recall: 0.3717937566938856
```



RNN Approach

Now we create a Simple RNN to repeat the same process for InfoTheory, CompVis and Math Classific

Importing Libraries

We load relevent modules for RNN model.

```
import torch
from torchtext.legacy import data
from torchtext.legacy.data import TabularDataset
```

Loading Data

We begin by definig how the data should be processed. We will be using `TEXT` field for Abstract and

```
SEED = 1234

torch.manual_seed(SEED)
torch.backends.cudnn.deterministic = True

TEXT = data.Field(sequential=True, tokenize = 'spacy', lower=True)
tokenize = lambda x: x.split()
TEXT = data.Field(sequential=True, tokenize = tokenize, lower=True) #in case you want

LABEL = data.LabelField(dtype = torch.float, use_vocab=False, preprocessing=int)
```

Using the TabularDataset we read our data in csv format. The two files are loaded as train and test se

```
tv_datafields = [("ID", None),
                 ("URL", None),
                 ("Date",None),
                 ("Title", None),
                 ("InfoTheory", LABEL),
                 ("CompVis", LABEL),
                 ("Math", LABEL),
                 ("Abstract", TEXT),]
```

```
train_data, test_data = TabularDataset.splits(
#   path='cola_public/for_torch_text', train='in_domain_train.tsv',
    path='/content/drive/Shareddrives/fit5212-s1-2021-tutorials/A1', train='axcs_trai
    fields=tv_datafields)
```

Now we split the training data into train and validation sets.

```
train_data, valid_data = train_data.split(split_ratio=0.8)
```

Vocabulary and Interator

We pick the most common 5400 words from our data and create a look up table for our model. Lastly
trainig and evaluation.

```
MAX_VOCAB_SIZE = 5400

TEXT.build_vocab(train_data, max_size = MAX_VOCAB_SIZE)
#LABEL.build_vocab(train_data)

BATCH_SIZE = 15

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

train_iterator, valid_iterator, test_iterator = data.BucketIterator.splits(
    (train_data, valid_data, test_data),
    batch_size = BATCH_SIZE,
    device = device,
    sort_key = lambda x: len(x.Abstract),
    sort_within_batch = False)

batch = next(train_iterator.__iter__())
batch
```

```
[torchtext.legacy.data.batch.Batch of size 15]
        [.InfoTheory]:[torch.cuda.FloatTensor of size 15 (GPU 0)]
        [.CompVis]:[torch.cuda.FloatTensor of size 15 (GPU 0)]
        [.Math]:[torch.cuda.FloatTensor of size 15 (GPU 0)]
        [.Abstract]:[torch.cuda.LongTensor of size 306x15 (GPU 0)]
```

Building Model

Creating a model with 3 layers - Embedding, RNN and Linear.

```
import torch.nn as nn
```

```
class RNN(nn.Module):
    def __init__(self, input_dim, embedding_dim, hidden_dim, output_dim):

        super().__init__()

        self.embedding = nn.Embedding(input_dim, embedding_dim)

        self.rnn = nn.RNN(embedding_dim, hidden_dim)

        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, text):

        #text = [sent len, batch size]

        embedded = self.embedding(text)

        #embedded = [sent len, batch size, emb dim]

        output, hidden = self.rnn(embedded)

        #output = [sent len, batch size, hid dim]
        #hidden = [1, batch size, hid dim]

        assert torch.equal(output[-1,:,:], hidden.squeeze(0))

        return self.fc(hidden.squeeze(0))
```

In the next cell we set dimensions for each of the layers.

```
INPUT_DIM = len(TEXT.vocab)
EMBEDDING_DIM = 100
HIDDEN_DIM = 256
OUTPUT_DIM = 1

model = RNN(INPUT_DIM, EMBEDDING_DIM, HIDDEN_DIM, OUTPUT_DIM)
```

A fucntion to tell us the number of parameters in the model.

```
def count_parameters(model):
    return sum(p.numel() for p in model.parameters() if p.requires_grad)

print(f'The model has {count_parameters(model):,} trainable parameters')
```

```
The model has 632,105 trainable parameters
```

Training the Model

We first create an optimizer and then a loss function. And we see if the model can be placed on GPU

```
import torch.optim as optim

optimizer = optim.SGD(model.parameters(), lr=1e-3)
# loss function
criterion = nn.BCEWithLogitsLoss()
#
model = model.to(device)
criterion = criterion.to(device)
```

Create a function compute accuracy.

```
def binary_accuracy(preds, y):
    """
    Returns accuracy per batch, i.e. if you get 8/10 right, this returns 0.8, NOT 8
    """

    #round predictions to the closest integer
    rounded_preds = torch.round(torch.sigmoid(preds))
    correct = (rounded_preds == y).float() #convert into float for division
    acc = correct.sum() / len(correct)
    return acc
```

InfoTheory Classification

We first train the model for classfying if the Abstract falls in InfoTheory Label or not.

Train function that iterates over all examples one batch at a time.

```
def train(model, iterator, optimizer, criterion):

    epoch_loss = 0
    epoch_acc = 0

    model.train()

    for batch in iterator:

        optimizer.zero_grad()

        predictions = model(batch.Abstract).squeeze(1)

        loss = criterion(predictions, batch.InfoTheory)

        acc = binary_accuracy(predictions, batch.InfoTheory)
```

```
        acc = binary_accuracy(predictions, batch.InfoTheory)

        loss.backward()

        optimizer.step()

        epoch_loss += loss.item()
        epoch_acc += acc.item()

    return epoch_loss / len(iterator), epoch_acc / len(iterator)
```

Evaluate function is similar to rain except it does not update the parameters.

```
def evaluate(model, iterator, criterion):

    epoch_loss = 0
    epoch_acc = 0

    model.eval()

    with torch.no_grad():

        for batch in iterator:

            predictions = model(batch.Abstract).squeeze(1)

            loss = criterion(predictions, batch.InfoTheory)

            acc = binary_accuracy(predictions, batch.InfoTheory)

            epoch_loss += loss.item()
            epoch_acc += acc.item()

    return epoch_loss / len(iterator), epoch_acc / len(iterator)
```

A function to tell us how long each epoch takes

```
import time

def epoch_time(start_time, end_time):
    elapsed_time = end_time - start_time
    elapsed_mins = int(elapsed_time / 60)
    elapsed_secs = int(elapsed_time - (elapsed_mins * 60))
    return elapsed_mins, elapsed_secs
```

Training model for multiple epochs.

```
N_EPOCHS = 5
```

```
N_EPOCHS = 5

best_valid_loss = float('inf')

for epoch in range(N_EPOCHS):

    start_time = time.time()

    train_loss, train_acc = train(model, train_iterator, optimizer, criterion)
    valid_loss, valid_acc = evaluate(model, valid_iterator, criterion)

    end_time = time.time()

    epoch_mins, epoch_secs = epoch_time(start_time, end_time)

    if valid_loss < best_valid_loss:
        best_valid_loss = valid_loss
        torch.save(model.state_dict(), 'RNN_model.pt')

    print(f'Epoch: {epoch+1:02} | Epoch Time: {epoch_mins}m {epoch_secs}s')
    print(f'\tTrain Loss: {train_loss:.3f} | Train Acc: {train_acc*100:.2f}%')
    print(f'\t Val. Loss: {valid_loss:.3f} |  Val. Acc: {valid_acc*100:.2f}%')
```

```
Epoch: 01 | Epoch Time: 0m 32s
	Train Loss: 0.498 | Train Acc: 80.18%
	 Val. Loss: 0.554 |  Val. Acc: 76.80%
Epoch: 02 | Epoch Time: 0m 32s
	Train Loss: 0.493 | Train Acc: 80.47%
	 Val. Loss: 0.531 |  Val. Acc: 78.72%
Epoch: 03 | Epoch Time: 0m 32s
	Train Loss: 0.492 | Train Acc: 80.54%
	 Val. Loss: 0.521 |  Val. Acc: 79.62%
Epoch: 04 | Epoch Time: 0m 32s
	Train Loss: 0.492 | Train Acc: 80.60%
	 Val. Loss: 0.514 |  Val. Acc: 80.04%
Epoch: 05 | Epoch Time: 0m 32s
	Train Loss: 0.492 | Train Acc: 80.60%
	 Val. Loss: 0.508 |  Val. Acc: 80.27%
```

```
model.load_state_dict(torch.load('RNN_model.pt'))

test_loss, test_acc = evaluate(model, test_iterator, criterion)
print(f'Test Loss: {test_loss:.3f} | Test Acc: {test_acc*100:.2f}%')
```

```
Test Loss: 0.509 | Test Acc: 81.08%
```

```
y_predict = []
y_test = []

model.eval()
with torch.no_grad():
    for batch in test_iterator:
        predictions = model(batch.Abstract).squeeze(1)
```

```
        predictions = model(batch.Abstract).squeeze(1)
        rounded_preds = torch.round(torch.sigmoid(predictions))
        y_predict += rounded_preds.tolist()
        y_test += batch.InfoTheory.tolist()
        #acc = binary_accuracy(predictions, batch.label)
```

```
# from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
# import numpy as np

y_predict = np.asarray(y_predict)
y_test = np.asarray(y_test)

recall=recall_score(y_test,y_predict,average='macro')
precision=precision_score(y_test,y_predict,average='macro')
f1score=f1_score(y_test,y_predict,average='macro')
accuracy=accuracy_score(y_test,y_predict)
average_precision = average_precision_score(y_test, y_predict)

print('Macro Precision: '+ str(precision))
print('Macro Recall: '+ str(recall))
print('Macro F1 score:'+ str(f1score))
print('Accuracy: '+ str(accuracy))



# disp = plot_precision_recall_curve(model, test_data, y_test)
# disp.ax_.set_title('2-class Precision-Recall curve: '
                  #  'AP={0:0.2f}'.format(average_precision))
```

```
Macro Precision: 0.553917480084759
Macro Recall: 0.5045978842494184
Macro F1 score:0.466657721960156
Accuracy: 0.8108039434901921
```

CompVis Classification

We train the model for classfying if the Abstract falls in CompVis Label or not.

