Assignment 1 for FIT5212, Semester 1

Student Name: Tsz Yan CHUNG

Student ID: 32973381

```
# !pip3 install torch==1.9.1+cu111 torchvision==0.10.1+cu111 torchaudio==0.9.1 -f https:
```

```
# !pip3 install torchtext==0.10.0
```

Import Libraries

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

```

```
import copy
import random
import time
import re

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

import spacy
from spacy.lang.en.stop_words import STOP_WORDS

import nltk
from nltk.corpus import stopwords
from nltk import word_tokenize
from nltk.tokenize import wordpunct_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.stem import PorterStemmer
nltk.download('punkt')
nltk.download('omw-1.4')
nltk.download('wordnet')
nltk.download('stopwords')

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

```
import torch.optim as optim
from torchtext.legacy import data
from torchtext.legacy.data import Field, LabelField, TabularDataset, Dataset

from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score, con
from sklearn.metrics import precision_recall_curve
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB, GaussianNB, BernoulliNB
from sklearn.svm import LinearSVC, SVC
from sklearn.ensemble import RandomForestClassifier

```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Sunny\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\Sunny\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Sunny\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Sunny\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
%matplotlib inline
import nltk
import torch
import torchtext
from torchtext.legacy import data
from torchtext.legacy.data import Field, LabelField, TabularDataset, Dataset

import nltk
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import spacy

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, accuracy_score, con
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB, GaussianNB, BernoulliNB
from sklearn.svm import LinearSVC, SVC
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_recall_curve


from torchtext.legacy import data
from torchtext.legacy.data import Field, LabelField, TabularDataset, Dataset

import copy
import re
from nltk.stem import PorterStemmer

# Download necessary NLTK datasets
nltk.download('punkt')
nltk.download('omw-1.4')
nltk.download('wordnet')
nltk.download('stopwords')
# Load spacy language model
spacy_en = spacy.load("en_core_web_sm")

# Load NLTK English stopwords
stop_words = set(stopwords.words('english'))

# Define NLTK WordNet lemmatizer
lemmatizer = WordNetLemmatizer()

# Define Porter Stemmer
porter_stemmer = PorterStemmer()

```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Sunny\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\Sunny\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Sunny\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Sunny\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Part 1: Text Classification

In this task, we will first go through the statisical method applied for this classification task, followed by the RNN algorithm.

Some parts of the code were assisted by ChatGPT, and those will be labelled for clarity

Part 1A: Statistical Method

SVC algorithm was applied in this part of the task.

Week 4 tutorial materials were referenced for this part of the code.

Code are put into functions for the sake of accessibility in further iterations.

```
# Create empty df to store all evaluation values
metric_results_df = pd.DataFrame()

# Model training, building, and evaluating function

def evaluate_models(tokenizer, df_train, target_field, df_test, metric_results_df):

    # Preprocess and training data
    label_names = ["InformationTheory", "ComputationalLinguistics", "ComputerVision"]

    trainDocs = df_train[target_field].tolist()
    trainLabels = df_train[label_names].to_numpy()

    vectorizer = TfidfVectorizer(analyzer='word', input='content', lowercase=True, min_d
    x_train = vectorizer.fit_transform(trainDocs)

    # Set up SVC model
    model = LinearSVC()
    CV = 5
    entries = []

    for idx, label_name in enumerate(label_names):
        accuracies = cross_val_score(model, x_train, trainLabels[:, idx], scoring='accur
        for fold_idx, accuracy in enumerate(accuracies):
            entries.append((label_name, model.__class__.__name__, fold_idx, accuracy))

    cv_df = pd.DataFrame(entries, columns=['label_name', 'model_name', 'fold_idx', 'accu
    sns.boxplot(x='label_name', y='accuracy', data=cv_df)
    sns.stripplot(x='label_name', y='accuracy', data=cv_df, size=8, jitter=True, edgecol
    plt.show()

    testDocs = df_test[target_field].tolist()
    testLabels = df_test[label_names].to_numpy()
    x_test = vectorizer.transform(testDocs)

     # Create an empty list to store metric results
    metric_results = []

    for idx, label_name in enumerate(label_names):
        print(f"Evaluating for {label_name}")
        y_train, y_test = trainLabels[:, idx], testLabels[:, idx]
        model.fit(x_train, y_train)
```

```
        y_predict = model.predict(x_test)
        y_probs = model.decision_function(x_test)  # Get the decision function scores fo

        # Create a dictionary to store the evaluation metrics for the current label
        label_metric_results = {'label_name': label_name, 'model_name': model.__class__.

        metrics = {
            'Accuracy': accuracy_score,
            'Macro Precision': precision_score,
            'Macro Recall': recall_score,
            'Macro F1 score': f1_score,
            'MCC': matthews_corrcoef
        }
        print(model.__class__.__name__)
        print(confusion_matrix(y_test, y_predict))
        for metric_name, metric_func in metrics.items():
            score = metric_func(y_test, y_predict, average='macro') if 'Macro' in metric
            label_metric_results[metric_name] = score
            print(f'{metric_name}: {score}')


        # Get and store precision-recall curve values
        precision, recall, _ = precision_recall_curve(y_test, y_probs)
        label_metric_results['Recall'] = recall
        label_metric_results['Precision'] = precision

        # Append dictionary to list
        metric_results.append(label_metric_results)

        print('-------\n')
    return metric_results_df.append(metric_results, ignore_index = True)
```

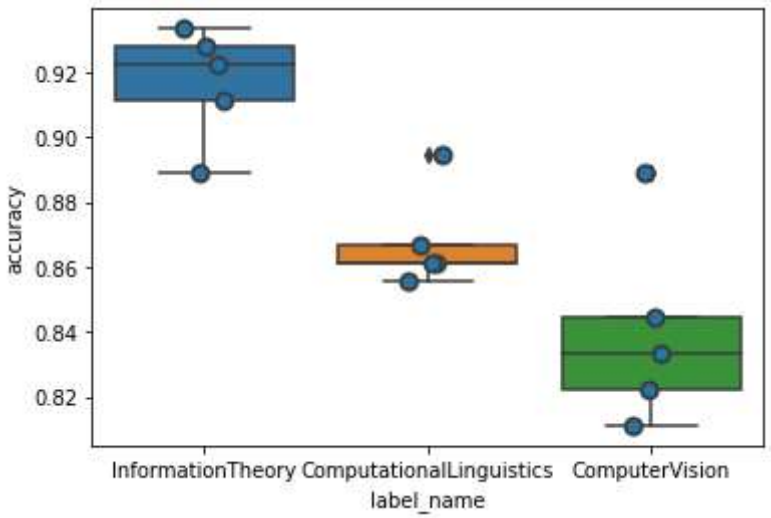
```
# Assited by ChatGPT to convert tokenization methods from the RNN application format to

# Preprocessing method 1 (P1)
# Removing stopwords

def custom_tokenizer(text):

    return [tok.text for tok in spacy_en.tokenizer(text) if tok.text.lower() not in STOP

# Preprocessing method 2 (P2)
# Stemming tokens
def custom_tokenizer_P2(text):

    ps = PorterStemmer()

    return [ps.stem(tok.text) for tok in spacy_en.tokenizer(text)]
```

```
# Splitting data

split_ratio = 0.9
SEED = 1
train_data = pd.read_csv('train.csv')
train_1000 = train_data.head(1000)
train_data, validation = train_test_split(train_data, test_size=1-split_ratio, random_st
train_900, valid_100 = train_test_split(train_1000, test_size=1-split_ratio, random_stat
test_data = pd.read_csv("test.csv")

len(train_data), len(validation), len(train_900), len(valid_100)
```

```
(112500, 12500, 900, 100)
```

P1 Title 1000

```
# Defining parameters

tokenizer = custom_tokenizer
df_train = train_900
target_field = "title"
df_test = test_data
metric_results_df = evaluate_models(tokenizer, df_train, target_field, df_test, metric_r
```

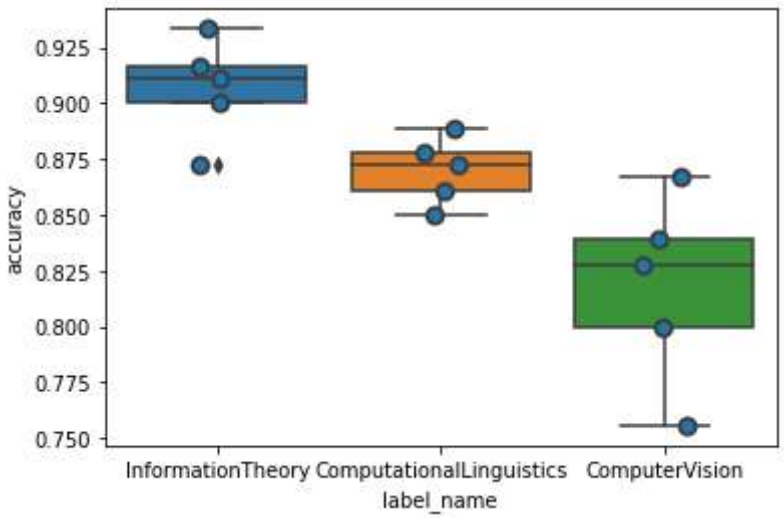



```
Evaluating for InformationTheory
LinearSVC
[[9249  338]
 [2616 5863]]
Accuracy: 0.8364884313074283
Macro Precision: 0.8625061289612967
Macro Recall: 0.8281084875655896
```

P2 Title 1000

```
Evaluating for ComputationalLinguistics
```

```
# Defining parameters

tokenizer = custom_tokenizer_P2
df_train = train_900
target_field = "title"
df_test = test_data
metric_results_df = evaluate_models(tokenizer, df_train, target_field, df_test, metric_r
```

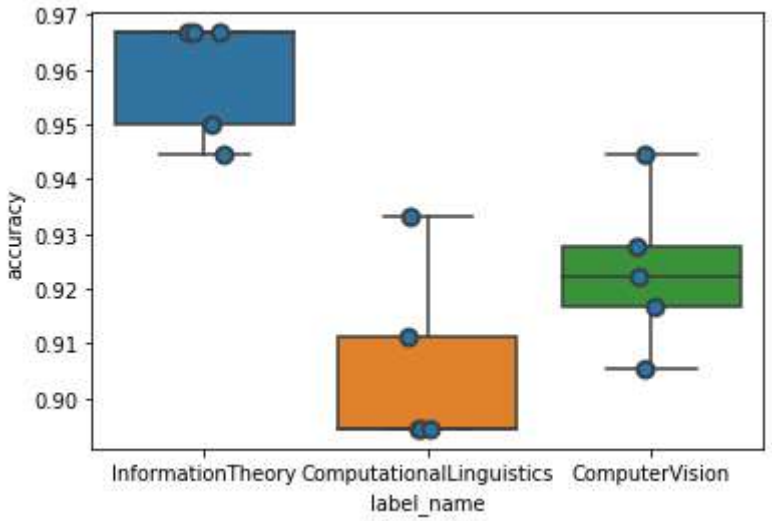



```
Evaluating for InformationTheory
LinearSVC
[[9245  342]
 [2424 6055]]
Accuracy: 0.8468947193623381
Macro Precision: 0.8694037784149105
Macro Recall: 0.8392219616007361
Macro F1 score: 0.8419674781353987
MCC: 0.707982695654716
-------

Evaluating for ComputationalLinguistics
LinearSVC
[[14082   616]
```

P1 Abstract 1000

```
Macro F1 score: 0.808611702132926
```

```
1 tokenizer = custom_tokenizer
2 df_train = train_900
3 target_field = "abstract"
4 df_test = test_data
5 metric_results_df = evaluate_models(tokenizer, df_train, target_field, df_test, metric_r
```

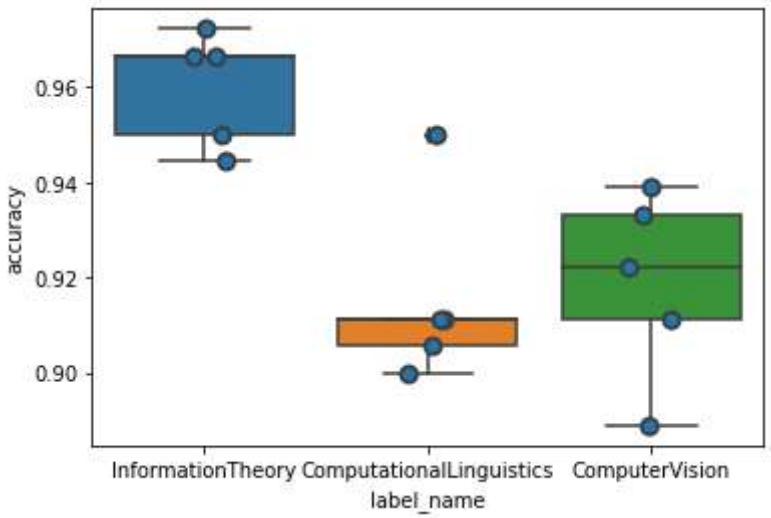



```
Evaluating for InformationTheory
LinearSVC
[[9524   63]
 [1297 7182]]
Accuracy: 0.9247204693900144
Macro Precision: 0.9357224077176827
Macro Recall: 0.9202312247810023
Macro F1 score: 0.9234337400519605
MCC: 0.8558134400901379
-------

Evaluating for ComputationalLinguistics
LinearSVC
[[14418   280]
 [  922  2446]]
Accuracy: 0.9334661795638215
Macro Precision: 0.9185905486880406
Macro Recall: 0.8535984099829053
Macro F1 score: 0.8813704151090274
```

P2 Abstract 1000

```
LinearSVC
```

```
tokenizer = custom_tokenizer_P2
df_train = train_900
target_field = "abstract"
df_test = test_data
metric_results_df = evaluate_models(tokenizer, df_train, target_field, df_test, metric_r
```

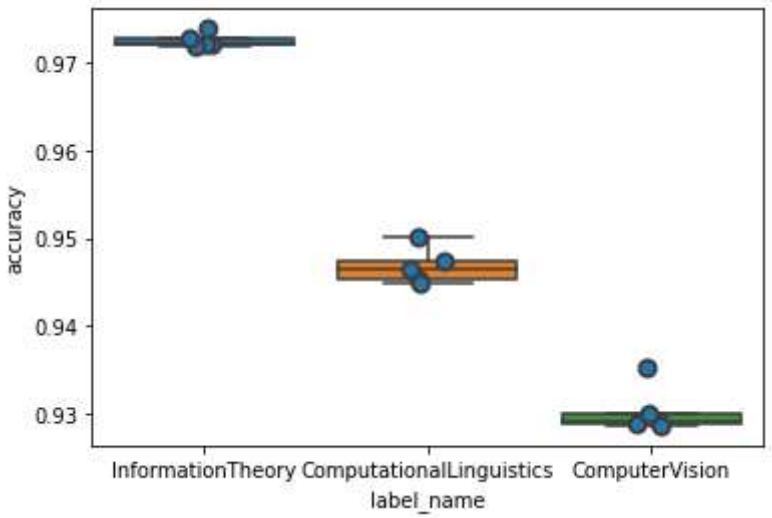



```
Evaluating for InformationTheory
LinearSVC
[[9513   74]
 [1287 7192]]
Accuracy: 0.9246651167939777
Macro Precision: 0.9353244563721442
Macro Recall: 0.920247223418344
Macro F1 score: 0.9234008159951866
MCC: 0.855438820901866
-------

Evaluating for ComputationalLinguistics
LinearSVC
[[14435   263]
 [  975  2393]]
Accuracy: 0.9314734861064984
Macro Precision: 0.9188541560792163
Macro Recall: 0.8463085489356736
Macro F1 score: 0.8766850361275769
MCC: 0.7617158919373275
-------

Evaluating for ComputerVision
LinearSVC
```

P1 Title ALL

```
Macro Recall: 0.927075682063205?
```

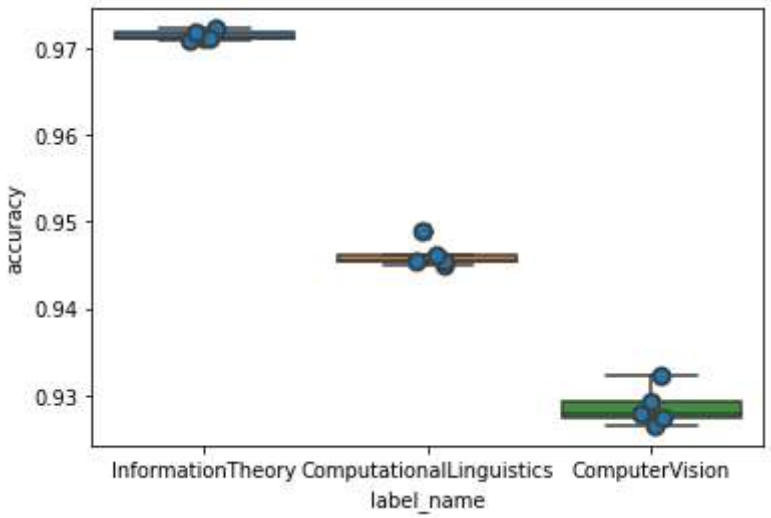
```
1 tokenizer = custom_tokenizer
2 df_train = train_data
3 target_field = "title"
4 df_test = test_data
5 metric_results_df = evaluate_models(tokenizer, df_train, target_field, df_test, metric_r
```




```
Evaluating for InformationTheory
LinearSVC
[[9407  180]
 [ 901 7578]]
Accuracy: 0.9401638436842688
Macro Precision: 0.9446951526397626
Macro Recall: 0.9374810219931994
Macro F1 score: 0.9395442009454607
MCC: 0.8821466768112096
-------

Evaluating for ComputationalLinguistics
LinearSVC
[[14318   380]
 [  514  2854]]
Accuracy: 0.9505147791431419
Macro Precision: 0.9239218267477629
Macro Recall: 0.9107666578644824
Macro F1 score: 0.917156095808995
MCC: 0.8345848116734086
-------

Evaluating for ComputerVision
LinearSVC
[[10957   890]
 [  406  5813]]
Accuracy: 0.9282630355363667
Macro Precision: 0.9157468151649284
Macro Recall: 0.9297958441100067
```

P2 Title ALL

```
1 tokenizer = custom_tokenizer_P2
2 df_train = train_data
3 target_field = "title"
4 df_test = test_data
5 metric_results_df = evaluate_models(tokenizer, df_train, target_field, df_test, metric_r
```

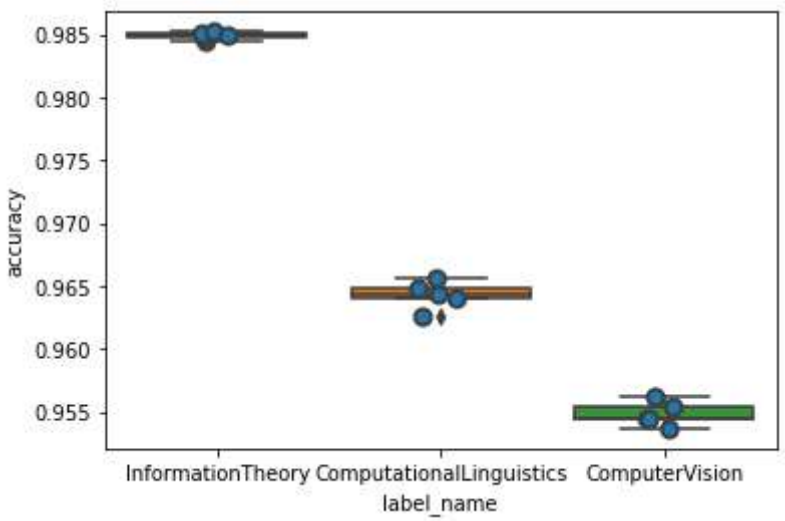



```
Evaluating for InformationTheory
LinearSVC
[[9408  179]
 [ 890 7589]]
Accuracy: 0.9408280748367098
Macro Precision: 0.9452660985835291
Macro Recall: 0.9381818373504347
Macro F1 score: 0.9402220613050182
MCC: 0.883419531564003
-------

Evaluating for ComputationalLinguistics
LinearSVC
[[14302   396]
 [  520  2848]]
Accuracy: 0.9492970220303332
Macro Precision: 0.9214227493007445
Macro Recall: 0.9093316297820668
Macro F1 score: 0.9152170972707914
MCC: 0.8306663850150988
-------

Evaluating for ComputerVision
LinearSVC
[[10981   866]
 [  431  5788]]
Accuracy: 0.9282076829403298
Macro Precision: 0.9160427288188252
Macro Recall: 0.928798789323482
Macro F1 score: 0.9217415622356171
MCC: 0.8447452122962283
-------
```

P1 Abstract ALL

```
1  tokenizer = custom_tokenizer
2  df_train = train_data
```

```
target_field = "abstract"
df_test = test_data
metric_results_df = evaluate_models(tokenizer, df_train, target_field, df_test, metric_r
```




```
Evaluating for InformationTheory
LinearSVC
[[9491   96]
 [ 574 7905]]
Accuracy: 0.9629137606553747
Macro Precision: 0.9654860951620987
Macro Recall: 0.9611448888142682
Macro F1 score: 0.9626257190651313
MCC: 0.9266208147848048
-------

Evaluating for ComputationalLinguistics
LinearSVC
[[14468   230]
 [  314  3054]]
Accuracy: 0.9698881877560057
Macro Precision: 0.9543607040264099
Macro Recall: 0.9455606043319029
Macro F1 score: 0.9498834477913826
MCC: 0.8998782803705756
-------

Evaluating for ComputerVision
LinearSVC
[[11296   551]
 [  228  5991]]
Accuracy: 0.9568803276873685
Macro Precision: 0.9479950977057862
Macro Recall: 0.9584142461829718
Macro F1 score: 0.9528113601579944
MCC: 0.9063494580093591
-------
```

P2 Abstract ALL

```
tokenizer = custom_tokenizer_P2
df_train = train_data
target_field = "abstract"
df_test = test_data
metric_results_df = evaluate_models(tokenizer, df_train, target_field, df_test, metric_r
```


```
# Saving all evaluation metrics into dataframe

metric_results_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24 entries, 0 to 23
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   label_name       24 non-null     object 
 1   model_name       24 non-null     object 
 2   target_field     24 non-null     object 
 3   train_size       24 non-null     int64  
 4   tokenizer        24 non-null     object 
 5   Accuracy         24 non-null     float64
 6   Macro Precision  24 non-null     float64
 7   Macro Recall     24 non-null     float64
 8   Macro F1 score   24 non-null     float64
 9   MCC              24 non-null     float64
 10  Recall           24 non-null     object 
 11  Precision        24 non-null     object 
dtypes: float64(5), int64(1), object(6)
memory usage: 2.4+ KB
```

```
metric_results_df
```

	label_name	model_name	target_field	train_size	tokenizer	Acc
0	InformationTheory	LinearSVC	title	900	custom_tokenizer	0.8
1	ComputationalLinguistics	LinearSVC	title	900	custom_tokenizer	0.8
2	ComputerVision	LinearSVC	title	900	custom_tokenizer	0.8
3	InformationTheory	LinearSVC	title	900	custom_tokenizer_P2	0.8

Libraries imports should have been integrated to the first part of the code.

This part is left here for error handling purpose.

```
# # Import Libraries
# import torch
# import torchtext

# from torchtext.legacy import data
# import torch.nn as nn
# from torchtext.legacy.data import Field, LabelField, TabularDataset, Dataset
# import copy

# import torch.optim as optim
# import collections
# import random
# import time
# import pandas as pd
# import re
# import spacy
# from nltk.stem import PorterStemmer

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

```
# spacy_en = spacy.load("en_core_web_sm")
```

```
# import torch
# import torchtext
# import torch.nn as nn
# import torch.optim as optim

# from torchtext.legacy import data
# from torchtext.legacy.data import Field, LabelField, TabularDataset
# from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score, c
# from sklearn.metrics import precision_recall_curve

# import spacy
# from spacy.lang.en.stop_words import STOP_WORDS
# import numpy as np
# import random
# import time
# import copy
# import collections

# from nltk.stem import PorterStemmer

```

Part 1B: RNN Method

Code was referenced to Week 4 tutorial material on Building RNN models, and converted into functions for the sake of accessibility.