```
def train(model, iterator, optimizer, criterion):

    epoch_loss = 0
    epoch_acc = 0

    model.train()

    for batch in iterator:

        optimizer.zero_grad()

        predictions = model(batch.Abstract).squeeze(1)
```

```
        loss = criterion(predictions, batch.CompVis)

        acc = binary_accuracy(predictions, batch.CompVis)

        loss.backward()

        optimizer.step()

        epoch_loss += loss.item()
        epoch_acc += acc.item()

    return epoch_loss / len(iterator), epoch_acc / len(iterator)
```

```
def evaluate(model, iterator, criterion):

    epoch_loss = 0
    epoch_acc = 0

    model.eval()

    with torch.no_grad():

        for batch in iterator:

            predictions = model(batch.Abstract).squeeze(1)

            loss = criterion(predictions, batch.CompVis)

            acc = binary_accuracy(predictions, batch.CompVis)

            epoch_loss += loss.item()
            epoch_acc += acc.item()

    return epoch_loss / len(iterator), epoch_acc / len(iterator)
```

```
N_EPOCHS = 5

best_valid_loss = float('inf')

for epoch in range(N_EPOCHS):

    start_time = time.time()

    train_loss, train_acc = train(model, train_iterator, optimizer, criterion)
    valid_loss, valid_acc = evaluate(model, valid_iterator, criterion)

    end_time = time.time()

    epoch_mins, epoch_secs = epoch_time(start_time, end_time)
```

```
    if valid_loss < best_valid_loss:
        best_valid_loss = valid_loss
        torch.save(model.state_dict(), 'RNN_model.pt')

    print(f'Epoch: {epoch+1:02} | Epoch Time: {epoch_mins}m {epoch_secs}s')
    print(f'\tTrain Loss: {train_loss:.3f} | Train Acc: {train_acc*100:.2f}%')
    print(f'\t Val. Loss: {valid_loss:.3f} |  Val. Acc: {valid_acc*100:.2f}%')
```

```
Epoch: 01 | Epoch Time: 0m 32s
	Train Loss: 0.200 | Train Acc: 95.24%
	 Val. Loss: 0.360 |  Val. Acc: 92.85%
Epoch: 02 | Epoch Time: 0m 32s
	Train Loss: 0.183 | Train Acc: 95.74%
	 Val. Loss: 0.301 |  Val. Acc: 94.87%
Epoch: 03 | Epoch Time: 0m 32s
	Train Loss: 0.180 | Train Acc: 95.85%
	 Val. Loss: 0.265 |  Val. Acc: 95.67%
Epoch: 04 | Epoch Time: 0m 32s
	Train Loss: 0.178 | Train Acc: 95.88%
	 Val. Loss: 0.236 |  Val. Acc: 95.91%
Epoch: 05 | Epoch Time: 0m 32s
	Train Loss: 0.176 | Train Acc: 95.90%
	 Val. Loss: 0.218 |  Val. Acc: 96.00%
```

```
model.load_state_dict(torch.load('RNN_model.pt'))

test_loss, test_acc = evaluate(model, test_iterator, criterion)
print(f'Test Loss: {test_loss:.3f} | Test Acc: {test_acc*100:.2f}%')
```

```
Test Loss: 0.385 | Test Acc: 89.02%
```

```
y_predict = []
y_test = []

model.eval()
with torch.no_grad():
    for batch in test_iterator:
        predictions = model(batch.Abstract).squeeze(1)
        rounded_preds = torch.round(torch.sigmoid(predictions))
        y_predict += rounded_preds.tolist()
        y_test += batch.InfoTheory.tolist()
        #acc = binary_accuracy(predictions, batch.label)
```

Let's see how the model predicted articles for CompVis Class

```
y_predict = np.asarray(y_predict)
y_test = np.asarray(y_test)

recall=recall_score(y_test,y_predict,average='macro')
precision=precision_score(y_test,y_predict,average='macro')
```

```
f1score=f1_score(y_test,y_predict,average='macro')
accuracy=accuracy_score(y_test,y_predict)
average_precision = average_precision_score(y_test, y_predict)

print('Macro Precision: '+ str(precision))
print('Macro Recall: '+ str(recall))
print('Macro F1 score:'+ str(f1score))
print('Accuracy: '+ str(accuracy))
```

```
Macro Precision: 0.5331342145399085
Macro Recall: 0.5000897724305043
Macro F1 score:0.44989700119778114
Accuracy: 0.816038215265779
```

Math Classification

Lastly we train the model to see if the Abstract falls in Math class or not.

```
def train(model, iterator, optimizer, criterion):

    epoch_loss = 0
    epoch_acc = 0

    model.train()

    for batch in iterator:

        optimizer.zero_grad()

        predictions = model(batch.Abstract).squeeze(1)

        loss = criterion(predictions, batch.Math)

        acc = binary_accuracy(predictions, batch.Math)

        loss.backward()

        optimizer.step()

        epoch_loss += loss.item()
        epoch_acc += acc.item()

    return epoch_loss / len(iterator), epoch_acc / len(iterator)
```

```
def evaluate(model, iterator, criterion):

    epoch_loss = 0
    epoch_acc = 0
```

```
    model.eval()

    with torch.no_grad():

        for batch in iterator:

            predictions = model(batch.Abstract).squeeze(1)

            loss = criterion(predictions, batch.Math)

            acc = binary_accuracy(predictions, batch.Math)

            epoch_loss += loss.item()
            epoch_acc += acc.item()

    return epoch_loss / len(iterator), epoch_acc / len(iterator)
```

```
N_EPOCHS = 5

best_valid_loss = float('inf')

for epoch in range(N_EPOCHS):

    start_time = time.time()

    train_loss, train_acc = train(model, train_iterator, optimizer, criterion)
    valid_loss, valid_acc = evaluate(model, valid_iterator, criterion)

    end_time = time.time()

    epoch_mins, epoch_secs = epoch_time(start_time, end_time)

    if valid_loss < best_valid_loss:
        best_valid_loss = valid_loss
        torch.save(model.state_dict(), 'RNN_model.pt')

    print(f'Epoch: {epoch+1:02} | Epoch Time: {epoch_mins}m {epoch_secs}s')
    print(f'\tTrain Loss: {train_loss:.3f} | Train Acc: {train_acc*100:.2f}%')
    print(f'\t Val. Loss: {valid_loss:.3f} |  Val. Acc: {valid_acc*100:.2f}%')
```

```
Epoch: 01 | Epoch Time: 0m 32s
	Train Loss: 0.623 | Train Acc: 69.62%
	 Val. Loss: 0.661 |  Val. Acc: 68.68%
Epoch: 02 | Epoch Time: 0m 32s
	Train Loss: 0.616 | Train Acc: 69.62%
	 Val. Loss: 0.632 |  Val. Acc: 68.71%
Epoch: 03 | Epoch Time: 0m 32s
	Train Loss: 0.615 | Train Acc: 69.62%
	 Val. Loss: 0.626 |  Val. Acc: 68.71%
Epoch: 04 | Epoch Time: 0m 32s
	Train Loss: 0.614 | Train Acc: 69.62%
	 Val. Loss: 0.625 |  Val. Acc: 68.71%
```

```
Epoch: 05 | Epoch Time: 0m 32s
	Train Loss: 0.614 | Train Acc: 69.62%
	 Val. Loss: 0.624 |  Val. Acc: 68.71%
```

```
model.load_state_dict(torch.load('RNN_model.pt'))

test_loss, test_acc = evaluate(model, test_iterator, criterion)
print(f'Test Loss: {test_loss:.3f} | Test Acc: {test_acc*100:.2f}%')
```

```
Test Loss: 0.614 | Test Acc: 69.87%
```

```
y_predict = []
y_test = []

model.eval()
with torch.no_grad():
    for batch in test_iterator:
        predictions = model(batch.Abstract).squeeze(1)
        rounded_preds = torch.round(torch.sigmoid(predictions))
        y_predict += rounded_preds.tolist()
        y_test += batch.InfoTheory.tolist()
        #acc = binary_accuracy(predictions, batch.label)
```

Let's see how the model performs for Math class.

```
y_predict = np.asarray(y_predict)
y_test = np.asarray(y_test)

recall=recall_score(y_test,y_predict,average='macro')
precision=precision_score(y_test,y_predict,average='macro')
f1score=f1_score(y_test,y_predict,average='macro')
accuracy=accuracy_score(y_test,y_predict)
average_precision = average_precision_score(y_test, y_predict)

print('Macro Precision: '+ str(precision))
print('Macro Recall: '+ str(recall))
print('Macro F1 score:'+ str(f1score))
print('Accuracy: '+ str(accuracy))
```

```
Macro Precision: 0.5581502948952614
Macro Recall: 0.5001969173931106
Macro F1 score:0.4501696473628823
Accuracy: 0.816038215265779
```

Part 2 : Topic Modelling

In this section we use the trainig data from Part 1 to train an LDA model and create Vidualisations fro

Just like in task one we will train the model on two training sets using two separate preprocessing tec

Thereby the two training tests include first 100 and first 20,000 records respectively.

Now the the preprocessing variations that I am going to implement are:

1. With bigrams and trigrams:

Tokenisation —> Remove stop-words, numbers and single characters —> Add bigrams and trigrams – Bag of Words representation

2. Without Bi-Grams

Tokenisation —> Remove stop-words, numbers and single characters —> Remove rare and common t

Considering we have to train this LDA on two different data-sets, I am expecting significantly better re the bigger dataset.

Importing Libraries

Though most of the libarraies have already been loaded, we import the relevant gensim modules for t

```
import logging
from nltk.tokenize import RegexpTokenizer
from nltk.stem.wordnet import WordNetLemmatizer
from gensim.models import Phrases
from gensim.corpora import Dictionary
from gensim.models import LdaModel
!pip install pyLDAvis==2.1.2
import pyLDAvis.gensim
```

```
Collecting pyLDAvis==2.1.2
  Downloading https://files.pythonhosted.org/packages/a5/3a/af82e070a8a96e13217c
     |████████████████████████████████| 1.6MB 7.3MB/s
Requirement already satisfied: wheel>=0.23.0 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: numpy>=1.9.2 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: scipy>=0.18.0 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: pandas>=0.17.0 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: joblib>=0.8.4 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: jinja2>=2.7.2 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: numexpr in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pytest in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: future in /usr/local/lib/python3.7/dist-packages
Collecting funcy
  Downloading https://files.pythonhosted.org/packages/66/89/479de0afbbfb98d1c4b8
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: attrs>=17.4.0 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: py>=1.5.0 in /usr/local/lib/python3.7/dist-package
```

```
Requirement already satisfied: pluggy<0.8,>=0.5 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: more-itertools>=4.0.0 in /usr/local/lib/python3.7/
Requirement already satisfied: atomicwrites>=1.0 in /usr/local/lib/python3.7/dist
Building wheels for collected packages: pyLDAvis
  Building wheel for pyLDAvis (setup.py) ... done
  Created wheel for pyLDAvis: filename=pyLDAvis-2.1.2-py2.py3-none-any.whl size=
  Stored in directory: /root/.cache/pip/wheels/98/71/24/513a99e58bb6b8465bae4d2d
Successfully built pyLDAvis
Installing collected packages: funcy, pyLDAvis
Successfully installed funcy-1.15 pyLDAvis-2.1.2
/usr/local/lib/python3.7/dist-packages/past/types/oldstr.py:5: DeprecationWarning
  from collections import Iterable
```

Just like in Part 1 (SVM approach), I'll now create functions for preprocessing, modelling anf visualis configurations.

Pre-processing

In this function, I pass two parameters:

1. Training set
2. Bigram Flag

As discussed above, the preprocessing variations are based on inclusion and exclusion of bigrams fr Bigram Flag is Boolean parameter that represents with bigrams when set to True and without bigram have already been written in the last Markdown cell, I'll skip this part here. Finally the function returns

```
def pre_process(docs, bigram_flag):
  # Tokenize the documents.
  docs = docs

  # Split the documents into tokens.
  tokenizer = RegexpTokenizer(r'\w+')
  for idx in range(len(docs)):
      docs[idx] = docs[idx].lower()  # Convert to lowercase.
      docs[idx] = tokenizer.tokenize(docs[idx])  # Split into words.