Summarising codes into functions were assisted by ChatGPT.

```
# I accidentally re-run this cell. Hopefully it will not affect the output of the later

spacy_en = spacy.load("en_core_web_sm")
SEED = 1
torch.manual_seed(SEED)
torch.backends.cudnn.deterministic = True
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
device
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
```

```
def load_data(path, train_file, test_file, target_field, tokenizer, MAX_VOCAB_SIZE):
    TEXT = data.Field(sequential=True, tokenize=tokenizer, lower=True)
    LABEL = data.LabelField(dtype=torch.float, use_vocab=False, preprocessing=int)
    if target_field == "title":
        train_datafield = [
            (target_field, TEXT),
            ("abstract", None),
            ("InformationTheory", LABEL),
            ("ComputationalLinguistics", LABEL),
            ("ComputerVision", LABEL)
        ]
    else:
        train_datafield = [
            ("title", None),
            (target_field, TEXT),
            ("InformationTheory", LABEL),
            ("ComputationalLinguistics", LABEL),
            ("ComputerVision", LABEL)
        ]

    train_data, test_data = TabularDataset.splits(
        path=path,
        train=train_file, test=test_file, format="csv",
        skip_header=True, fields=train_datafield
    )
    train_data, valid_data = train_data.split(split_ratio=0.9, random_state=random.getst

    MAX_VOCAB_SIZE = MAX_VOCAB_SIZE

    TEXT.build_vocab(train_data, max_size=MAX_VOCAB_SIZE)

    return TEXT, LABEL, train_data, valid_data, test_data
```

```
def generate_label_iterator(train_data, valid_data, test_data, label, target_field):

    label_attr = f"label_{label}"
    BATCH_SIZE = 64
    iterators = data.BucketIterator.splits(
        (train_data, valid_data, test_data),
        batch_size = BATCH_SIZE,
        device = device,
        sort_key = lambda x: len(getattr(x, target_field)),
        sort_within_batch = False)
    return iterators[0], iterators[1], iterators[2]
```

```

class RNN(nn.Module):
```

```
    def __init__(self, input_dim, embedding_dim, hidden_dim, output_dim, pos_weight = No

        super(RNN, self).__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)

        self.pos_weight = pos_weight

        if self.pos_weight is not None:
            self.criterion = nn.BCEWithLogitsLoss(pos_weight=self.pos_weight)
        else:
            self.criterion = nn.BCEWithLogitsLoss()

    def forward(self, text):

        embedded = self.embedding(text)

        output, hidden = self.rnn(embedded)

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

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

```
#·Assisted·by·ChatGPT

def get_pos_weight(train_data):
    pos_weights = []
    for field in ["InformationTheory", "ComputationalLinguistics", "ComputerVision"]:
        negative_count, positive_count = 0, 0
        for example in train_data.examples:
            if getattr(example, field) == 0:
                negative_count += 1
            else:
                positive_count += 1

        pos_weights.append(torch.tensor([negative_count / positive_count]).to(device))
    return pos_weights

```

```
def generate_model_and_optimizer(pos_weight, embedding_dim=100, hidden_dim=256, output_d
    INPUT_DIM = len(TEXT.vocab)

    model = RNN(INPUT_DIM, embedding_dim, hidden_dim, output_dim, pos_weight)

    optimizer = optim.SGD(model.parameters(), lr=lr)

```

```
    model = model.to(device)

    return model, optimizer


# Evaluation functions

def binary_accuracy(preds, y):

    #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

def train(model, iterator, optimizer, criterion, label_field, target_field):

    epoch_loss = 0
    epoch_acc = 0

    model.train()

    for batch in iterator:

        optimizer.zero_grad()

        predictions = model(getattr(batch, target_field)).squeeze(1)

        # Use the specific label field
        loss = criterion(predictions, getattr(batch, label_field))

        acc = binary_accuracy(predictions, getattr(batch, label_field))

        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, label_field, target_field):

    epoch_loss = 0
    epoch_acc = 0

    model.eval()

    with torch.no_grad():

```

```
        for batch in iterator:

            predictions = model(getattr(batch, target_field)).squeeze(1)

            # Use the specific label field
            loss = criterion(predictions, getattr(batch, label_field))

            acc = binary_accuracy(predictions, getattr(batch, label_field))

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

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

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
```

```
# Assisted by ChatGPT to build up the train_loop, creating early stopping and error hand

rnn_results_df = pd.DataFrame()

def train_loop(ModelsList, OptimizersList, IteratorsList, N_EPOCHS, label_names, target_
    result = []
    best_model_states = []
    best_epochs = [0, 0, 0]
    for idx, (label_name, model, optimizer, iterators_) in enumerate(zip(label_names, Mo
        print(f"Training model for {preprocess}_{label_name}, using {train_size} data of

        bad_epochs = 0
        best_model_state = None
        best_epoch = 0

        for epoch in range(N_EPOCHS):
            start_time = time.time()

            train_iterator = iterators_[0]
            test_iterator = iterators_[2] if len(iterators_) > 2 else iterators_[1]
            valid_iterator = iterators_[1] if len(iterators_) > 2 else None

            train_loss, train_acc = train(model, train_iterator, optimizer, model.criter

            if valid_iterator:
                valid_loss, valid_acc = evaluate(model, valid_iterator, model.criterion,

            test_loss, test_acc = evaluate(model, test_iterator, model.criterion, label_

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

```
32
33             if test_loss < bestLossesList[idx]:
34                 bestLossesList[idx] = test_loss
35                 best_model_state = copy.deepcopy(model.state_dict())
36 #                 best_model_state = model.state_dict()
37                 bad_epochs = 0
38                 best_epochs[idx] = epoch + 1
39
40                 print(f'Epoch: {epoch+1:02} | Epoch Time: {epoch_mins}m {epoch_secs}s')
41                 print(f'{label_name} Train Loss: {train_loss:.3f} | Train Acc: {train_ac
42                 print(f'{label_name} Valid Loss: {valid_loss:.3f} | Valid Acc: {valid_ac
43                 print(f'{label_name} Test Loss: {test_loss:.3f} | Test Acc: {test_acc*10
44             else:
45                 bad_epochs += 1
46
47             if bad_epochs > patience:
48                 print(f"Early stopping at epoch {epoch+1} for {label_name} model.")
49 #                 model.load_state_dict(best_model_state)
50                 break
51
52             # For predictions and ground truth collection
53             model.eval()
54             y_predict, y_test = [], []
55             with torch.no_grad():
56                 for batch in test_iterator:
57                     predictions = model(getattr(batch, target_field)).squeeze(1)
58                     rounded_preds = torch.round(torch.sigmoid(predictions))
59                     y_predict += rounded_preds.tolist()
60                     y_test += getattr(batch, label_name).tolist()
61         print(f"Best performing epoch for {label_name} model: {best_epochs[idx]}")
62         best_model_states.append(best_model_state)
63
64         model.load_state_dict(best_model_state)
65         # Call 'evaluate_model' and store its return value
66         rnn_eval_metrics = evaluate_model(model, test_iterator, label_name, target_field
67 #         print(rnn_eval_metrics)
68         # Append the evaluation metrics to the 'rnn_metric_results' list
69         rnn_metric_results.append({
70             "accuracy": rnn_eval_metrics["accuracy"],
71             'label_name': label_name,
72             'model_name': model.__class__.__name__,
73             'target_field': target_field,
74             'train_size': train_size,
75             'tokenizer': preprocess,
76             **rnn_eval_metrics
77         })
78 #         result.append((y_predict, y_test))
79
80     return rnn_metric_results, best_model_states
81
82
```

```
# Assisted by ChatGPT to return evaluatiton matrics to df


def evaluate_model(model, iterator, label_field, target_field):


    y_probs = []
    y_predict = []
    y_test = []

    model.eval()
    with torch.no_grad():
        for batch in iterator:
            predictions = model(getattr(batch, target_field)).squeeze(1)
            probs = torch.sigmoid(predictions)
            rounded_preds = torch.round(torch.sigmoid(predictions))

            y_predict += rounded_preds.tolist()
            y_test += getattr(batch, label_field).tolist()
            y_probs += probs.tolist()

    y_predict = np.asarray(y_predict)
    y_test = np.asarray(y_test)
    y_probs = np.asarray(y_probs)
    # Compute metrics
    recall_macro = recall_score(y_test, y_predict, average='macro')
    precision_macro = precision_score(y_test, y_predict, average='macro')
#     recall = recall_score(y_test, y_predict, average=None)
#     precision = precision_score(y_test, y_predict, average=None)
    f1score = f1_score(y_test, y_predict, average='macro')
    accuracy = accuracy_score(y_test, y_predict)
    matthews = matthews_corrcoef(y_test, y_predict)
#     precision_curve, recall_curve, _ = precision_recall_curve(y_test, y_probs)
    precision, recall, _ = precision_recall_curve(y_test, y_probs)

    # Print metrics

    print(f"{label_field}:")
    print(confusion_matrix(y_test, y_predict))
    print('Accuracy:', accuracy)
    print('Macro Precision:', precision_macro)
    print('Macro Recall:', recall_macro)
    print('Macro F1 score:', f1score)
    print('MCC:', matthews)
    print("\n")
    return {
        "accuracy": accuracy,
        'Macro Recall': recall_macro,
        'Macro Precision': precision_macro,
        'Macro F1 score': f1score,
```

```
        'MCC': matthews,
        'Recall': recall,
        'Precision': precision
    }
```

```
```

P1 Title 1000

```
def custom_tokenizer(text):
    return [tok.text for tok in spacy_en.tokenizer(text) if tok.text.lower() not in STOP

path = './'
train_file = "train_1000.csv"
test_file = "test.csv"
target_field = "title"
tokenizer = custom_tokenizer
MAX_VOCAB_SIZE = 5000

TEXT, LABEL, train_data, valid_data, test_data = load_data(path, train_file, test_file,


train_iterator_IT, validation_IT, test_iterator_IT = generate_label_iterator(train_data,
train_iterator_CL, validation_CL, test_iterator_CL = generate_label_iterator(train_data,
train_iterator_CV, validation_CV, test_iterator_CV = generate_label_iterator(train_data,

pos_weights = get_pos_weight(train_data)

model_IT, optimizer_IT = generate_model_and_optimizer(pos_weight=pos_weights[0])
model_CL, optimizer_CL = generate_model_and_optimizer(pos_weight=pos_weights[1])
model_CV, optimizer_CV = generate_model_and_optimizer(pos_weight=pos_weights[2])

N_EPOCHS = 2
label_names = ["InformationTheory", "ComputationalLinguistics", "ComputerVision"]
models = [model_IT, model_CL, model_CV]
optimizers = [optimizer_IT, optimizer_CL, optimizer_CV]
iterators = [(train_iterator_IT, validation_IT, test_iterator_IT), (train_iterator_CL, v
model_file_names = ["RNN_model_IT_T_P1_1000", "RNN_model_CL_T_P1_1000.pt", "RNN_model_CV
best_valid_losses = [float("inf"), float("inf"), float("inf")]
```

```
  # Making sure data are loaded correctly

  # print(TEXT.vocab.freqs.most_common(200))
  len(train_data), len(valid_data), len(test_data), len(TEXT.vocab)

```

```
    (900, 100, 18066, 2497)
```

```
# Run model and store output

rnn_results, best_model_states = train_loop(models, optimizers, iterators, N_EPOCHS, lab
rnn_results_df = rnn_results_df.append(pd.DataFrame(rnn_results), ignore_index = True)

```

```
Training model for P1_InformationTheory, using 900 data of title...
Epoch: 01 | Epoch Time: 0m 2s
InformationTheory Train Loss: 1.057 | Train Acc: 40.73%
InformationTheory Valid Loss: 0.978 | Valid Acc: 22.05%
InformationTheory Test Loss: 1.323 | Test Acc: 51.96%
Best performing epoch for InformationTheory model: 1
InformationTheory:
[[6510 3077]
 [5607 2872]]
Accuracy: 0.5193180560168272
Macro Precision: 0.5100159559607327
Macro Recall: 0.5088818640320529
Macro F1 score: 0.4990020990638365
MCC: 0.01886375985791359


Training model for P1_ComputationalLinguistics, using 900 data of title...
Epoch: 01 | Epoch Time: 0m 0s
ComputationalLinguistics Train Loss: 1.085 | Train Acc: 49.38%
ComputationalLinguistics Valid Loss: 1.272 | Valid Acc: 67.36%
ComputationalLinguistics Test Loss: 1.002 | Test Acc: 54.41%
Best performing epoch for ComputationalLinguistics model: 1
ComputationalLinguistics:
[[8274 6424]
 [1828 1540]]
Accuracy: 0.5432303775047049
Macro Precision: 0.5062079496319858
Macro Recall: 0.5100891940312787
Macro F1 score: 0.46952737323935645
MCC: 0.015828229006873723


Training model for P1_ComputerVision, using 900 data of title...
Epoch: 01 | Epoch Time: 0m 0s
ComputerVision Train Loss: 0.678 | Train Acc: 48.44%
ComputerVision Valid Loss: 0.678 | Valid Acc: 49.05%
ComputerVision Test Loss: 0.693 | Test Acc: 50.67%
Best performing epoch for ComputerVision model: 1
ComputerVision:
[[5748 6099]
 [2802 3417]]
Accuracy: 0.5073065426768515
Macro Precision: 0.5156800734497025
Macro Recall: 0.5173156857506777
Macro F1 score: 0.4989653472003046
MCC: 0.032955195305298446
```

```
rnn_results_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   accuracy         3 non-null      float64
 1   label_name       3 non-null      object 
 2   model_name       3 non-null      object 
 3   target_field     3 non-null      object 
 4   train_size       3 non-null      int64  
 5   tokenizer        3 non-null      object 
 6   Macro Recall     3 non-null      float64
 7   Macro Precision  3 non-null      float64
 8   Macro F1 score   3 non-null      float64
 9   MCC              3 non-null      float64
 10  Recall           3 non-null      object 
 11  Precision        3 non-null      object 
dtypes: float64(5), int64(1), object(6)
memory usage: 416.0+ bytes
```

P2 Title 1000

```
def custom_tokenizer_P2(text):
    ps = PorterStemmer()
    return [ps.stem(tok.text) for tok in spacy_en.tokenizer(text)]

path = './'
train_file = "train_1000.csv"
test_file = "test.csv"
target_field = "title"
tokenizer = custom_tokenizer_P2
MAX_VOCAB_SIZE = 5000

TEXT, LABEL, train_data, valid_data, test_data = load_data(path, train_file, test_file,


train_iterator_IT, validation_IT, test_iterator_IT = generate_label_iterator(train_data,
train_iterator_CL, validation_CL, test_iterator_CL = generate_label_iterator(train_data,
train_iterator_CV, validation_CV, test_iterator_CV = generate_label_iterator(train_data,

pos_weights = get_pos_weight(train_data)

model_IT, optimizer_IT = generate_model_and_optimizer(pos_weight=pos_weights[0])
model_CL, optimizer_CL = generate_model_and_optimizer(pos_weight=pos_weights[1])
model_CV, optimizer_CV = generate_model_and_optimizer(pos_weight=pos_weights[2])
```

```
24
25 N_EPOCHS = 30
26 label_names = ["InformationTheory", "ComputationalLinguistics", "ComputerVision"]
27 models = [model_IT, model_CL, model_CV]
28 optimizers = [optimizer_IT, optimizer_CL, optimizer_CV]
29 iterators = [(train_iterator_IT, validation_IT, test_iterator_IT), (train_iterator_CL, v
30 model_file_names = ["RNN_model_IT_T_P2_1000", "RNN_model_CL_T_P2_1000.pt", "RNN_model_CV
31 best_valid_losses = [float("inf"), float("inf"), float("inf")]
```

```
1 print(TEXT.vocab.freqs.most_common(200))
2 len(train_data), len(valid_data), len(test_data), len(TEXT.vocab)
3
4
```

```
[('-', 602), ('\n  ', 444), ('for', 353), ('of', 247), (':', 221), ('and', 202), ('a',
(900, 100, 18066, 2072)
```

```
1 rnn_results, best_model_states = train_loop(models, optimizers, iterators, N_EPOCHS, lab
2 rnn_results_df = rnn_results_df.append(pd.DataFrame(rnn_results), ignore_index = True)
3
```

```
ComputerVision Test Loss: 0.682 | Test Acc: 52.54%
Epoch: 17 | Epoch Time: 0m 0s
ComputerVision Train Loss: 0.680 | Train Acc: 48.44%
ComputerVision Valid Loss: 0.684 | Valid Acc: 51.04%
ComputerVision Test Loss: 0.682 | Test Acc: 52.75%
Epoch: 19 | Epoch Time: 0m 0s
ComputerVision Train Loss: 0.677 | Train Acc: 50.00%
ComputerVision Valid Loss: 0.684 | Valid Acc: 51.04%
ComputerVision Test Loss: 0.682 | Test Acc: 52.52%
Epoch: 21 | Epoch Time: 0m 0s
ComputerVision Train Loss: 0.677 | Train Acc: 52.19%
ComputerVision Valid Loss: 0.686 | Valid Acc: 35.50%
ComputerVision Test Loss: 0.682 | Test Acc: 52.27%
Epoch: 25 | Epoch Time: 0m 0s
ComputerVision Train Loss: 0.673 | Train Acc: 48.54%
ComputerVision Valid Loss: 0.680 | Valid Acc: 51.82%
ComputerVision Test Loss: 0.678 | Test Acc: 53.37%
Epoch: 27 | Epoch Time: 0m 0s
ComputerVision Train Loss: 0.677 | Train Acc: 52.60%
ComputerVision Valid Loss: 0.685 | Valid Acc: 51.82%
ComputerVision Test Loss: 0.678 | Test Acc: 53.41%
Best performing epoch for ComputerVision model: 27
ComputerVision:
[[7128 4719]
 [3706 2513]]
Accuracy: 0.5336543783903465
Macro Precision: 0.50270607496312
```

P1 Title ALL

```
path = './'
train_file = "train.csv"
test_file = "test.csv"
target_field = "title"
tokenizer = custom_tokenizer
MAX_VOCAB_SIZE = 5000

TEXT, LABEL, train_data, valid_data, test_data = load_data(path, train_file, test_file,


train_iterator_IT, validation_IT, test_iterator_IT = generate_label_iterator(train_data,
train_iterator_CL, validation_CL, test_iterator_CL = generate_label_iterator(train_data,
train_iterator_CV, validation_CV, test_iterator_CV = generate_label_iterator(train_data,

pos_weights = get_pos_weight(train_data)

model_IT, optimizer_IT = generate_model_and_optimizer(pos_weight=pos_weights[0])
model_CL, optimizer_CL = generate_model_and_optimizer(pos_weight=pos_weights[1])
model_CV, optimizer_CV = generate_model_and_optimizer(pos_weight=pos_weights[2])

N_EPOCHS = 30
```

```
label_names = ["InformationTheory", "ComputationalLinguistics", "ComputerVision"]
models = [model_IT, model_CL, model_CV]
optimizers = [optimizer_IT, optimizer_CL, optimizer_CV]
iterators = [(train_iterator_IT, validation_IT, test_iterator_IT), (train_iterator_CL, v
model_file_names = ["RNN_model_IT_T_P1_ALL", "RNN_model_CL_T_P1_ALL.pt", "RNN_model_CV_T
best valid losses = [float("inf"), float("inf"), float("inf")]
```

```
print(TEXT.vocab.freqs.most_common(50))
len(train_data), len(valid_data), len(test_data), len(TEXT.vocab)
```

```
[('-', 75468), ('\n  ', 55134), (':', 27878), ('learning', 15463), ('based', 10302), ('
(112500, 12500, 18066, 5002)
```

```
rnn_results, best_model_states = train_loop(models, optimizers, iterators, N_EPOCHS, lab
rnn_results_df = rnn_results_df.append(pd.DataFrame(rnn_results), ignore_index = True)

```

```
Epoch: 13 | Epoch Time: 0m 7s
ComputerVision Train Loss: 0.242 | Train Acc: 89.64%
ComputerVision Valid Loss: 0.318 | Valid Acc: 86.01%
ComputerVision Test Loss: 0.345 | Test Acc: 84.18%
Epoch: 14 | Epoch Time: 0m 7s
ComputerVision Train Loss: 0.234 | Train Acc: 90.07%
ComputerVision Valid Loss: 0.314 | Valid Acc: 86.07%
ComputerVision Test Loss: 0.336 | Test Acc: 84.60%
Epoch: 15 | Epoch Time: 0m 7s
ComputerVision Train Loss: 0.222 | Train Acc: 90.65%
ComputerVision Valid Loss: 0.304 | Valid Acc: 86.82%
ComputerVision Test Loss: 0.330 | Test Acc: 85.10%
Epoch: 18 | Epoch Time: 0m 7s
ComputerVision Train Loss: 0.202 | Train Acc: 91.54%
ComputerVision Valid Loss: 0.295 | Valid Acc: 87.06%
ComputerVision Test Loss: 0.305 | Test Acc: 86.31%
Epoch: 26 | Epoch Time: 0m 7s
ComputerVision Train Loss: 0.164 | Train Acc: 93.27%
ComputerVision Valid Loss: 0.275 | Valid Acc: 87.83%
ComputerVision Test Loss: 0.298 | Test Acc: 86.78%
Best performing epoch for ComputerVision model: 26
ComputerVision:
```

P2 Title ALL

```
path = './'
train_file = "train.csv"
test_file = "test.csv"
target_field = "title"
tokenizer = custom_tokenizer_P2
MAX_VOCAB_SIZE = 5000

TEXT, LABEL, train_data, valid_data, test_data = load_data(path, train_file, test_file,


train_iterator_IT, validation_IT, test_iterator_IT = generate_label_iterator(train_data,
train_iterator_CL, validation_CL, test_iterator_CL = generate_label_iterator(train_data,
train_iterator_CV, validation_CV, test_iterator_CV = generate_label_iterator(train_data,

pos_weights = get_pos_weight(train_data)

model_IT, optimizer_IT = generate_model_and_optimizer(pos_weight=pos_weights[0])
model_CL, optimizer_CL = generate_model_and_optimizer(pos_weight=pos_weights[1])
model_CV, optimizer_CV = generate_model_and_optimizer(pos_weight=pos_weights[2])

N_EPOCHS = 30
label_names = ["InformationTheory", "ComputationalLinguistics", "ComputerVision"]
models = [model_IT, model_CL, model_CV]
optimizers = [optimizer_IT, optimizer_CL, optimizer_CV]
iterators = [(train_iterator_IT, validation_IT, test_iterator_IT), (train_iterator_CL, v
```

```
model_file_names = ["RNN_model_IT_T_P2_ALL", "RNN_model_CL_T_P2_ALL.pt", "RNN_model_CV_T
```

```
print(TEXT.vocab.freqs.most_common(50))
len(train_data), len(valid_data), len(test_data), len(TEXT.vocab)
```

```
[('-', 75468), ('\n  ', 55134), ('for', 44553), ('of', 28257), (':', 27878), ('and', 27
(112500, 12500, 18066, 5002)
```

```
rnn_results, best_model_states = train_loop(models, optimizers, iterators, N_EPOCHS, lab
rnn_results_df = rnn_results_df.append(pd.DataFrame(rnn_results), ignore_index = True)

```

```
ComputerVision Test Loss: 0.299 | Test Acc: 87.24%
Epoch: 24 | Epoch Time: 0m 8s
ComputerVision Train Loss: 0.179 | Train Acc: 92.69%
ComputerVision Valid Loss: 0.272 | Valid Acc: 88.37%
ComputerVision Test Loss: 0.294 | Test Acc: 87.40%
Epoch: 25 | Epoch Time: 0m 8s
ComputerVision Train Loss: 0.177 | Train Acc: 92.85%
ComputerVision Valid Loss: 0.265 | Valid Acc: 88.32%
ComputerVision Test Loss: 0.281 | Test Acc: 88.20%
Epoch: 26 | Epoch Time: 0m 8s
ComputerVision Train Loss: 0.170 | Train Acc: 93.10%
ComputerVision Valid Loss: 0.267 | Valid Acc: 88.52%
ComputerVision Test Loss: 0.273 | Test Acc: 88.56%
Best performing epoch for ComputerVision model: 26
ComputerVision:
[[10595  1252]
 [  818  5401]]
```

```
1
```

P1 Abstract 1000

```
path = './'
train_file = "train_1000.csv"
test_file = "test.csv"
target_field = "abstract"
tokenizer = custom_tokenizer
MAX_VOCAB_SIZE = 10000

TEXT, LABEL, train_data, valid_data, test_data = load_data(path, train_file, test_file,


train_iterator_IT, validation_IT, test_iterator_IT = generate_label_iterator(train_data,
train_iterator_CL, validation_CL, test_iterator_CL = generate_label_iterator(train_data,
train_iterator_CV, validation_CV, test_iterator_CV = generate_label_iterator(train_data,

pos_weights = get_pos_weight(train_data)

model_IT, optimizer_IT = generate_model_and_optimizer(pos_weight=pos_weights[0])
model_CL, optimizer_CL = generate_model_and_optimizer(pos_weight=pos_weights[1])
model_CV, optimizer_CV = generate_model_and_optimizer(pos_weight=pos_weights[2])

N_EPOCHS = 30
label_names = ["InformationTheory", "ComputationalLinguistics", "ComputerVision"]
models = [model_IT, model_CL, model_CV]
optimizers = [optimizer_IT, optimizer_CL, optimizer_CV]
iterators = [(train_iterator_IT, validation_IT, test_iterator_IT), (train_iterator_CL, v
model_file_names = ["RNN_model_IT_A_P1_1000", "RNN_model_CL_A_P1_1000.pt", "RNN_model_CV
best_valid_losses = [float("inf"), float("inf"), float("inf")]
```

```
print(TEXT.vocab.freqs.most_common(50))
len(train_data), len(valid_data), len(test_data), len(TEXT.vocab)
```

```
[('\n', 12514), ('.', 6196), ('-', 4941), (')', 1610), ('(', 1537), ('  ', 900), ('mode
(900, 100, 18066, 10002)
```

```
rnn_results, best_model_states = train_loop(models, optimizers, iterators, N_EPOCHS, lab
rnn_results_df = rnn_results_df.append(pd.DataFrame(rnn_results), ignore_index = True)