  # Remove numbers, but not words that contain numbers.
  docs = [[token for token in doc if not token.isnumeric()] for doc in docs]

  # Remove words that are only one character.
  docs = [[token for token in doc if len(token) > 1] for doc in docs]

  lemmatizer = WordNetLemmatizer()
  docs = [[lemmatizer.lemmatize(token) for token in doc] for doc in docs]

  if (bigram_flag):
    print("Bigrams Added")
    bigram = Phrases(docs, min_count=20)
    for idx in range(len(docs)):
```

```
        for token in bigram[docs[idx]]:
            if '_' in token:
                # Token is a bigram, add to document.
                docs[idx].append(token)
  else:
    print("Skipping Bigrams")

  # Create a dictionary representation of the documents.
  dictionary = Dictionary(docs)

  # Filter out words that occur less than 20 documents, or more than 50% of the docum
  dictionary.filter_extremes(no_below=20, no_above=0.5)

  # Bag-of-words representation of the documents.
  corpus = [dictionary.doc2bow(doc) for doc in docs]
  print('\n')
  print('Number of unique tokens: %d' % len(dictionary))
  print('Number of documents: %d' % len(corpus))

  return(dictionary, corpus)
```

Model Training

Using the dictionary and corpus generated from preprocessing the text, we set the training paarmeter function return s the model created for further visualising the returns.

```
def model_training(dictionary, corpus):
  # Train LDA model.


  # Set training parameters.
  NUM_TOPICS = 4
  chunksize = 2000
  passes = 20
  iterations = 400
  eval_every = None  # Don't evaluate model perplexity, takes too much time.

  # Make a index to word dictionary.
  temp = dictionary[0]  # This is only to "load" the dictionary.
  id2word = dictionary.id2token

  model = LdaModel(
      corpus=corpus,
      id2word=id2word,
      chunksize=chunksize,
      alpha='auto',
      eta='auto',
      iterations=iterations,
      num_topics=NUM_TOPICS,
```

```
      passes=passes,
      eval_every=eval_every
  )
  outputfile = f'model{NUM_TOPICS}.gensim'
  print("Saving model in " + outputfile)
  print("")
  model.save(outputfile)
  return(model)
```

Visualisation

Finally, I will use the dictionary, model and corpus generated from the previous functions and pass th visual reprenstation of topic modelling is returned.

```
def visualisation(model, corpus, dictionary):
  lda_display = pyLDAvis.gensim.prepare(model, corpus, dictionary, sort_topics=False)
  return(pyLDAvis.display(lda_display))
```

Now I'll call the three functions for each configuration:

1. Smaller training set with Bigrams
2. Smaller trainig set without Bigrams
3. Bigger training set with Bigrams
4. Bigger training set without Bigrams

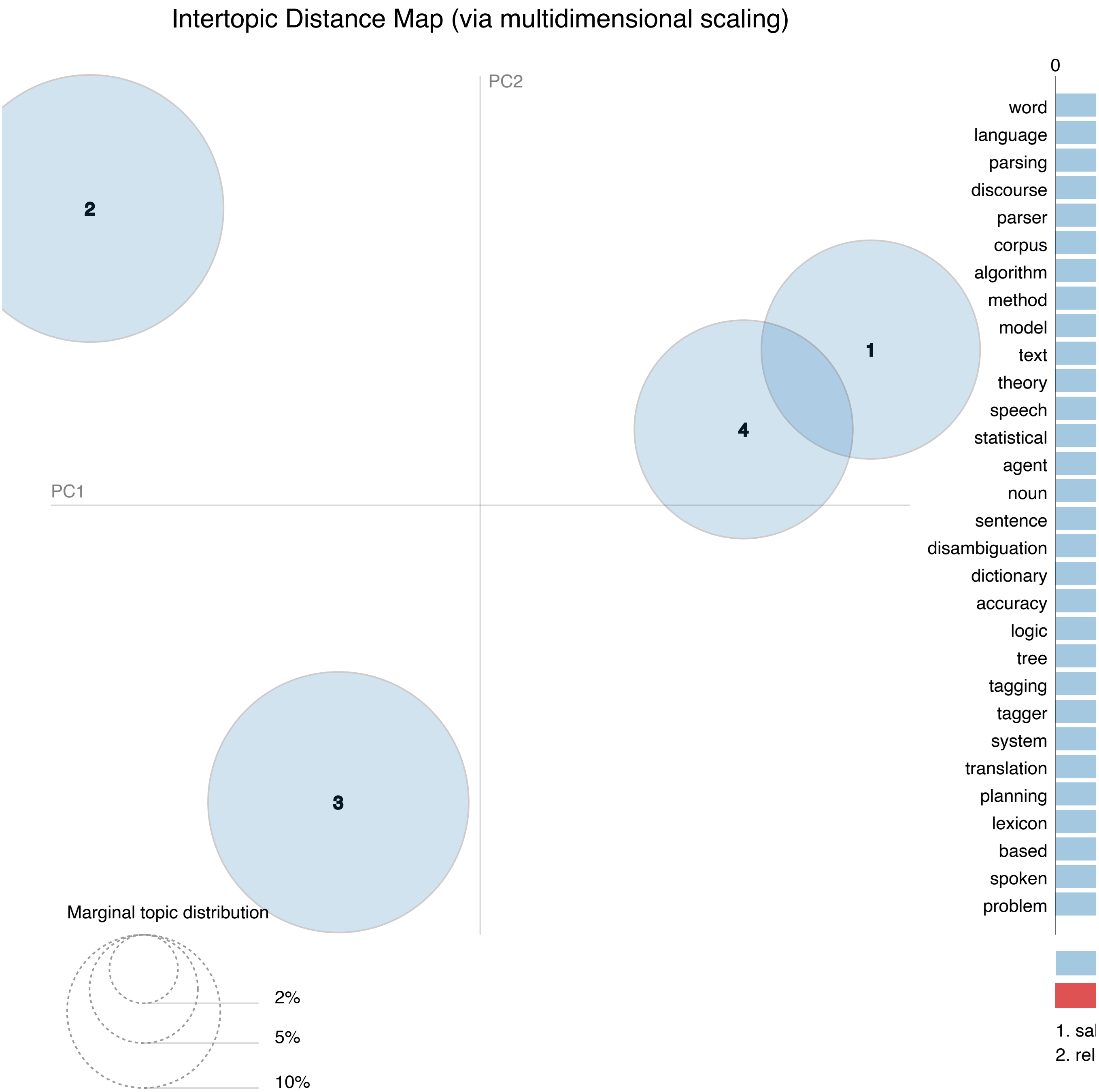
1.1 First 1000 records without Bigrams.

```
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging

# load up this data
text_data = []
df = pd.read_csv('/content/drive/Shareddrives/fit5212-s1-2021-tutorials/A1/axcs_train
df = df[:1000]
docs = df['Abstract'].tolist()
print('Number of articles in training set: ', len(docs))
raw_docs = docs.copy()

dictionary, corpus = pre_process(docs, False)
model = model_training(dictionary, corpus)
visualisation(model,corpus,dictionary)
```

```
Number of articles in training set:  1000
2021-04-25 11:25:52,393 : INFO : adding document #0 to Dictionary(0 unique tokens
2021-04-25 11:25:52,494 : INFO : built Dictionary(6630 unique tokens: ['assuming
2021-04-25 11:25:52,514 : INFO : discarding 5950 tokens: [('assuming', 7), ('conv
2021-04-25 11:25:52,515 : INFO : keeping 680 tokens which were in no less than 2
2021-04-25 11:25:52,518 : INFO : resulting dictionary: Dictionary(680 unique tok
2021-04-25 11:25:52,588 : INFO : using autotuned alpha, starting with [0.25, 0.2
2021-04-25 11:25:52,591 : INFO : using serial LDA version on this node
Skipping Bigrams


Number of unique tokens: 680
Number of documents: 1000
2021-04-25 11:25:52,599 : INFO : running online (multi-pass) LDA training, 4 top
2021-04-25 11:25:52,601 : INFO : PROGRESS: pass 0, at document #1000/1000
2021-04-25 11:25:55,199 : INFO : optimized alpha [0.009431198, 0.12684974, 0.098
2021-04-25 11:25:55,201 : INFO : topic #0 (0.009): 0.015*"be" + 0.012*"can" + 0.
2021-04-25 11:25:55,204 : INFO : topic #1 (0.127): 0.015*"model" + 0.014*"be" +
2021-04-25 11:25:55,207 : INFO : topic #2 (0.099): 0.019*"language" + 0.016*"by"
2021-04-25 11:25:55,210 : INFO : topic #3 (0.034): 0.013*"grammar" + 0.013*"from
2021-04-25 11:25:55,215 : INFO : topic diff=0.742527, rho=1.000000
2021-04-25 11:25:55,218 : INFO : PROGRESS: pass 1, at document #1000/1000
2021-04-25 11:25:56,347 : INFO : optimized alpha [0.010240176, 0.092822045, 0.07
2021-04-25 11:25:56,349 : INFO : topic #0 (0.010): 0.014*"be" + 0.013*"word" + 0
2021-04-25 11:25:56,354 : INFO : topic #1 (0.093): 0.015*"model" + 0.014*"be" +
2021-04-25 11:25:56,357 : INFO : topic #2 (0.077): 0.021*"language" + 0.016*"a" -
2021-04-25 11:25:56,360 : INFO : topic #3 (0.034): 0.014*"grammar" + 0.013*"from
2021-04-25 11:25:56,361 : INFO : topic diff=0.126149, rho=0.577350
2021-04-25 11:25:56,369 : INFO : PROGRESS: pass 2, at document #1000/1000
2021-04-25 11:25:57,387 : INFO : optimized alpha [0.011069208, 0.079387106, 0.06
2021-04-25 11:25:57,389 : INFO : topic #0 (0.011): 0.014*"word" + 0.013*"parsing
2021-04-25 11:25:57,393 : INFO : topic #1 (0.079): 0.015*"model" + 0.015*"be" +
2021-04-25 11:25:57,397 : INFO : topic #2 (0.067): 0.023*"language" + 0.017*"sys
2021-04-25 11:25:57,399 : INFO : topic #3 (0.033): 0.014*"grammar" + 0.013*"mode
2021-04-25 11:25:57,401 : INFO : topic diff=0.102789, rho=0.500000
2021-04-25 11:25:57,404 : INFO : PROGRESS: pass 3, at document #1000/1000
2021-04-25 11:25:58,330 : INFO : optimized alpha [0.011954197, 0.07178612, 0.061
2021-04-25 11:25:58,332 : INFO : topic #0 (0.012): 0.016*"word" + 0.014*"parsing
2021-04-25 11:25:58,334 : INFO : topic #1 (0.072): 0.015*"be" + 0.015*"model" +
2021-04-25 11:25:58,336 : INFO : topic #2 (0.062): 0.024*"language" + 0.018*"sys
2021-04-25 11:25:58,339 : INFO : topic #3 (0.033): 0.014*"grammar" + 0.014*"mode
2021-04-25 11:25:58,342 : INFO : topic diff=0.082079, rho=0.447214
2021-04-25 11:25:58,346 : INFO : PROGRESS: pass 4, at document #1000/1000
2021-04-25 11:25:59,185 : INFO : optimized alpha [0.012822084, 0.06706446, 0.058
2021-04-25 11:25:59,190 : INFO : topic #0 (0.013): 0.017*"word" + 0.015*"based" -
2021-04-25 11:25:59,193 : INFO : topic #1 (0.067): 0.015*"be" + 0.015*"model" +
2021-04-25 11:25:59,197 : INFO : topic #2 (0.058): 0.025*"language" + 0.019*"sys
2021-04-25 11:25:59,198 : INFO : topic #3 (0.032): 0.015*"model" + 0.014*"gramma
2021-04-25 11:25:59,201 : INFO : topic diff=0.066001, rho=0.408248
2021-04-25 11:25:59,203 : INFO : PROGRESS: pass 5, at document #1000/1000
2021-04-25 11:25:59,994 : INFO : optimized alpha [0.013667848, 0.0638764, 0.0557
2021-04-25 11:25:59,996 : INFO : topic #0 (0.014): 0.017*"word" + 0.016*"based" -
2021-04-25 11:26:00,000 : INFO : topic #1 (0.064): 0.016*"be" + 0.014*"it" + 0.0
2021-04-25 11:26:00,001 : INFO : topic #2 (0.056): 0.026*"language" + 0.019*"sys
2021-04-25 11:26:00,003 : INFO : topic #3 (0.032): 0.016*"model" + 0.014*"word" -
2021-04-25 11:26:00,005 : INFO : topic diff=0.054125, rho=0.377964
2021-04-25 11:26:00,008 : INFO : PROGRESS: pass 6, at document #1000/1000
```

```
2021-04-25 11:26:06,150 : INFO : topic #1 (0.053): 0.016*"be" + 0.014*"it" + 0.0
2021-04-25 11:26:06,154 : INFO : topic #2 (0.050): 0.028*"language" + 0.021*"sys
2021-04-25 11:26:06,154 : INFO : topic #3 (0.031): 0.020*"model" + 0.015*"word" +
2021-04-25 11:26:06,156 : INFO : topic diff=0.016021, rho=0.250000
2021-04-25 11:26:06,158 : INFO : PROGRESS: pass 15, at document #1000/1000
2021-04-25 11:26:06,839 : INFO : optimized alpha [0.02045913, 0.052654803, 0.049
2021-04-25 11:26:06,841 : INFO : topic #0 (0.020): 0.019*"word" + 0.017*"based" +
2021-04-25 11:26:06,845 : INFO : topic #1 (0.053): 0.016*"be" + 0.014*"it" + 0.0
2021-04-25 11:26:06,847 : INFO : topic #2 (0.050): 0.028*"language" + 0.021*"sys
2021-04-25 11:26:06,851 : INFO : topic #3 (0.031): 0.020*"model" + 0.016*"word" +
2021-04-25 11:26:06,854 : INFO : topic diff=0.015063, rho=0.242536
2021-04-25 11:26:06,860 : INFO : PROGRESS: pass 16, at document #1000/1000
2021-04-25 11:26:07,467 : INFO : optimized alpha [0.02098118, 0.0522637, 0.049460
2021-04-25 11:26:07,469 : INFO : topic #0 (0.021): 0.019*"word" + 0.017*"based" +
2021-04-25 11:26:07,471 : INFO : topic #1 (0.052): 0.017*"be" + 0.014*"it" + 0.0
2021-04-25 11:26:07,474 : INFO : topic #2 (0.049): 0.028*"language" + 0.021*"sys
2021-04-25 11:26:07,476 : INFO : topic #3 (0.031): 0.021*"model" + 0.016*"word" +
2021-04-25 11:26:07,478 : INFO : topic diff=0.013813, rho=0.235702
2021-04-25 11:26:07,481 : INFO : PROGRESS: pass 17, at document #1000/1000
2021-04-25 11:26:08,066 : INFO : optimized alpha [0.021494994, 0.05189599, 0.049
2021-04-25 11:26:08,070 : INFO : topic #0 (0.021): 0.019*"word" + 0.017*"based" +
2021-04-25 11:26:08,073 : INFO : topic #1 (0.052): 0.017*"be" + 0.014*"it" + 0.0
2021-04-25 11:26:08,076 : INFO : topic #2 (0.049): 0.028*"language" + 0.021*"sys
2021-04-25 11:26:08,082 : INFO : topic #3 (0.031): 0.021*"model" + 0.016*"word" +
2021-04-25 11:26:08,083 : INFO : topic diff=0.012732, rho=0.229416
2021-04-25 11:26:08,088 : INFO : PROGRESS: pass 18, at document #1000/1000
2021-04-25 11:26:08,678 : INFO : optimized alpha [0.021982525, 0.05159373, 0.049
2021-04-25 11:26:08,680 : INFO : topic #0 (0.022): 0.019*"word" + 0.017*"based" +
2021-04-25 11:26:08,684 : INFO : topic #1 (0.052): 0.017*"be" + 0.014*"it" + 0.0
2021-04-25 11:26:08,687 : INFO : topic #2 (0.050): 0.028*"language" + 0.021*"sys
2021-04-25 11:26:08,688 : INFO : topic #3 (0.031): 0.021*"model" + 0.016*"word" +
2021-04-25 11:26:08,690 : INFO : topic diff=0.011712, rho=0.223607
2021-04-25 11:26:08,693 : INFO : PROGRESS: pass 19, at document #1000/1000
2021-04-25 11:26:09,348 : INFO : optimized alpha [0.022444164, 0.051303525, 0.04
2021-04-25 11:26:09,350 : INFO : topic #0 (0.022): 0.019*"word" + 0.017*"based" +
2021-04-25 11:26:09,351 : INFO : topic #1 (0.051): 0.017*"be" + 0.014*"it" + 0.0
2021-04-25 11:26:09,359 : INFO : topic #2 (0.050): 0.028*"language" + 0.021*"sys
2021-04-25 11:26:09,361 : INFO : topic #3 (0.031): 0.022*"model" + 0.016*"word" +
2021-04-25 11:26:09,364 : INFO : topic diff=0.010994, rho=0.218218
2021-04-25 11:26:09,370 : INFO : saving LdaState object under model4.gensim.stat
/usr/local/lib/python3.7/dist-packages/smart_open/smart_open_lib.py:479: Depreca
  warnings.warn(message, category=DeprecationWarning)
2021-04-25 11:26:09,372 : INFO : saved model4.gensim.state
/usr/local/lib/python3.7/dist-packages/smart_open/smart_open_lib.py:479: Depreca
  warnings.warn(message, category=DeprecationWarning)
2021-04-25 11:26:09,375 : INFO : saving LdaModel object under model4.gensim, sep
2021-04-25 11:26:09,378 : INFO : storing np array 'expElogbeta' to model4.gensim
2021-04-25 11:26:09,383 : INFO : not storing attribute dispatcher
2021-04-25 11:26:09,385 : INFO : not storing attribute state
2021-04-25 11:26:09,387 : INFO : not storing attribute id2word
/usr/local/lib/python3.7/dist-packages/smart_open/smart_open_lib.py:479: Depreca
  warnings.warn(message, category=DeprecationWarning)
2021-04-25 11:26:09,391 : INFO : saved model4.gensim
Saving model in model4.gensim
```

Selected Topic: 0 Previous Topic Next Topic Clear Topic Slid



1.2 First 1000 records with Bigrams

```
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging

# load up this data
text_data = []
df = pd.read_csv('/content/drive/Shareddrives/fit5212-s1-2021-tutorials/A1/axcs_train
df = df[:1000]
docs = df['Abstract'].tolist()
print('Number of articles in training set: ', len(docs))
raw_docs = docs.copy()
```

```
  warnings.warn(message, category=DeprecationWarning)
2021-04-25 11:26:30,393 : INFO : saved model4.gensim
Saving model in model4.gensim
```

2.1 First 20,000 records without Bigrams

```
# load up this data
text_data = []
df = pd.read_csv('/content/drive/Shareddrives/fit5212-s1-2021-tutorials/A1/axcs_train
df = df[:20000]
docs = df['Abstract'].tolist()
print('Number of articles in training set: ', len(docs))
raw_docs = docs.copy()

dictionary, corpus = pre_process(docs, False)
model = model_training(dictionary, corpus)
visualisation(model,corpus,dictionary)
```

```
Number of articles in training set:  20000
2021-04-25 11:37:25,692 : INFO : adding document #0 to Dictionary(0 unique tokens
Skipping Bigrams
2021-04-25 11:37:26,598 : INFO : adding document #10000 to Dictionary(26612 uniqu
2021-04-25 11:37:27,570 : INFO : built Dictionary(38303 unique tokens: ['assuming
2021-04-25 11:37:27,647 : INFO : discarding 33048 tokens: [('in', 17737), ('is',
2021-04-25 11:37:27,648 : INFO : keeping 5255 tokens which were in no less than 2
2021-04-25 11:37:27,664 : INFO : resulting dictionary: Dictionary(5255 unique tok
2021-04-25 11:37:29,057 : INFO : using autotuned alpha, starting with [0.25, 0.25
2021-04-25 11:37:29,060 : INFO : using serial LDA version on this node
2021-04-25 11:37:29,069 : INFO : running online (multi-pass) LDA training, 4 topi
2021-04-25 11:37:29,070 : INFO : PROGRESS: pass 0, at document #2000/20000


Number of unique tokens: 5255
Number of documents: 20000
2021-04-25 11:37:34,924 : INFO : optimized alpha [0.09071927, 0.078422055, 0.0597
2021-04-25 11:37:34,925 : INFO : merging changes from 2000 documents into a model
2021-04-25 11:37:34,934 : INFO : topic #0 (0.091): 0.011*"model" + 0.011*"system"
2021-04-25 11:37:34,938 : INFO : topic #1 (0.078): 0.011*"algorithm" + 0.010*"lan
2021-04-25 11:37:34,939 : INFO : topic #2 (0.060): 0.011*"model" + 0.009*"languag
2021-04-25 11:37:34,943 : INFO : topic #3 (0.036): 0.013*"a" + 0.009*"it" + 0.009
2021-04-25 11:37:34,944 : INFO : topic diff=2.835881, rho=1.000000
2021-04-25 11:37:34,954 : WARNING : updated prior not positive
2021-04-25 11:37:34,955 : INFO : PROGRESS: pass 0, at document #4000/20000
2021-04-25 11:37:37,767 : INFO : optimized alpha [0.087769255, 0.087944746, 0.064
2021-04-25 11:37:37,770 : INFO : merging changes from 2000 documents into a model
2021-04-25 11:37:37,778 : INFO : topic #0 (0.088): 0.011*"system" + 0.010*"model"
2021-04-25 11:37:37,779 : INFO : topic #1 (0.088): 0.012*"algorithm" + 0.009*"a"
2021-04-25 11:37:37,782 : INFO : topic #2 (0.064): 0.013*"quantum" + 0.012*"model
2021-04-25 11:37:37,785 : INFO : topic #3 (0.046): 0.013*"a" + 0.009*"it" + 0.009
2021-04-25 11:37:37,787 : INFO : topic diff=0.883455, rho=0.707107
2021-04-25 11:37:37,797 : INFO : PROGRESS: pass 0, at document #6000/20000
2021-04-25 11:37:40,043 : INFO : optimized alpha [0.09314304, 0.09800852, 0.06986
2021-04-25 11:37:40,044 : INFO : merging changes from 2000 documents into a model
2021-04-25 11:37:40,050 : INFO : topic #0 (0.093): 0.011*"system" + 0.009*"inform
2021-04-25 11:37:40,051 : INFO : topic #1 (0.098): 0.013*"algorithm" + 0.009*"net
2021-04-25 11:37:40,053 : INFO : topic #2 (0.070): 0.011*"quantum" + 0.010*"model
2021-04-25 11:37:40,056 : INFO : topic #3 (0.059): 0.011*"a" + 0.009*"code" + 0.0
2021-04-25 11:37:40,059 : INFO : topic diff=1.097633, rho=0.577350
2021-04-25 11:37:40,069 : INFO : PROGRESS: pass 0, at document #8000/20000
2021-04-25 11:37:41,992 : INFO : optimized alpha [0.09814961, 0.11151452, 0.0741
2021-04-25 11:37:41,993 : INFO : merging changes from 2000 documents into a model
2021-04-25 11:37:42,001 : INFO : topic #0 (0.098): 0.011*"channel" + 0.010*"syste
2021-04-25 11:37:42,003 : INFO : topic #1 (0.112): 0.011*"algorithm" + 0.010*"net
2021-04-25 11:37:42,004 : INFO : topic #2 (0.074): 0.009*"quantum" + 0.008*"bound
2021-04-25 11:37:42,007 : INFO : topic #3 (0.071): 0.011*"a" + 0.009*"code" + 0.0
2021-04-25 11:37:42,008 : INFO : topic diff=0.604172, rho=0.500000
2021-04-25 11:37:42,020 : INFO : PROGRESS: pass 0, at document #10000/20000
2021-04-25 11:37:43,941 : INFO : optimized alpha [0.10515009, 0.120686635, 0.081
2021-04-25 11:37:43,942 : INFO : merging changes from 2000 documents into a model
2021-04-25 11:37:43,948 : INFO : topic #0 (0.105): 0.013*"channel" + 0.010*"syste
2021-04-25 11:37:43,950 : INFO : topic #1 (0.121): 0.012*"network" + 0.011*"algor
2021-04-25 11:37:43,951 : INFO : topic #2 (0.081): 0.010*"bound" + 0.008*"channel
2021-04-25 11:37:43,954 : INFO : topic #3 (0.083): 0.011*"a" + 0.009*"code" + 0.0
2021-04-25 11:37:43,955 : INFO : topic diff=0.388050, rho=0.447214
2021-04-25 11:37:43,966 : INFO : PROGRESS: pass 0, at document #12000/20000
```