```

```
Accuracy: 0.4754234473596812
Macro Precision: 0.5606306227183617
Macro Recall: 0.5048577669472286
Macro F1 score: 0.3429903488800032
MCC: 0.03432371978857447


Training model for P1_ComputationalLinguistics, using 900 data of abstract...
Epoch: 01 | Epoch Time: 0m 2s
ComputationalLinguistics Train Loss: 1.078 | Train Acc: 58.54%
ComputationalLinguistics Valid Loss: 1.229 | Valid Acc: 34.64%
ComputationalLinguistics Test Loss: 1.007 | Test Acc: 59.05%
Early stopping at epoch 12 for ComputationalLinguistics model.
Best performing epoch for ComputationalLinguistics model: 1
ComputationalLinguistics:
[[9462 5236]
 [2143 1225]]
Accuracy: 0.5915531938447913
```

```
ComputerVision Test Loss: 0.649 | Test Acc: 65.49%
Epoch: 25 | Epoch Time: 0m 2s
ComputerVision Train Loss: 0.678 | Train Acc: 46.77%
ComputerVision Valid Loss: 0.679 | Valid Acc: 51.30%
ComputerVision Test Loss: 0.641 | Test Acc: 65.53%
Best performing epoch for ComputerVision model: 25
ComputerVision:
[[11820    27]
 [ 6204    15]]
Accuracy: 0.655097974094985
Macro Precision: 0.5064675670534525
Macro Recall: 0.5000664526743964
Macro F1 score: 0.39809729994221926
MCC: 0.0013111630372156966
```

```
1
```

P2 Abstract 1000

```
path = './'
train_file = "train_1000.csv"
test_file = "test.csv"
target_field = "abstract"
tokenizer = custom_tokenizer_P2
MAX_VOCAB_SIZE = 10000

TEXT, LABEL, train_data, valid_data, test_data = load_data(path, train_file, test_file,


train_iterator_IT, validation_IT, test_iterator_IT = generate_label_iterator(train_data,
train_iterator_CL, validation_CL, test_iterator_CL = generate_label_iterator(train_data,
train_iterator_CV, validation_CV, test_iterator_CV = generate_label_iterator(train_data,

pos_weights = get_pos_weight(train_data)

model_IT, optimizer_IT = generate_model_and_optimizer(pos_weight=pos_weights[0])
model_CL, optimizer_CL = generate_model_and_optimizer(pos_weight=pos_weights[1])
model_CV, optimizer_CV = generate_model_and_optimizer(pos_weight=pos_weights[2])

N_EPOCHS = 30
label_names = ["InformationTheory", "ComputationalLinguistics", "ComputerVision"]
models = [model_IT, model_CL, model_CV]
optimizers = [optimizer_IT, optimizer_CL, optimizer_CV]
iterators = [(train_iterator_IT, validation_IT, test_iterator_IT), (train_iterator_CL, v
model_file_names = ["RNN_model_IT_A_P2_1000", "RNN_model_CL_A_P2_1000.pt", "RNN_model_CV
best_valid_losses = [float("inf"), float("inf"), float("inf")]
```

```
print(TEXT.vocab.freqs.most_common(50))
len(train_data), len(valid_data), len(test_data), len(TEXT.vocab)
```

```
[('\n', 12514), ('the', 8593), ('.', 6196), ('of', 4993), ('-', 4941), ('and', 4018), (
(900, 100, 18066, 7054)
```

```
rnn_results, best_model_states = train_loop(models, optimizers, iterators, N_EPOCHS, lab
rnn_results_df = rnn_results_df.append(pd.DataFrame(rnn_results), ignore_index = True)

```

```
Training model for P2_ComputationalLinguistics, using 900 data of abstract...
Epoch: 01 | Epoch Time: 0m 3s
ComputationalLinguistics Train Loss: 1.080 | Train Acc: 34.38%
ComputationalLinguistics Valid Loss: 1.228 | Valid Acc: 32.47%
ComputationalLinguistics Test Loss: 0.999 | Test Acc: 58.32%
Epoch: 02 | Epoch Time: 0m 3s
ComputationalLinguistics Train Loss: 1.109 | Train Acc: 33.54%
```

```
ComputerVision Valid Loss: 0.677 | Valid Acc: 51.91%
ComputerVision Test Loss: 0.667 | Test Acc: 63.32%
Epoch: 25 | Epoch Time: 0m 3s
ComputerVision Train Loss: 0.678 | Train Acc: 46.98%
ComputerVision Valid Loss: 0.684 | Valid Acc: 51.91%
ComputerVision Test Loss: 0.654 | Test Acc: 64.88%
Best performing epoch for ComputerVision model: 25
ComputerVision:
[[11645   202]
 [ 6154    65]]
Accuracy: 0.6481788995903908
Macro Precision: 0.44884796583067726
Macro Recall: 0.49670055549468134
Macro F1 score: 0.4028233119980119
```

P1 Abstract ALL

```
path = './'
train_file = "train.csv"
test_file = "test.csv"
target_field = "abstract"
tokenizer = custom_tokenizer
MAX_VOCAB_SIZE = 10000

TEXT, LABEL, train_data, valid_data, test_data = load_data(path, train_file, test_file,


train_iterator_IT, validation_IT, test_iterator_IT = generate_label_iterator(train_data,
train_iterator_CL, validation_CL, test_iterator_CL = generate_label_iterator(train_data,
train_iterator_CV, validation_CV, test_iterator_CV = generate_label_iterator(train_data,

pos_weights = get_pos_weight(train_data)

model_IT, optimizer_IT = generate_model_and_optimizer(pos_weight=pos_weights[0])
model_CL, optimizer_CL = generate_model_and_optimizer(pos_weight=pos_weights[1])
model_CV, optimizer_CV = generate_model_and_optimizer(pos_weight=pos_weights[2])

N_EPOCHS = 30
label_names = ["InformationTheory", "ComputationalLinguistics", "ComputerVision"]
models = [model_IT, model_CL, model_CV]
optimizers = [optimizer_IT, optimizer_CL, optimizer_CV]
iterators = [(train_iterator_IT, validation_IT, test_iterator_IT), (train_iterator_CL, v
model_file_names = ["RNN_model_IT_A_P2_1000", "RNN_model_CL_A_P2_1000.pt", "RNN_model_CV
best_valid_losses = [float("inf"), float("inf"), float("inf")]
```

```
# print(TEXT.vocab.freqs.most_common(50))
len(train_data), len(valid_data), len(test_data), len(TEXT.vocab)
```

```
(112500, 12500, 18066, 10002)
```

```
rnn_results, best_model_states = train_loop(models, optimizers, iterators, N_EPOCHS, lab
rnn_results_df = rnn_results_df.append(pd.DataFrame(rnn_results), ignore_index = True)

```

```
Best performing epoch for ComputerVision model: 24
ComputerVision:
[[  719 11128]
 [  446  5773]]
Accuracy: 0.35934905347060775
Macro Precision: 0.4793724017143004
Macro Recall: 0.4944873801200065
```

P2 Abstract ALL

```
path = './'
train_file = "train.csv"
test_file = "test.csv"
target_field = "abstract"
tokenizer = custom_tokenizer_P2
MAX_VOCAB_SIZE = 10000

TEXT, LABEL, train_data, valid_data, test_data = load_data(path, train_file, test_file,

train_iterator_IT, validation_IT, test_iterator_IT = generate_label_iterator(train_data,
train_iterator_CL, validation_CL, test_iterator_CL = generate_label_iterator(train_data,
train_iterator_CV, validation_CV, test_iterator_CV = generate_label_iterator(train_data,

pos_weights = get_pos_weight(train_data)

model_IT, optimizer_IT = generate_model_and_optimizer(pos_weight=pos_weights[0])
model_CL, optimizer_CL = generate_model_and_optimizer(pos_weight=pos_weights[1])
model_CV, optimizer_CV = generate_model_and_optimizer(pos_weight=pos_weights[2])

N_EPOCHS = 30
label_names = ["InformationTheory", "ComputationalLinguistics", "ComputerVision"]
models = [model_IT, model_CL, model_CV]
optimizers = [optimizer_IT, optimizer_CL, optimizer_CV]
iterators = [(train_iterator_IT, validation_IT, test_iterator_IT), (train_iterator_CL, v
model_file_names = ["RNN_model_IT_A_P2_ALL", "RNN_model_CL_A_P2_ALL.pt", "RNN_model_CV_A
best_valid_losses = [float("inf"), float("inf"), float("inf")]
```

```
# print(TEXT.vocab.freqs.most_common(50))
len(train_data), len(valid_data), len(test_data), len(TEXT.vocab)
```

```
(112500, 12500, 18066, 10002)
```

```
rnn_results, best_model_states = train_loop(models, optimizers, iterators, N_EPOCHS, lab
rnn_results_df = rnn_results_df.append(pd.DataFrame(rnn_results), ignore_index = True)
```

```
ComputationalLinguistics Test Loss: 0.980 | Test Acc: 54.54%
Epoch: 03 | Epoch Time: 0m 58s
ComputationalLinguistics Train Loss: 1.053 | Train Acc: 46.70%
ComputationalLinguistics Valid Loss: 1.112 | Valid Acc: 75.64%
ComputationalLinguistics Test Loss: 0.968 | Test Acc: 80.78%
Early stopping at epoch 9 for ComputationalLinguistics model.
Best performing epoch for ComputationalLinguistics model: 3
ComputationalLinguistics:
[[14573   125]
 [ 3346    22]]
Accuracy: 0.8078711391564264
Macro Precision: 0.4814653468954968
Macro Recall: 0.49901375403249393
Macro F1 score: 0.4530504408879102
MCC: -0.008550959451065022


Training model for P2_ComputerVision, using 112500 data of abstract...
Epoch: 01 | Epoch Time: 0m 58s
ComputerVision Train Loss: 0.666 | Train Acc: 50.03%
ComputerVision Valid Loss: 0.667 | Valid Acc: 50.67%
ComputerVision Test Loss: 0.688 | Test Acc: 43.47%
Epoch: 02 | Epoch Time: 0m 58s
ComputerVision Train Loss: 0.665 | Train Acc: 50.25%
ComputerVision Valid Loss: 0.665 | Valid Acc: 51.70%
ComputerVision Test Loss: 0.684 | Test Acc: 43.22%
Epoch: 04 | Epoch Time: 0m 57s
ComputerVision Train Loss: 0.665 | Train Acc: 50.01%
ComputerVision Valid Loss: 0.664 | Valid Acc: 52.84%
ComputerVision Test Loss: 0.674 | Test Acc: 52.54%
Epoch: 06 | Epoch Time: 0m 59s
ComputerVision Train Loss: 0.665 | Train Acc: 50.11%
ComputerVision Valid Loss: 0.662 | Valid Acc: 54.05%
ComputerVision Test Loss: 0.670 | Test Acc: 52.59%
Epoch: 08 | Epoch Time: 0m 59s
ComputerVision Train Loss: 0.665 | Train Acc: 50.13%
ComputerVision Valid Loss: 0.662 | Valid Acc: 54.44%
ComputerVision Test Loss: 0.668 | Test Acc: 53.50%
Epoch: 14 | Epoch Time: 0m 59s
ComputerVision Train Loss: 0.665 | Train Acc: 50.18%
ComputerVision Valid Loss: 0.660 | Valid Acc: 54.11%
ComputerVision Test Loss: 0.659 | Test Acc: 56.98%
Epoch: 16 | Epoch Time: 0m 58s
ComputerVision Train Loss: 0.665 | Train Acc: 50.07%
ComputerVision Valid Loss: 0.659 | Valid Acc: 53.48%
ComputerVision Test Loss: 0.657 | Test Acc: 59.57%
Early stopping at epoch 22 for ComputerVision model.
Best performing epoch for ComputerVision model: 16
ComputerVision:
[[8012 3835]
 [3480 2739]]
Accuracy: 0.5950957599911436
Macro Precision: 0.5569109808376108
Macro Recall: 0.5583569307513049
Macro F1 score: 0.5573895203512239
```

```
1 rnn_results_df.columns
```

```
Index(['accuracy', 'label_name', 'model_name', 'target_field', 'train_size',
       'tokenizer', 'Macro Recall', 'Macro Precision', 'Macro F1 score', 'MCC',
       'Recall', 'Precision'],
      dtype='object')
```

Storing output

```
1 metric_results_df.to_csv("svc_results.csv", index = False)
```

```
1 rnn_results_df.to_csv("rnn_results.csv", index = False)
```

Part 1C: Results for Methods

F1, precision, etc.

```
1 from google.colab import drive
2 drive.mount('/content/drive')
3
```

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

```
1 import pandas as pd
2 import os
3
4 os.chdir("/content/drive/MyDrive/")
5
6 rnn_results = pd.read_csv("rnn_results.csv")
7 svc_results = pd.read_csv("svc_results.csv")
8
```

```
1 combined_df = pd.concat([rnn_results, svc_results], ignore_index = True)
```

```
1 combined_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48 entries, 0 to 47
Data columns (total 13 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   accuracy         24 non-null     float64
 1   label_name       48 non-null     object
 2   model_name       48 non-null     object
 3   target_field     48 non-null     object
```

```
 4   train_size       48 non-null     int64
 5   tokenizer        48 non-null     object
 6   Macro Recall     48 non-null     float64
 7   Macro Precision  48 non-null     float64
 8   Macro F1 score   48 non-null     float64
 9   MCC              48 non-null     float64
 10  Recall           48 non-null     object
 11  Precision        48 non-null     object
 12  Accuracy         24 non-null     float64
dtypes: float64(6), int64(1), object(6)
memory usage: 5.0+ KB
```

RNN models result

```
rnn_results = rnn_results[['label_name', 'model_name', 'target_field', 'train_size',
                           'tokenizer', 'accuracy','Macro Recall', 'Macro Precision'
                           'Recall', 'Precision']]
rnn_results = rnn_results.rename(columns = {"accuracy": "Accuracy"})
rnn_results
```

	label_name	model_name	target_field	train_size	tokenizer	Accuracy	
0	InformationTheory	RNN	title	900	P1	0.519318	0.5
1	ComputationalLinguistics	RNN	title	900	P1	0.543230	0.5

SVC models result

```
1 svc_results
```

	label_name	model_name	target_field	train_size	tokenizer	Acc
0	InformationTheory	LinearSVC	title	900	custom_tokenizer	0.8
1	ComputationalLinguistics	LinearSVC	title	900	custom_tokenizer	0.8

1

Part 1D: Plots for Methods

F1, precision, etc.

```
%matplotlib inline

import matplotlib.pyplot as plt
from sklearn.metrics import auc


def plot_precision_recall_curve(df):
    plt.figure(figsize=(10, 6))

    for index, row in df.iterrows():
        Model = "RNN" if row["model_name"] == "RNN" else "SVC"
        tag = ''
        target_field = 'T' if row['target_field'] == 'title' else 'A'
        size = str(1000) if row['train_size'] == 900 else 'All'
        tokenizer = 'P1' if row['tokenizer'] == 'P1' else 'P2'
        if row["label_name"]== "InformationTheory":
            tag = "IT"
        elif row["label_name"] == "ComputationalLinguistics":
            tag = "CL"
        else:
            tag = "CV"
        label = f"{Model}{tokenizer}{size}{target_field}_{tag}"
#         plt.plot(row['Recall'], row['Precision'], label=f"{label} (AUC: {auc(row['Reca
        plt.plot(row['Recall'], row['Precision'], label=f"{label}")

    plt.xlabel("Recall")
    plt.ylabel("Precision")
    plt.title("Precision-Recall Curve")
    plt.legend(loc='best')
    plt.show()
```

```
# I accidentally run this part of code again, hopefully it wont affect the following out

rnn_results_df = rnn_results_df[['label_name', 'model_name', 'target_field', 'train_size
                                 'tokenizer', 'accuracy','Macro Recall', 'Macro Precision'
                                 'Recall', 'Precision']]
rnn_results_df = rnn_results_df.rename(columns = {"accuracy": "Accuracy"})

combined_df = pd.concat([rnn_results_df, metrics_results_df], axis = 0, ignore_index = T
```

```
Traceback (most recent call last):

  Cell In[106], line 1
    rnn_results_df = rnn_results_df[['label_name', 'model_name', 'target_field',
'train_size',

  File ~\anaconda3\envs\5212A1\lib\site-packages\pandas\core\frame.py:3813 in
__getitem__
    indexer = self.columns._get_indexer_strict(key, "columns")[1]
```

```
combined_df = pd.concat([rnn_results_df, metric_results_df], axis = 0, ignore_index = Tr
combined_df.info()

combined_df.loc[combined_df["tokenizer"] == "custom_tokenizer", "tokenizer"] = "P1"
combined_df.loc[combined_df["tokenizer"] == "custom_tokenizer_P2", "tokenizer"] = "P2"

combined_df
```

16	ComputationalLinguistics	RNN	abstract	900	P2	0.795749
17	ComputerVision	RNN	abstract	900	P2	0.648179
18	InformationTheory	RNN	abstract	112500	P1	0.517547
19	ComputationalLinguistics	RNN	abstract	112500	P1	0.807041
20	ComputerVision	RNN	abstract	112500	P1	0.359349
21	InformationTheory	RNN	abstract	112500	P2	0.521255
22	ComputationalLinguistics	RNN	abstract	112500	P2	0.807871
23	ComputerVision	RNN	abstract	112500	P2	0.595096
24	InformationTheory	LinearSVC	title	900	P1	0.836488
25	ComputationalLinguistics	LinearSVC	title	900	P1	0.881103
26	ComputerVision	LinearSVC	title	900	P1	0.834883
27	InformationTheory	LinearSVC	title	900	P2	0.846895

```
df_abstract = combined_df[combined_df["target_field"] == "abstract"]
df_title = combined_df[combined_df["target_field"] == "title"]
```

```
df_abstract.info()
```

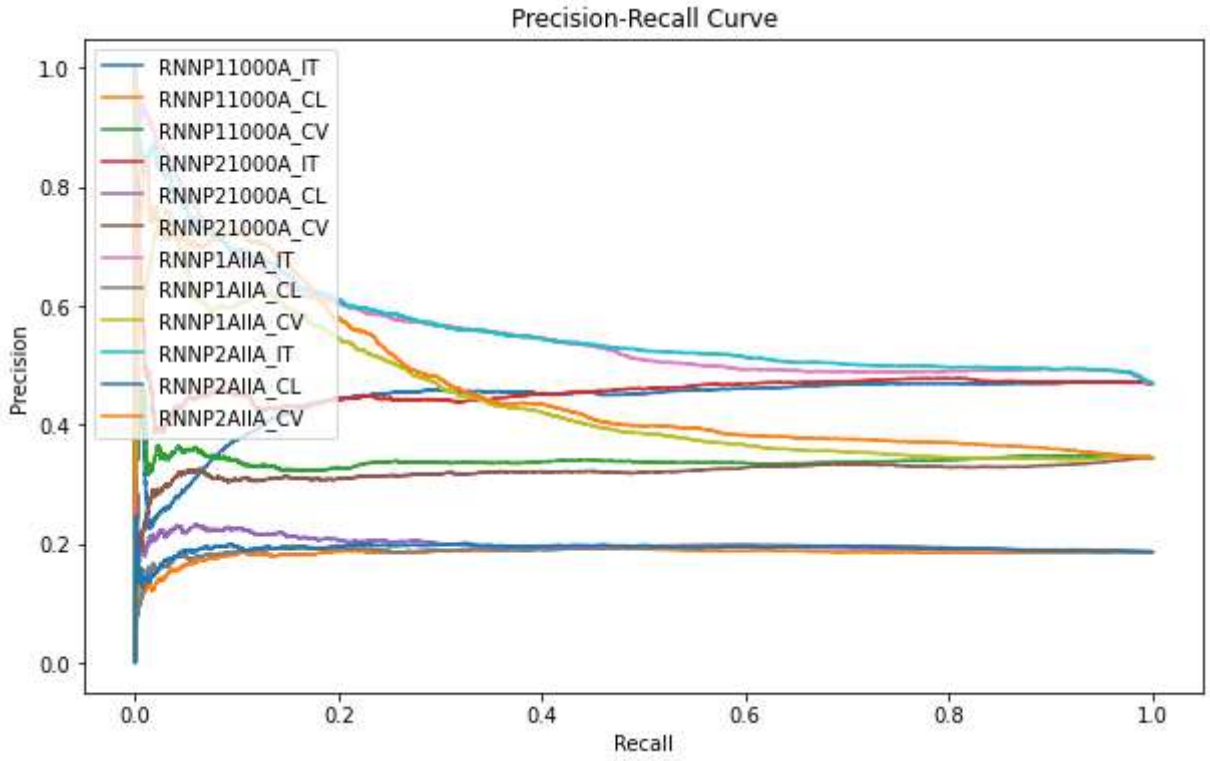
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 24 entries, 12 to 47
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   label_name       24 non-null     object 
 1   model_name       24 non-null     object 
 2   target_field     24 non-null     object 
 3   train_size       24 non-null     int64  
 4   tokenizer        24 non-null     object 
 5   Accuracy         24 non-null     float64
 6   Macro Recall     24 non-null     float64
 7   Macro Precision  24 non-null     float64
 8   Macro F1 score   24 non-null     float64
 9   MCC              24 non-null     float64
 10  Recall           24 non-null     object 
 11  Precision        24 non-null     object 
dtypes: float64(5), int64(1), object(6)
memory usage: 2.4+ KB
```

```
df_title.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 24 entries, 0 to 41
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   label_name       24 non-null     object 
 1   model_name       24 non-null     object 
 2   target_field     24 non-null     object 
 3   train_size       24 non-null     int64  
 4   tokenizer        24 non-null     object 
 5   Accuracy         24 non-null     float64
 6   Macro Recall     24 non-null     float64
 7   Macro Precision  24 non-null     float64
 8   Macro F1 score   24 non-null     float64
 9   MCC              24 non-null     float64
 10  Recall           24 non-null     object 
 11  Precision        24 non-null     object 
dtypes: float64(5), int64(1), object(6)
memory usage: 2.4+ KB
```

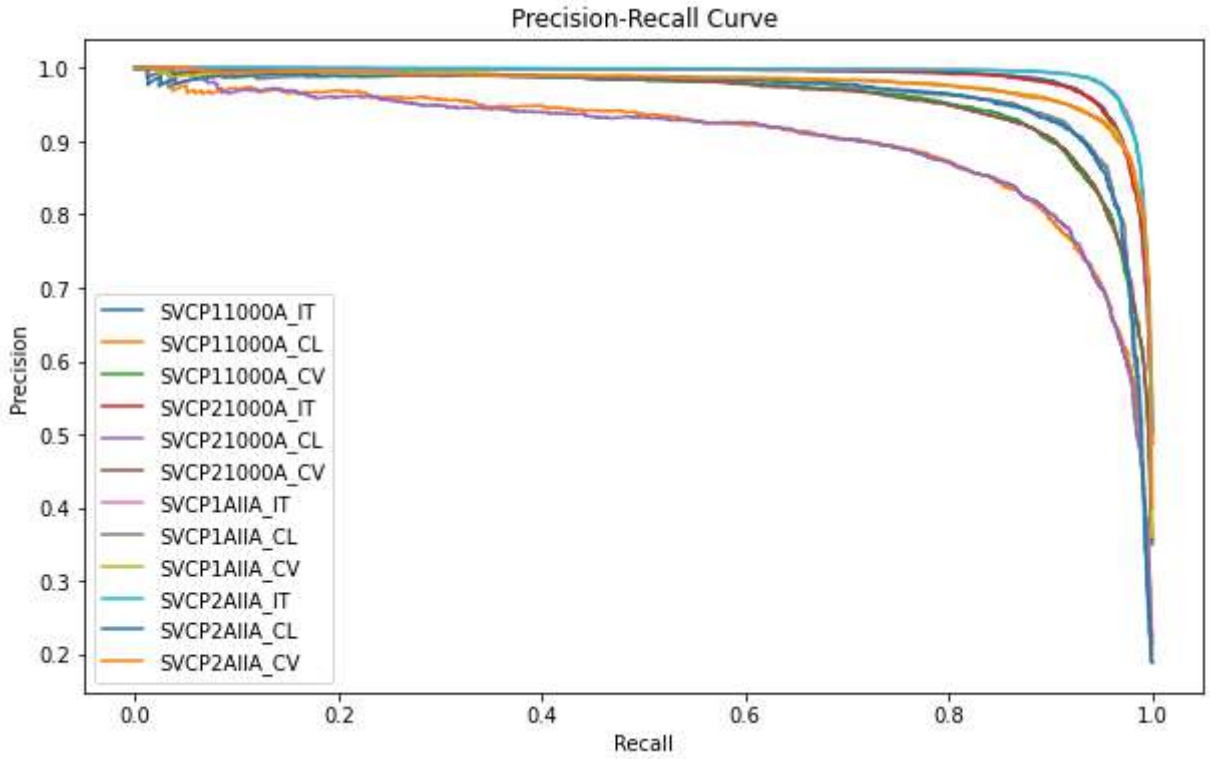
ALL RNN models training on the "abstract" column