```
2021-04-25 11:41:09,163 : INFO : topic diff=0.059782, rho=0.182574
2021-04-25 11:41:09,174 : INFO : saving LdaState object under model4.gensim.state
/usr/local/lib/python3.7/dist-packages/smart_open/smart_open_lib.py:479: Depreca
  warnings.warn(message, category=DeprecationWarning)
2021-04-25 11:41:09,179 : INFO : saved model4.gensim.state
/usr/local/lib/python3.7/dist-packages/smart_open/smart_open_lib.py:479: Depreca
  warnings.warn(message, category=DeprecationWarning)
2021-04-25 11:41:09,185 : INFO : saving LdaModel object under model4.gensim, sepa
2021-04-25 11:41:09,188 : INFO : storing np array 'expElogbeta' to model4.gensim
2021-04-25 11:41:09,191 : INFO : not storing attribute dispatcher
2021-04-25 11:41:09,195 : INFO : not storing attribute state
2021-04-25 11:41:09,196 : INFO : not storing attribute id2word
/usr/local/lib/python3.7/dist-packages/smart_open/smart_open_lib.py:479: Depreca
  warnings.warn(message, category=DeprecationWarning)
2021-04-25 11:41:09,202 : INFO : saved model4.gensim
Saving model in model4.gensim
```

Selected Topic: 0 Previous Topic Next Topic Clear Topic Slid

Intertopic Distance Map (via multidimensional scaling)

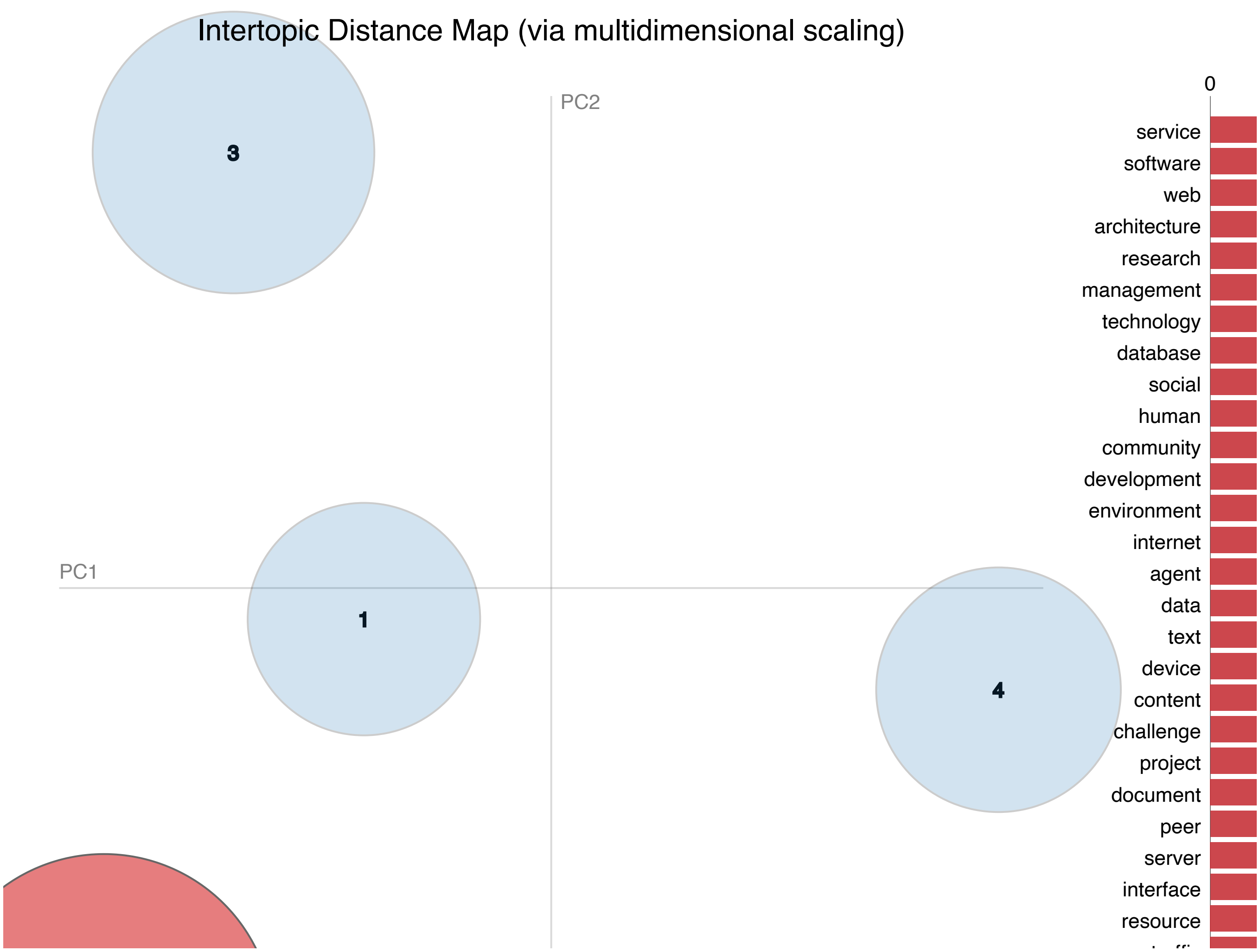
2.2 First 20,000 records with Bigrams

```
# load up this data
text_data = []
df = pd.read_csv('/content/drive/Shareddrives/fit5212-s1-2021-tutorials/A1/axcs_train
df = df[:20000]
docs = df['Abstract'].tolist()
print('Number of articles in training set: ', len(docs))
raw_docs = docs.copy()

dictionary, corpus = pre_process(docs, True)
model = model_training(dictionary, corpus)
visualisation(model,corpus,dictionary)
```

```
Number of articles in training set:  20000
2021-04-25 11:41:31,039 : INFO : collecting all words and their counts
2021-04-25 11:41:31,040 : INFO : PROGRESS: at sentence #0, processed 0 words and
Bigrams Added
2021-04-25 11:41:33,052 : INFO : PROGRESS: at sentence #10000, processed 1290845
2021-04-25 11:41:35,292 : INFO : collected 702068 word types from a corpus of 27
2021-04-25 11:41:35,293 : INFO : using 702068 counts as vocab in Phrases<0 vocab
/usr/local/lib/python3.7/dist-packages/gensim/models/phrases.py:598: UserWarning
  warnings.warn("For a faster implementation, use the gensim.models.phrases.Phra
2021-04-25 11:41:44,694 : INFO : adding document #0 to Dictionary(0 unique token
2021-04-25 11:41:45,687 : INFO : adding document #10000 to Dictionary(28169 uniq
2021-04-25 11:41:46,776 : INFO : built Dictionary(39881 unique tokens: ['assumin
2021-04-25 11:41:46,880 : INFO : discarding 33407 tokens: [('in', 17737), ('is',
2021-04-25 11:41:46,881 : INFO : keeping 6474 tokens which were in no less than
2021-04-25 11:41:46,898 : INFO : resulting dictionary: Dictionary(6474 unique to
2021-04-25 11:41:48,493 : INFO : using autotuned alpha, starting with [0.25, 0.2
2021-04-25 11:41:48,495 : INFO : using serial LDA version on this node
2021-04-25 11:41:48,504 : INFO : running online (multi-pass) LDA training, 4 top
2021-04-25 11:41:48,505 : INFO : PROGRESS: pass 0, at document #2000/20000


Number of unique tokens: 6474
Number of documents: 20000
2021-04-25 11:41:54,401 : INFO : optimized alpha [0.042382985, 0.07852848, 0.078
2021-04-25 11:41:54,403 : INFO : merging changes from 2000 documents into a mode
2021-04-25 11:41:54,412 : INFO : topic #0 (0.042): 0.014*"model" + 0.010*"a" + 0
2021-04-25 11:41:54,413 : INFO : topic #1 (0.079): 0.010*"a" + 0.009*"system" +
2021-04-25 11:41:54,415 : INFO : topic #2 (0.078): 0.009*"language" + 0.009*"it"
2021-04-25 11:41:54,418 : INFO : topic #3 (0.048): 0.010*"be" + 0.009*"can" + 0.
2021-04-25 11:41:54,420 : INFO : topic diff=3.036457, rho=1.000000
2021-04-25 11:41:54,432 : WARNING : updated prior not positive
2021-04-25 11:41:54,433 : INFO : PROGRESS: pass 0, at document #4000/20000
2021-04-25 11:41:56,918 : INFO : optimized alpha [0.050672114, 0.09463162, 0.079
2021-04-25 11:41:56,920 : INFO : merging changes from 2000 documents into a mode
2021-04-25 11:41:56,927 : INFO : topic #0 (0.051): 0.013*"model" + 0.012*"quantu
2021-04-25 11:41:56,929 : INFO : topic #1 (0.095): 0.010*"system" + 0.009*"a" +
2021-04-25 11:41:56,931 : INFO : topic #2 (0.080): 0.009*"problem" + 0.008*"it"
2021-04-25 11:41:56,934 : INFO : topic #3 (0.060): 0.010*"be" + 0.008*"can" + 0.
2021-04-25 11:41:56,936 : INFO : topic diff=0.987760, rho=0.707107
2021-04-25 11:41:56,948 : INFO : PROGRESS: pass 0, at document #6000/20000
2021-04-25 11:41:58,821 : INFO : optimized alpha [0.058916904, 0.10948626, 0.086
2021-04-25 11:41:58,823 : INFO : merging changes from 2000 documents into a mode
2021-04-25 11:41:58,831 : INFO : topic #0 (0.059): 0.013*"model" + 0.011*"quantu
2021-04-25 11:41:58,833 : INFO : topic #1 (0.109): 0.010*"network" + 0.010*"syst
2021-04-25 11:41:58,836 : INFO : topic #2 (0.087): 0.010*"problem" + 0.008*"it"
2021-04-25 11:41:58,838 : INFO : topic #3 (0.073): 0.012*"code" + 0.011*"channel
2021-04-25 11:41:58,840 : INFO : topic diff=1.226538, rho=0.577350
2021-04-25 11:41:58,851 : INFO : PROGRESS: pass 0, at document #8000/20000
2021-04-25 11:42:00,452 : INFO : optimized alpha [0.06606102, 0.12771736, 0.0933
2021-04-25 11:42:00,454 : INFO : merging changes from 2000 documents into a mode
2021-04-25 11:42:00,460 : INFO : topic #0 (0.066): 0.013*"model" + 0.009*"a" + 0
2021-04-25 11:42:00,462 : INFO : topic #1 (0.128): 0.011*"network" + 0.010*"syst
2021-04-25 11:42:00,464 : INFO : topic #2 (0.093): 0.011*"problem" + 0.008*"a" +
2021-04-25 11:42:00,465 : INFO : topic #3 (0.086): 0.013*"channel" + 0.012*"code
2021-04-25 11:42:00,468 : INFO : topic diff=0.694497, rho=0.500000
2021-04-25 11:42:00,481 : INFO : PROGRESS: pass 0, at document #10000/20000
2021-04-25 11:42:01,900 : INFO : optimized alpha [0.07192906, 0.1383983, 0.09983
```

```
2021-04-25 11:45:11,002 : INFO : PROGRESS: pass 19, at document #20000/20000
2021-04-25 11:45:11,927 : INFO : optimized alpha [0.18327516, 0.29783577, 0.2359
2021-04-25 11:45:11,930 : INFO : merging changes from 2000 documents into a mode
2021-04-25 11:45:11,943 : INFO : topic #0 (0.183): 0.010*"method" + 0.010*"model"
2021-04-25 11:45:11,945 : INFO : topic #1 (0.298): 0.011*"system" + 0.011*"netwo
2021-04-25 11:45:11,955 : INFO : topic #2 (0.236): 0.015*"problem" + 0.013*"algo
2021-04-25 11:45:11,958 : INFO : topic #3 (0.130): 0.022*"channel" + 0.015*"code"
2021-04-25 11:45:11,959 : INFO : topic diff=0.063862, rho=0.182574
2021-04-25 11:45:11,973 : INFO : saving LdaState object under model4.gensim.state
/usr/local/lib/python3.7/dist-packages/smart_open/smart_open_lib.py:479: Depreca
  warnings.warn(message, category=DeprecationWarning)
2021-04-25 11:45:11,977 : INFO : saved model4.gensim.state
/usr/local/lib/python3.7/dist-packages/smart_open/smart_open_lib.py:479: Depreca
  warnings.warn(message, category=DeprecationWarning)
2021-04-25 11:45:11,984 : INFO : saving LdaModel object under model4.gensim, sep
2021-04-25 11:45:11,986 : INFO : storing np array 'expElogbeta' to model4.gensim
2021-04-25 11:45:11,988 : INFO : not storing attribute dispatcher
2021-04-25 11:45:11,990 : INFO : not storing attribute state
2021-04-25 11:45:11,991 : INFO : not storing attribute id2word
/usr/local/lib/python3.7/dist-packages/smart_open/smart_open_lib.py:479: Depreca
  warnings.warn(message, category=DeprecationWarning)
2021-04-25 11:45:11,995 : INFO : saved model4.gensim
Saving model in model4.gensim
```

Selected Topic: 2 Previous Topic Next Topic Clear Topic Slid



Aggregating this information in a table

```
def get_document_topics(ldamodel=model, corpus=corpus, texts=raw_docs):
   # Init output
    document_topics_df = pd.DataFrame()

    # Get main topic in each document
    for i, row in enumerate(ldamodel[corpus]):
        row = sorted(row, key=lambda x: (x[1]), reverse=True)
        # Get the Dominant topic, Perc Contribution and Keywords for each document
        for j, (topic_num, prop_topic) in enumerate(row):
            if j == 0:  # => dominant topic
                wp = ldamodel.show_topic(topic_num)
                topic_keywords = ", ".join([word for word, prop in wp])
                document_topics_df = document_topics_df.append(pd.Series([int(topic_n
            else:
                break
    document_topics_df.columns = ['Dominant_Topic', 'Perc_Contribution', 'Topic_Keywo

    # Add original text to the end of the output
    contents = pd.Series(texts)
    document_topics_df = pd.concat([document_topics_df, contents], axis=1)

    document_topics_df.columns = ['Dominant_Topic', 'Perc_Contribution', 'Topic_Keywo

    return document_topics_df
```

```
doc_topic_df = get_document_topics(ldamodel=model, corpus=corpus, texts=raw_docs)
```

```
doc_topic_df.head()
```

	Dominant_Topic	Perc_Contribution	Topic_Keywords	
0	2.0	0.9759	problem, algorithm, graph, be, set, a, it, whi...	Nes
1	2.0	0.8223	problem, algorithm, graph, be, set, a, it, whi...	A note
2	2.0	0.8860	problem, algorithm, graph, be, set, a, it, whi...	Textbook
3	1.0	0.8310	system, network, a, data, it, based, paper, be...	Theor
4	2.0	0.6912	problem, algorithm, graph, be, set, a, it, whi...	Cont

Find the most representative document for each topic

```
# Group top 5 sentences under each topic
doc_topics_sorted_df = pd.DataFrame()

doc_topic_df_grpd = doc_topic_df.groupby('Dominant_Topic')

for i, grp in doc_topic_df_grpd:
    doc_topics_sorted_df = pd.concat([doc_topics_sorted_df,
                                      grp.sort_values(['Perc_Contribution'], a
                                     axis=0)

doc_topics_sorted_df.reset_index(drop=True, inplace=True)
doc_topics_sorted_df.columns = ['Topic_Num', "Topic_Perc_Contrib", "Keywords", "Text"
doc_topics_sorted_df.head(10)
```

	Topic_Num	Topic_Perc_Contrib	Keywords	
0	0.0	0.9936	method, model, algorithm, a, it, quantum, be, ...	A topol
1	1.0	0.9975	system, network, a, data, it, based, paper, be...	Design &
2	2.0	0.9971	problem, algorithm, graph, be, set, a, it, whi...	A Simple
3	3.0	0.9969	channel, code, network, rate, capacity, scheme...	High-rate S

Find the top-k most representative document for each topic

```
def find_top_k_doc(doc_topic_df=doc_topic_df, k=5):

  doc_topics_sorted_df = pd.DataFrame()

  doc_topic_df_grpd = doc_topic_df.groupby('Dominant_Topic')

  for i, grp in doc_topic_df_grpd:
      doc_topics_sorted_df = pd.concat([doc_topics_sorted_df,
                                        grp.sort_values(['Perc_Contribution'],
                                        axis=0)

  doc_topics_sorted_df.reset_index(drop=True, inplace=True)
  doc_topics_sorted_df.columns = ['Topic_Num', "Topic_Perc_Contrib", "Keywords", "Tex
  return doc_topics_sorted_df
```