```
1 plot_precision_recall_curve(df_abstract[df_abstract["model_name"] == "RNN"])
```



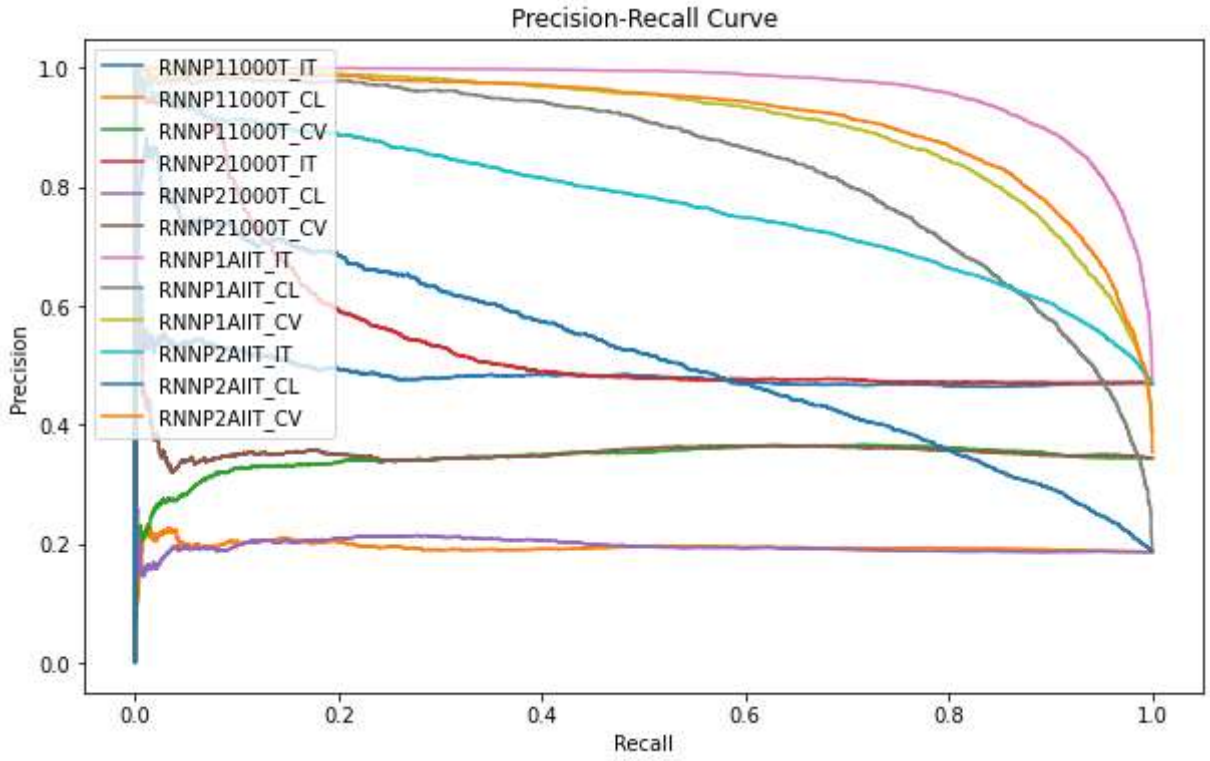
ALL SVC models training on the "abstract" column

```
1 plot_precision_recall_curve(df_abstract[df_abstract["model_name"] != "RNN"])
```



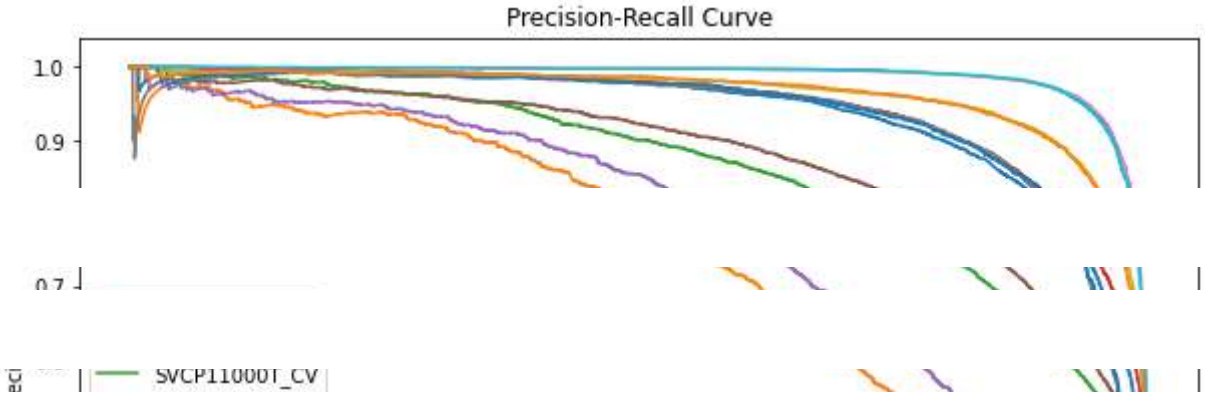
ALL RNN models training on the "title" column

```
1 plot_precision_recall_curve(df_title[df_title["model_name"] == "RNN"])
```



ALL SVC models training on the "abstract" column

```
1 plot_precision_recall_curve(df_title[df_title["model_name"] != "RNN"])
```



Part 2: Topic Modelling

```
# !pip3 install pyldavis==2.1.2
# !pip3 install scikit-learn==0.24.2
# !pip install gensim
```

```
      self.initialize()
    File "C:\Users\Sunny\AppData\Local\Temp\pip-build-env-ocn1k1ah\overlay\Lib\site-
      vc_env = _get_vc_env(plat_spec)
    File "C:\Users\Sunny\AppData\Local\Temp\pip-build-env-ocn1k1ah\overlay\Lib\site-
      return _msvc14_get_vc_env(plat_spec)
    File "C:\Users\Sunny\AppData\Local\Temp\pip-build-env-ocn1k1ah\overlay\Lib\site-
      raise distutils.errors.DistutilsPlatformError(
  distutils.errors.DistutilsPlatformError: Microsoft Visual C++ 14.0 or greater is r
  [end of output]

  note: This error originates from a subprocess, and is likely not a problem with pi
error: metadata-generation-failed

Encountered error while generating package metadata.

See above for output.

note: This is an issue with the package mentioned above, not pip.
hint: See above for details.
Requirement already satisfied: gensim in c:\users\sunny\appdata\local\programs\pytho
```

```
1 import warnings
2 warnings.filterwarnings('ignore')
```

```
1 import re, nltk, spacy, string
2
3 from sklearn.decomposition import LatentDirichletAllocation
4 from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
5 from pprint import pprint
6
7 import pyLDAvis
8 import pyLDAvis.sklearn
9 import matplotlib.pyplot as plt
10 %matplotlib inline
11
12 from plotly.offline import plot
13 import plotly.graph_objects as go
14 import plotly.express as px
15
16 import pandas as pd
17 import seaborn as sns
18 from nltk.tokenize import RegexpTokenizer
19 import nltk
20 from nltk.stem.wordnet import WordNetLemmatizer
21 from gensim.models import Phrases
22 from gensim.corpora import Dictionary
23 from gensim.models.coherencemodel import CoherenceModel
24 from gensim.models import LdaModel
25 from spacy.lang.en.stop_words import STOP_WORDS
26
```

```
import random
SEED = 1
random.seed(SEED)
```

```
def prepare_data(df_train, tokenizer_func, tokenizer, *args, **kwargs):
    docs = df_train['abstract'].tolist()
    raw_docs = docs.copy()

    for idx in range(len(docs)):
        docs[idx] = docs[idx].lower()
        docs[idx] = tokenizer_func(docs[idx], tokenizer, *args, **kwargs)

    bigram = Phrases(docs, min_count=20)
    for idx in range(len(docs)):
        for token in bigram[docs[idx]]:
            if '_' in token:
                docs[idx].append(token)

    dictionary = Dictionary(docs)

    dictionary.filter_extremes(no_below=20, no_above=0.5)

    corpus = [dictionary.doc2bow(doc) for doc in docs]

    return corpus, dictionary

```

```
def train_lda(corpus, dictionary, num_topics_range):
    best_avg_topic_coherence = -100
    best_num_topics = 0
    avg_topic_coherence_values = []
    chunksize = 2000
    passes = 20
    iterations = 400
    eval_every = None  # Don't evaluate model perplexity, takes too much time.

    # Initialize variables
    best_avg_topic_coherence = -100
    best_num_topics = 0
    avg_topic_coherence_values = []
    for num_topics in num_topics_range:
        print(f"Now processing LDA model with {num_topics} number of topics")
        temp = dictionary[0]
        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,
                )

        top_topics = model.top_topics(corpus)
        avg_topic_coherence = sum([t[1] for t in top_topics]) / num_topics
        avg_topic_coherence_values.append(avg_topic_coherence)

        if avg_topic_coherence > best_avg_topic_coherence:
            best_avg_topic_coherence = avg_topic_coherence
            best_num_topics = num_topics
            best_model = model
    print(f"Best number of topics: {best_num_topics}, average topic coherence: {best_avg

    # Plot the average topic coherence values for different numbers of topics
    plt.plot(num_topics_range, avg_topic_coherence_values)
    plt.xlabel("Number of Topics")
    plt.ylabel("Average Topic Coherence")
    plt.show()

    return best_num_topics, best_avg_topic_coherence, avg_topic_coherence_values, best_m
```

```
def plot_topic_coherence(num_topics_range, avg_topic_coherence_values):
    plt.plot(num_topics_range, avg_topic_coherence_values)
    plt.xlabel("Number of Topics")
    plt.ylabel("Average Topic Coherence")
    plt.show()
```

```
def get_document_topics(ldamodel, corpus, texts):
   # 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_num)
            else:
                break
    document_topics_df.columns = ['Dominant_Topic', 'Perc_Contribution', 'Topic_Keywords

    # 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_Keywords

    return document_topics_df
```

```
def find_top_k_doc(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'], ascendi
                                          axis=0)

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

```
stop_words = set(STOP_WORDS)

def custom_tokenizer(text, tokenizer, stop_words):
    return [tok for tok in tokenizer.tokenize(text) if tok.lower() not in stop_words]

def custom_tokenizer_P2(text, tokenizer, stemmer):
    return [stemmer.stem(tok) for tok in tokenizer.tokenize(text)]

```

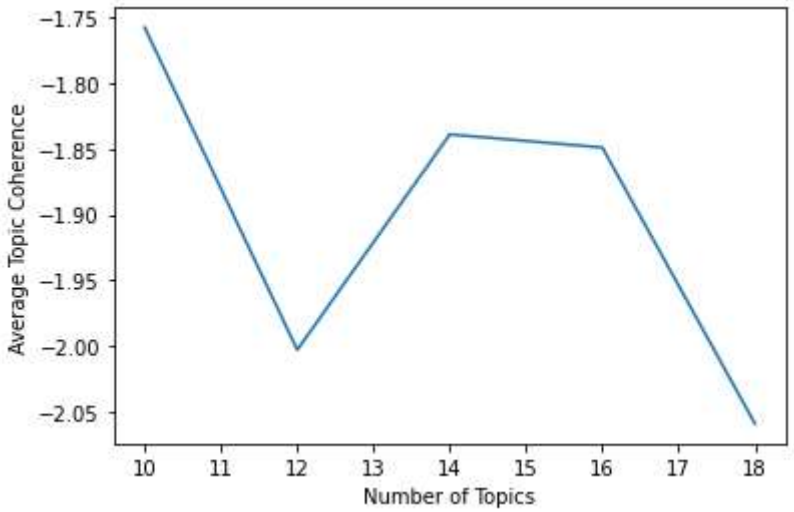
```
import pyLDAvis.gensim
from pprint import pprint

def summarize_lda_results(ldamodel, corpus, dictionary, raw_docs, num_topics, top_k=5):
    # Display LDA visualization
    # lda_display = pyLDAvis.gensim.prepare(ldamodel, corpus, dictionary, sort_topics=Fa
    # pyLDAvis.display(lda_display)

    # Get document topics
    doc_topic_df = get_document_topics(ldamodel, corpus, raw_docs)

    # Find top k documents for each topic
    top_k_df = find_top_k_doc(doc_topic_df, k=top_k)


    return doc_topic_df, top_k_df

```

```
train_data = pd.read_csv("train.csv")
train_1000 = train_data.head(1000)
train_data = train_data.head(20000)
test_data = pd.read_csv("test.csv")
```