```
top_k_df = find_top_k_doc()
top_k_df
```

	Topic_Num	Topic_Perc_Contrib	Keywords	
0	0.0	0.9936	method, model, algorithm, a, it, quantum, be, ...	A topo
1	0.0	0.9934	method, model, algorithm, a, it, quantum, be, ...	A Prob
2	0.0	0.9922	method, model, algorithm, a, it, quantum, be, ...	Multi-Di
3	0.0	0.9917	method, model, algorithm, a, it, quantum, be, ...	The B-
4	0.0	0.9907	method, model, algorithm, a, it, quantum, be, ...	Fin
5	1.0	0.9975	system, network, a, data, it, based, paper, be...	Design &
6	1.0	0.9974	system, network, a, data, it, based, paper, be...	Extrac
7	1.0	0.9973	system, network, a, data, it, based, paper, be...	CDTOM:
8	1.0	0.9972	system, network, a, data, it, based, paper, be...	Une p
9	1.0	0.9972	system, network, a, data, it, based, paper, be...	Bulk Sch
10	2.0	0.9971	problem, algorithm, graph, be, set, a, it, whi...	A Simp
11	2.0	0.9969	problem, algorithm, graph, be, set, a, it, whi...	A new
12	2.0	0.9967	problem, algorithm, graph, be, set, a, it, whi...	On Canon
13	2.0	0.9967	problem, algorithm, graph, be, set, a, it, whi...	Cut-I
14	2.0	0.9963	problem, algorithm, graph, be, set, a, it, whi...	Algorithr
15	3.0	0.9969	channel, code, network, rate, capacity, scheme...	High-rate
16	3.0	0.9968	channel, code, network, rate, capacity, scheme...	Feedback
17	3.0	0.9968	channel, code, network, rate, capacity, scheme...	Study of
18	3.0	0.9968	channel, code, network, rate, capacity, scheme...	Improved
19	3.0	0.9965	channel, code, network, rate, capacity, scheme...	The

✓ 0s completed at 21:46

FIT5212 Assignment 1
Simran Singh Gulati
31125301

Analysing Model Performance

Part A : Text Classification

In an attempt to predict the *subject* a piece of text belongs to, we created a bunch of models — each with a different set of parameters (configurations) to see what combination of preprocessing, training set and model performed best at classifying the given text.

To discuss the results from each configuration, let's start with my approach towards the variation among each model. As given in the assignment specification, we had 3 separate Binary Classification tasks wherein each article had to be tested against the below said subjects:

1. InfoTheory
2. CompVis
3. Math

An article may belong to multiple classes or even none of them, which basically means it could be both InfoTheory and Math related topic or even none of the give labels. This was motivation behind Binary Classification and not multi-class prediction. Thereby we had 8 models (which simply means 8 sets of configurations) for each class.

In those 8 models, we had 2 subsets — a statistical algorithm and a deep learning technique. For the statistical classifier I shortlisted SVM and Logistic Regression, from those taught in the tutorials. As per this blog on Medium, SVM is a better choice than the Logistic Regression for both un-structured and semi-structured data. Plus the author claims SVM isn't as prone to over-fitting as is Logistic Regression. Lastly, as observed in the tutorial for Week 4, Logistic Regression couldn't predict very well when exposed to an imbalanced dataset (or skewed distribution) which is the case with our dataset as well. Thus I decided to go with

SVM for the statistical classification. And a simple RNN for the other subset. In nutshell, we had 4 models using Linear SVC and the other 4 using Simple RNN for each of the 3 classes.

The 2 preprocessing techniques involved lemmatising the text using two different libraries — NLTK and Spacy. Both of them tokenised the text and lemmatised but the difference is that Spacy does Part of Speech (POS) tagging as well. Note that the Spacy is an exhaustive pipeline with plenty of components which also include dependency labelling and entity name detection. Considering the size of our dataset the *nlp* pipeline from Spacy would take infinitely long to process all the text articles, so just like in tutorials we disabled the former two components (parser and ner) to restrict the pipeline to just POS tagging. Plus this allows us to draw a comparison between lemmatisation done by NLTK and Spacy more precisely as the only difference here is POS tagging (only in Spacy, not in NLTK).

Now that we have 3 classification tasks, each modelled on 2 types of algorithms, with 2 different preprocessing techniques — we further train the using 2 different training sets. While one is trained on the entire set of records (close to 54000) the other one restricted to just the top 1000 records. Thereby half the models are are trained on 50 times the size of data in the other half.

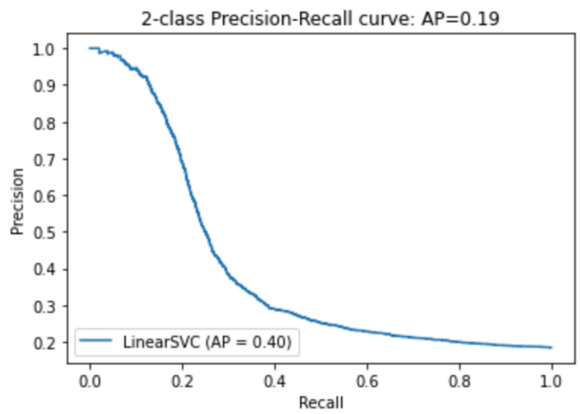
Considering the 24 (3 X 2 X 2 X 2) sets of configuration, I expect better results from the simple RNN and those trained on the bigger dataset. Lastly those process with Spacy are also expected with better performance metrics. Now given the skewness of data, most of the articles are labelled as class 0 (across all three subjects), thereby accuracy might not be the best measure. For example (drawn from the Jupyter notebook attached) : 96% of articles are labelled as 0 for the CompVis class and blindly labelling each article as 0 will return very high accuracy. Considering a serious classification task like in the field of medicine, False Positives may lead to sever repercussion. Therefore we look at Precision, Recall, F1 Score and a precision- recall curve to draw conclusions from our models.

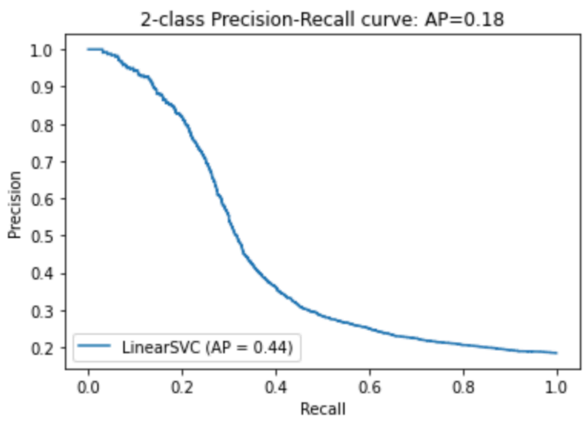
Precision = TP/(TP+FP)

Recall = TP/(TP+FN)

Considering the mathematical definition both these metrics can be *individually* cheated with a *specific case (*or simply imbalanced dataset) to make the denominator equal to TP. This can only be done to either of them and not both. Therefore a third metric — F1 score which is a combination of both seems the best measure for determining the model performance.

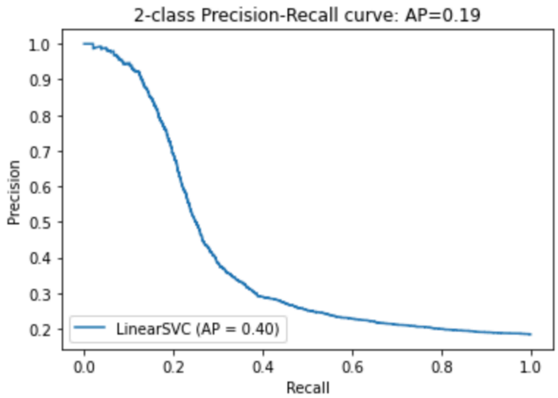
Looking at the graphs, we can see there's not much difference between the two lemmatisers.

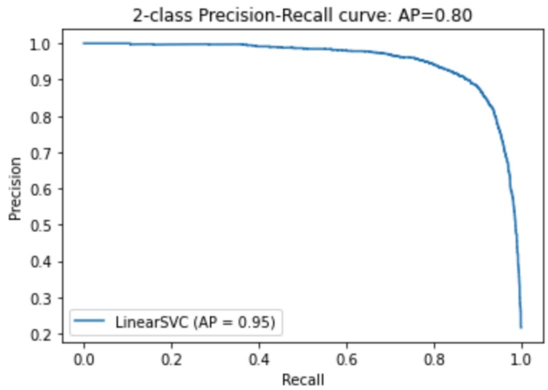




The graph on the left represents configuration 1.1 (refer Jupyter notebook) and the one on right represents 1.2 — both are for InfoTheory classification trained on SVM with the smaller dataset. These were pre-processed using Tf-IDF vectoriser with NLTK LemmaTokeniser (1.1) and Spacy's LemmaTokeniser (1.2). The only difference we can see is there's a slightly gradual decline in Precision for the Spacy (right). And the left one is a *little* steeper. Plus the average precision from Spacy (left) is around 10% better (higher) than NLTK. While I would like to conclude that Spacy is better than NLTK (restricting it to Lemmatisation) but the other graph don't say the same. In fact all other graph have same precision for each pre-processing. In fact, in the last 2 (3.3 and 3.4 — refer Jupyter) NLTK has slightly better average-precision than the Spacy. The difference in above attached graphs (1.1 and 1.2) is not enough to make an overall statement. Had the other two cases (2.1 and 2.2, 3.1 and 3.2) shown same results it would have been wise to generalise. Nonetheless we expected Spacy to better, even though slightly, in this particular case it does align with out expectation.

This motivated me to compare the same configuration (SVM with Spacy) with the one trained on bigger dataset. The graph on left (next page) represents 1.2 (smaller dataset) and that on right represents 1.4 (bigger dataset).





While my focus for comparison would be F-1 score in this case, we cannot ignore the huge difference in the precision-recall curve. The average precision for bigger 1.4 is 0.95 which is more than the double of 1.2. We must note that around 80% of texts in InfoTheory are labelled as 0, so the data is highly skewed. Thus looking at just the precision is not a good idea, the False Positive might be influenced. Moving to the F1 Score, I see 0.45 for 1.2 versus 0.94 for 1.4. Similarly with 2.2 and 2.4 (higher F1), the bigger dataset has better prediction results. It looks all well when I move to 3.2 with a F1 score of 0.45 but the results are surprising when we look at the 3.4 (F1 of 0.4) — Contrary to the trend in InfoTheory and CompVis, Math had better F1 Score with smaller training set. Thus I looked at the percentage of 0 labels in the Math Class for the bigger training set, which was approximately 70%.

Probably the high percentage of '1' labels in smaller training set caused the model to overfit but this can not be the explanation for results from 3.4 as it already has all the data records. In fact the bias due to skewed distribution should have been in favour of class 0, unlike the case here. Thinking on those lines, maybe the other 2 classes (InfoTheory and CompVis) had bias in results due to extremely high proportion of 0 labels (0.8 and 0.9 respectively and SVM failed on Math. Probably it was not the bigger training set that was yielding good results but that the SVM is not a good a choice for a skewed dataset.

This made me look at the Simple RNN performance wherein it *proves* that **SVM was not wrong for Math but was biased due to the skewness in InfoTheory and CompVis classes.** If this statement was overwhelming at this stage, let's build a table for a detailed comparison of the results from the two algorithms. To keep it concise, I'll limit to the models trained on bigger datasets with Spacy's LemmaTokeniser :

		Linear SVC	Simple RNN
InfoTheory	F1	0.92	0.46
	Precision	0.94	0.55
	Recall	0.90	0.50
CompVis	F1	0.92	0.44
	Precision	0.96	0.53
	Recall	0.88	0.50
Math	F1	0.39	0.45
	Precision	0.43	0.55
	Recall	0.37	0.50

The results are drawn from the Jupyter notebook attached (please refer for verification). Here we can clearly see that the SVM had better performance than RNN for both InfoTheory and CompVis. While one may claim that SVM is better than RNN bit things change when we look at the results for Math class. Here the RNN outperformed the SVM by 15% for the F1 score. We can even look at the individual Precision and Recall scores which align well with this observation. Had SVM really been better than RNN, it should have had the same accuracy (general use, not the performance metric) in predicting the labels for Math class as well. Looking at the consistency among RNN for all kinds of data distribution (0.7, 0.8, 0.9) in the class labels I can say Linear SVC was overfitting due to the overfitting.

Although, this is opposite to what were taught in the tutorial where Linear SVC outperformed Simple RNN (where we were taught about tweaking the RNN parameters) — In my experiment I feel the skewness in InfoTheory and CompVis is the reason for not achieving the optimal performance with SVM.

To conclude, Spacy was *not* significantly better than NLTK in lemmatisation. Rather they had identical performance. Next the bigger dataset was clearly a good choice for training the models. Lastly, though RNN didn't have great F1 score when compared to SVM but the results were pretty consistent across all three classes despite the varied skewness in each class. So at least we can say it did not overfit and could handle skewness better than the SVM.

Part B : Topic Modelling

In this section we perform unsupervised clustering (using LDA) on the training data from Part A, to find out the dominant topics in each text article. Just like in last section, we deploy a range of configurations with 2 different training sets and 2 different preprocessing techniques. The two datasets comprise of first 1000 and first 20,000 records respectively. And the two preprocessing techniques would be :

1. Without bigrams :

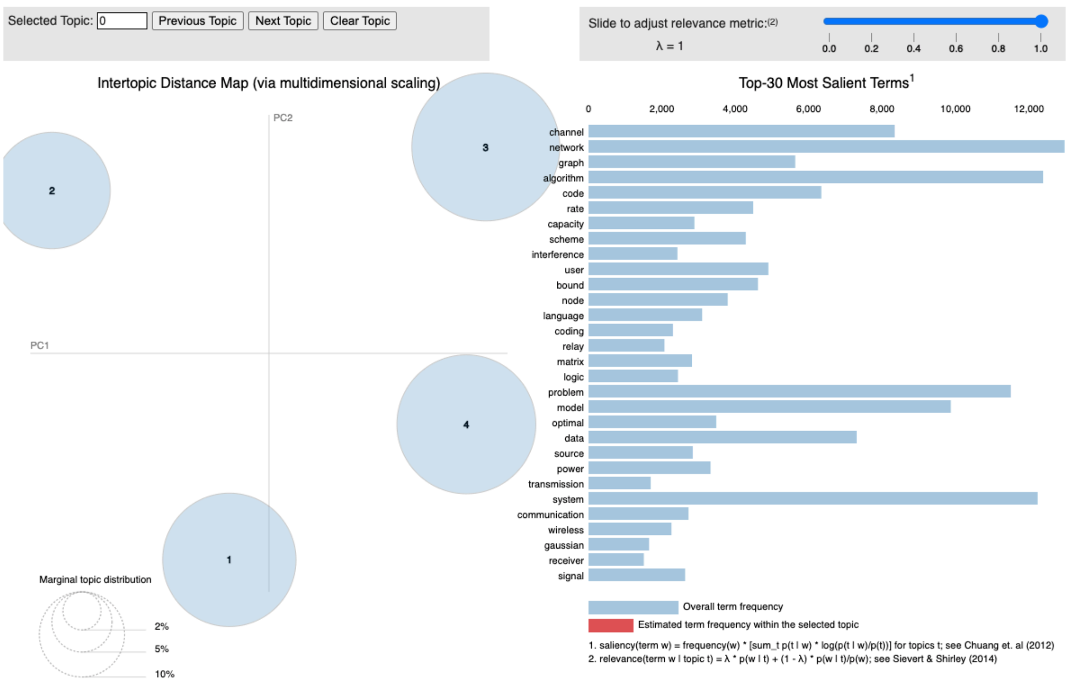
Tokenisation —> Remove stop-words, numbers and single characters —> Remove rare and common tokens —> Bag of Words representation

2. With Bi-Grams

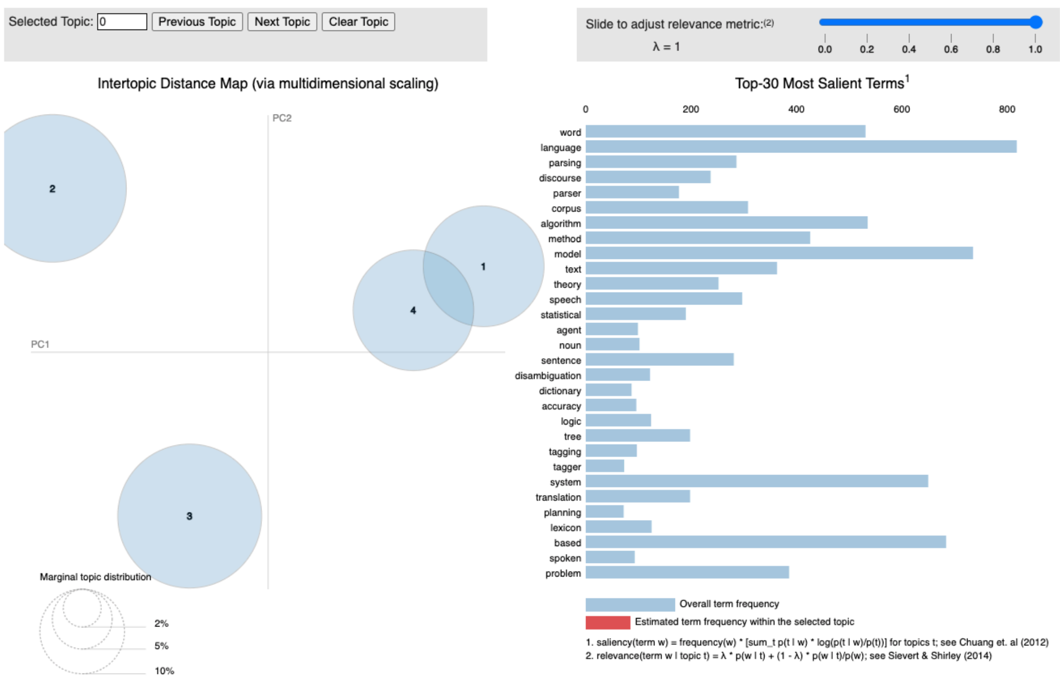
Tokenisation —> Remove stop-words, numbers and single characters —> **Add bigrams and trigrams** —> Remove rare and common tokens —> Bag of Words representation

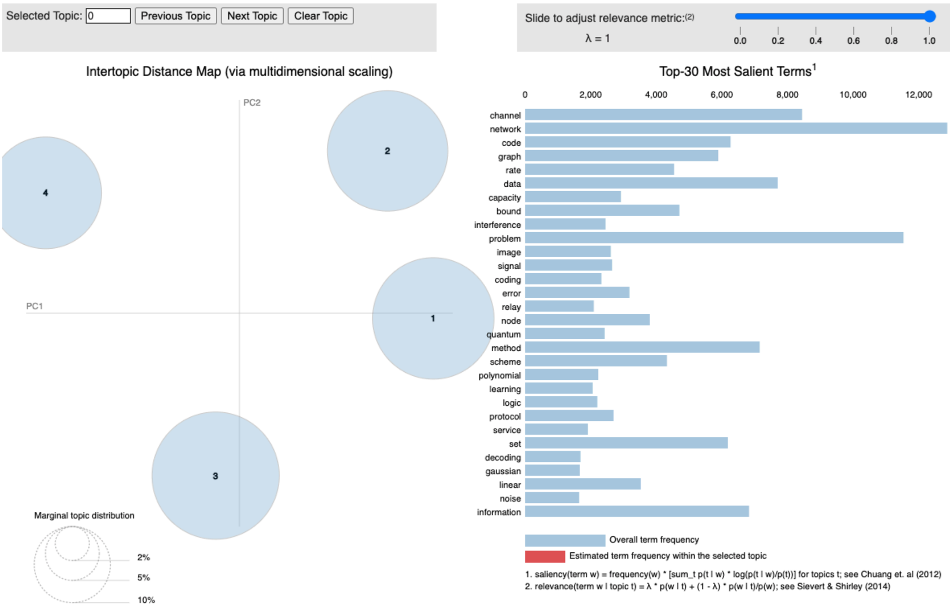
Considering we have to train this LDA on two different data-sets, I am expecting *significantly* better results from variation 2 (with bigrams) on the bigger dataset. Plus we will set number of topics to 4 as:

(i) 3 main clusters - 1 for each class (InfoTheory, CompVis and Math)
(ii) 1 cluster for documents that fall in none of the categories

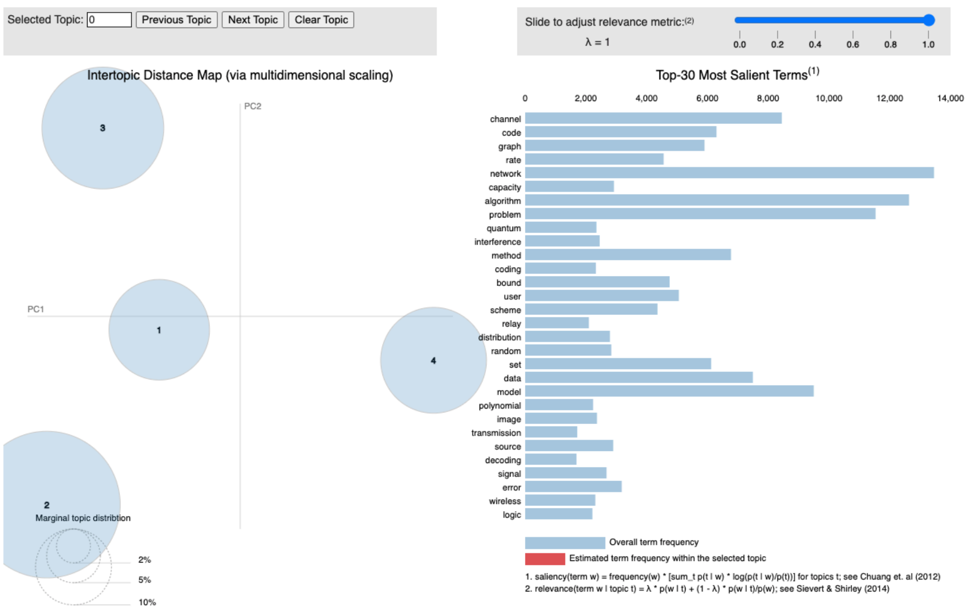


Here's a screenshot (attached above, refer to Jupyter Notebook for interactivity) of the visualisation from one of the LDA models. This is from the *smaller dataset with bigrams*. We can clearly see the 4 topics are far apart from each other, unlike the one attached below which is using the same data (1000 records) but without bigrams. As expected the configuration with bigrams can appropriately differentiate between the given topics.





Referring the visualisation (figure above) from clustering on the bigger dataset without bigrams, we can draw — topics 1 and 2 are pretty close to each other when comparing their distance with clusters 3 and 4. Again as per expectation the addition of bigrams on this bigger dataset again changes the placement of the clusters. The one with bigrams (attached below) can cluster the 3 big topics in a more interpretable manner. For instance — the clusters 2,3 and 4 *may* belong the to the topics InfoTheory, CompVis and Math which are equally distanced from each other with another cluster (labelled as 1) placed right in the middle of the three. The central cluster *may* represent those articles which do not belong to any of the three mentioned classes.



Thus I composed a quick look up table (attached below) to see the common words in each topic.

	Topic_Num	Topic_Perc_Contrib	Keywords	Text
0	0.0	0.9936	method, model, algorithm, a, it, quantum, be, ...	A topological chaos framework for hash functi...
1	0.0	0.9934	method, model, algorithm, a, it, quantum, be, ...	A Probabilistic Model For Sequence Analysis T...
2	0.0	0.9922	method, model, algorithm, a, it, quantum, be, ...	Multi-Dimensional Hash Chains and Application...
3	0.0	0.9917	method, model, algorithm, a, it, quantum, be, ...	The B-Exponential Map: A Generalization of th...
4	0.0	0.9907	method, model, algorithm, a, it, quantum, be, ...	Finite Dimensional Statistical Inference In t...
5	1.0	0.9975	system, network, a, data, it, based, paper, be...	Design & Deploy Web 2.0 enable services over ...
6	1.0	0.9974	system, network, a, data, it, based, paper, be...	Extraction of Flat and Nested Data Records fr...
7	1.0	0.9973	system, network, a, data, it, based, paper, be...	CDTOM: A Context-driven Task-oriented Middlew...
8	1.0	0.9972	system, network, a, data, it, based, paper, be...	Une plate-forme dynamique pour l\'evaluation...
9	1.0	0.9972	system, network, a, data, it, based, paper, be...	Bulk Scheduling with DIANA Scheduler Results ...
10	2.0	0.9971	problem, algorithm, graph, be, set, a, it, whi...	A Simple Polynomial Algorithm for the Longest...
11	2.0	0.9969	problem, algorithm, graph, be, set, a, it, whi...	A new algebraic technique for polynomial-time...
12	2.0	0.9967	problem, algorithm, graph, be, set, a, it, whi...	On Canonical Forms of Complete Problems via F...
13	2.0	0.9967	problem, algorithm, graph, be, set, a, it, whi...	Cut-Elimination and Proof Search for Bi-Intui...
14	2.0	0.9963	problem, algorithm, graph, be, set, a, it, whi...	Algorithmic correspondence and completeness i...
15	3.0	0.9969	channel, code, network, rate, capacity, scheme...	High-rate Space-Time-Frequency Codes Achievin...
16	3.0	0.9968	channel, code, network, rate, capacity, scheme...	Feedback Reduction for Random Beamforming in ...
17	3.0	0.9968	channel, code, network, rate, capacity, scheme...	Study of Gaussian Relay Channels with Correla...
18	3.0	0.9968	channel, code, network, rate, capacity, scheme...	Improved Bounds on the Parity-Check Density a...
19	3.0	0.9965	channel, code, network, rate, capacity, scheme...	The Diversity-Multiplexing Tradeoff of the Dy...

This reveals that the dominant words in the Cluster 1(Topic_Num 0 in table) are method, model, algorithm and quantum.

Cluster 2 comprises of system network and data.

Cluster 3 comprises of problem, algorithm and graph.

Cluster 4 comprises of channel, code, network rate, capacity.

Cluster 3 seems to be **Math** as it has the keywords *graph* and *set.* Cluster 4 looks like Networking (probably **InfoTheory**) articles as it has the keywords *interference, transmission, wireless, network, rate, etc.* Lastly the cluster 2 looks like Human-Computer Interaction or Application development (potentially **CompVis**) as it has keywords *mobile, device, content, platform, architecture, database. etc.*

To conclude, the addition of bigrams helped in differentiating the topics very well. And so did the inclusion of extra 19,000 records in the second dataset. Unlike the Part A, the results are aligning well with our expectations (proposed in the beginning).