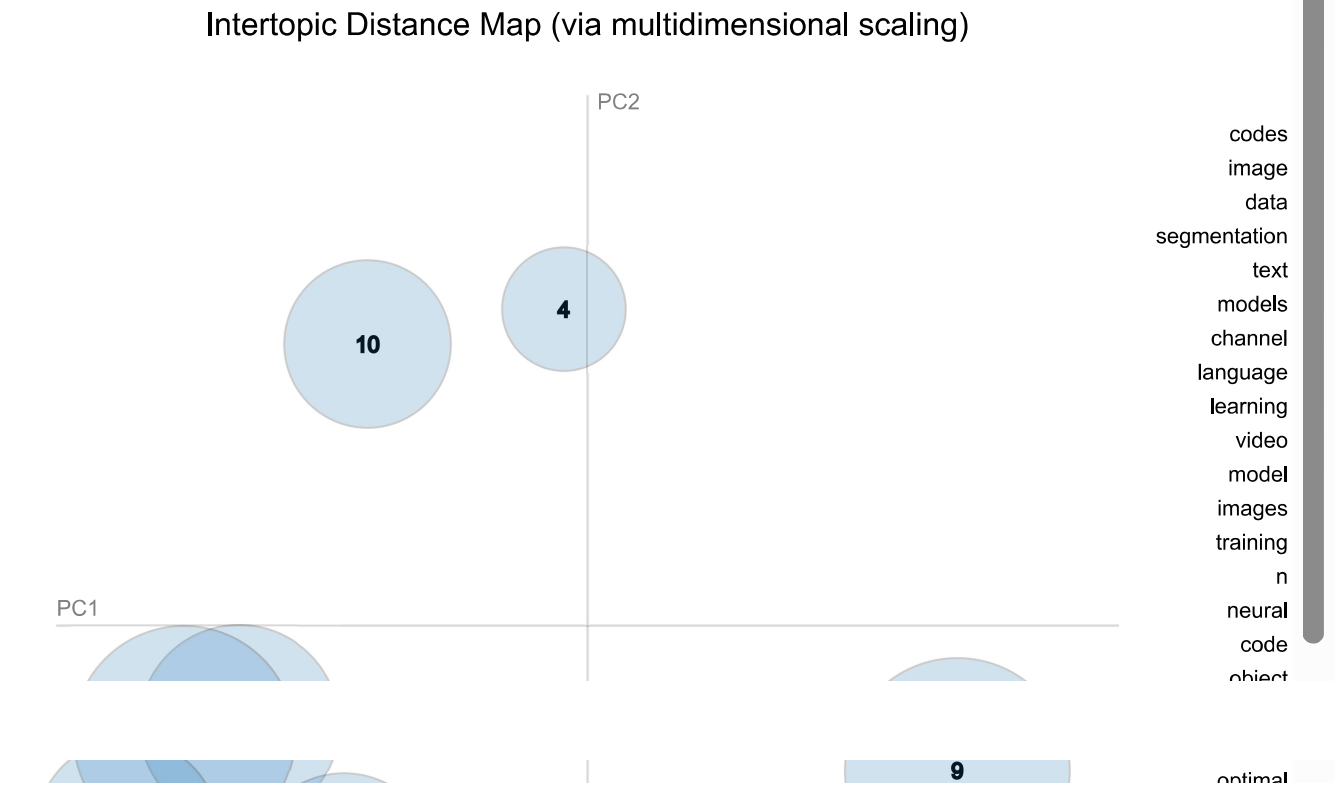
P1 1000

```
df_train = train_1000

raw_docs = df_train['abstract'].tolist().copy

tokenizer = RegexpTokenizer(r'\w+')
corpus, dictionary = prepare_data(df_train, custom_tokenizer, tokenizer, stop_words=stop


# Train LDA model
num_topics_range = range(10, 15, 2)
best_num_topics, best_avg_topic_coherence, avg_topic_coherence_values, best_model = trai

doc_topic_df_P1_1000, top_k_df_P1_1000 = summarize_lda_results(ldamodel=best_model, corp

lda_display = pyLDAvis.gensim.prepare(best_model, corpus, dictionary, sort_topics=False)

pyLDAvis.display(lda_display)
```

```
Now processing LDA model with 10 number of topics
Now processing LDA model with 12 number of topics
Now processing LDA model with 14 number of topics
Now processing LDA model with 16 number of topics
Now processing LDA model with 18 number of topics
Best number of topics: 10, average topic coherence: -1.757952894995696
```



Selected Topic: 0 Previous Topic Next Topic Clear Topic



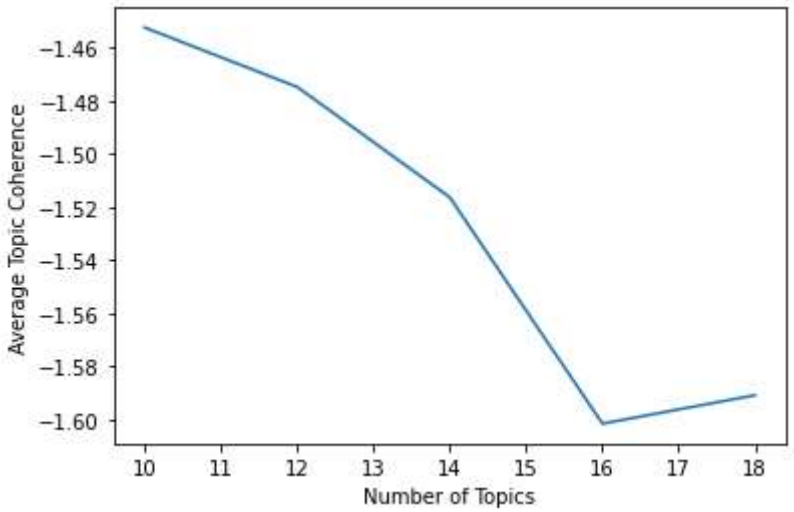
1

P2 1000

```
1 df_train = train_1000
2
```

```
stemmer = PorterStemmer()

tokenizer = RegexpTokenizer(r'\w+')
corpus, dictionary = prepare_data(df_train, custom_tokenizer_P2, tokenizer, stemmer=stem

# Train LDA model
num_topics_range = range(10, 15, 2)
best_num_topics, best_avg_topic_coherence, avg_topic_coherence_values, best_model = trai

doc_topic_df_P2_1000, top_k_df_P2_1000 = summarize_lda_results(ldamodel=best_model, corp

lda_display = pyLDAvis.gensim.prepare(best_model, corpus, dictionary, sort_topics=False)

pyLDAvis.display(lda_display)
```

```
Now processing LDA model with 10 number of topics
Now processing LDA model with 12 number of topics
Now processing LDA model with 14 number of topics
Now processing LDA model with 16 number of topics
Now processing LDA model with 18 number of topics
Best number of topics: 10, average topic coherence: -1.4523068232507323
```



Selected Topic: 0 Previous Topic Next Topic Clear Topic

Intertopic Distance Map (via multidimensional scaling)

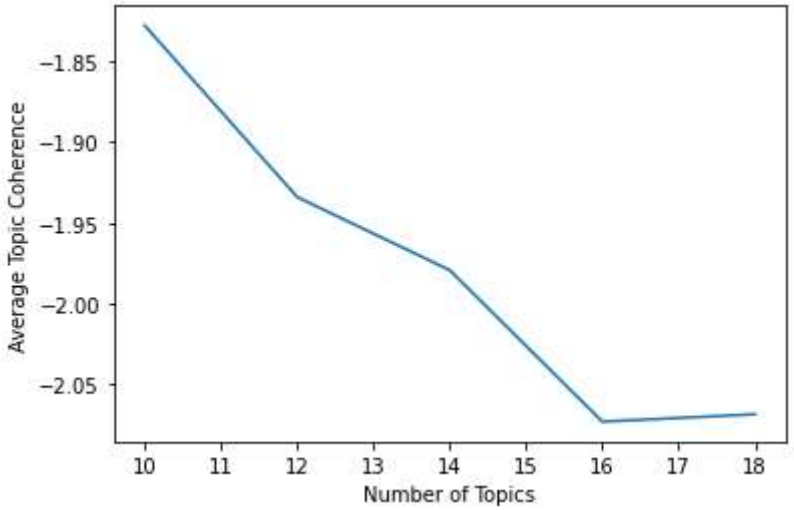
PC2

imag

P1 ALL

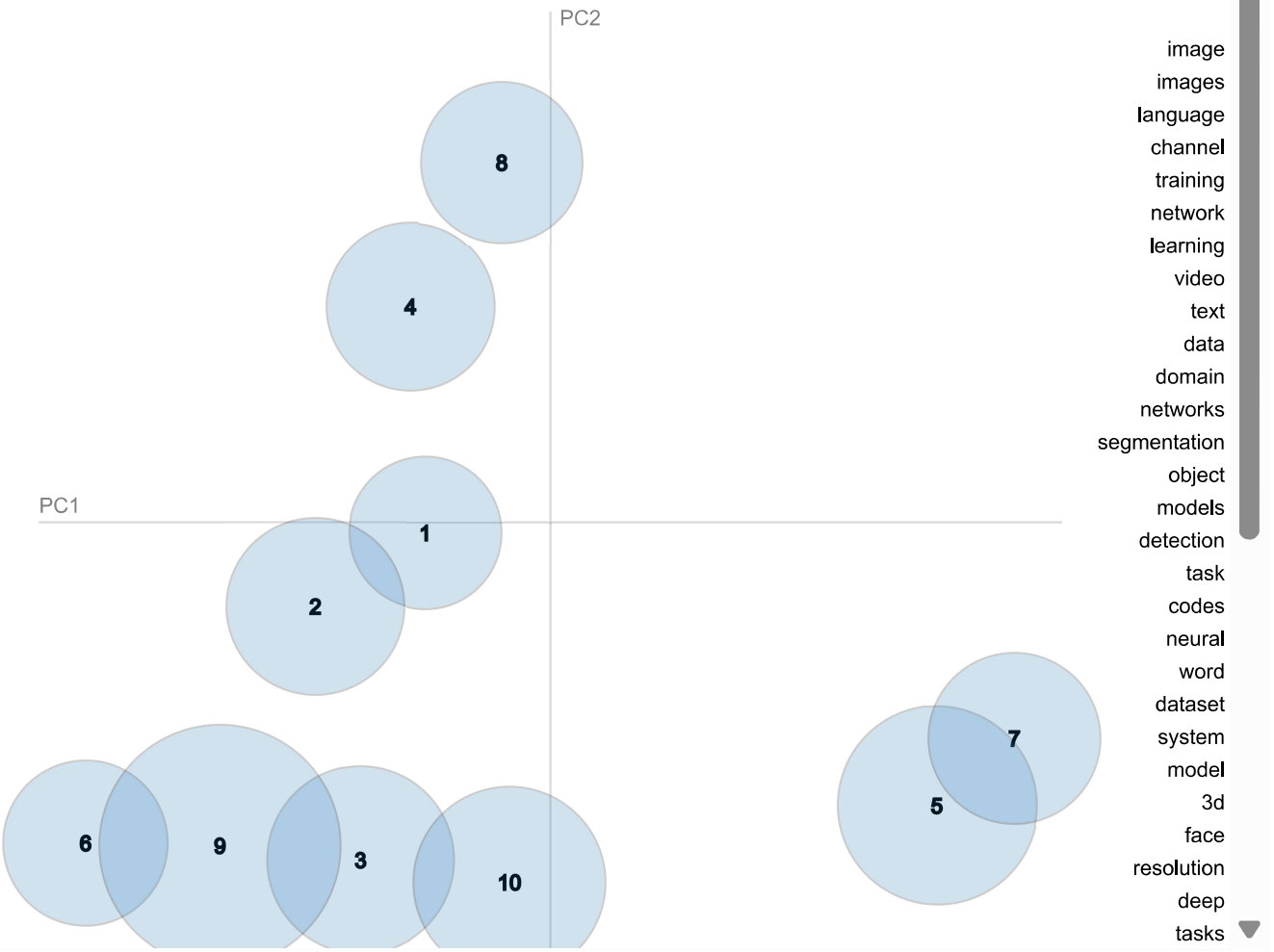
```
df_train = train_data

raw_docs = df_train['abstract'].tolist().copy

tokenizer = RegexpTokenizer(r'\w+')
corpus, dictionary = prepare_data(df_train, custom_tokenizer, tokenizer, stop_words=stop


# Train LDA model
num_topics_range = range(10, 15, 2)
best_num_topics, best_avg_topic_coherence, avg_topic_coherence_values, best_model = trai

doc_topic_df_P1_ALL, top_k_df_P1_ALL = summarize_lda_results(ldamodel=best_model, corpus

lda_display = pyLDAvis.gensim.prepare(best_model, corpus, dictionary, sort_topics=False)

pyLDAvis.display(lda_display)
```

```
Now processing LDA model with 10 number of topics
Now processing LDA model with 12 number of topics
Now processing LDA model with 14 number of topics
Now processing LDA model with 16 number of topics
Now processing LDA model with 18 number of topics
Best number of topics: 10, average topic coherence: -1.8275886401598131
```



Selected Topic: 0 Previous Topic Next Topic Clear Topic

Intertopic Distance Map (via multidimensional scaling)



P2 ALL

```
df_train = train_data

stemmer = PorterStemmer()

tokenizer = RegexpTokenizer(r'\w+')
corpus, dictionary = prepare_data(df_train, custom_tokenizer_P2, tokenizer, stemmer=stem

# Train LDA model
num_topics_range = range(10, 15, 2)
best_num_topics, best_avg_topic_coherence, avg_topic_coherence_values, best_model = trai

doc_topic_df_P2_ALL, top_k_df_P2_ALL = summarize_lda_results(ldamodel=best_model, corpus

lda_display = pyLDAvis.gensim.prepare(best_model, corpus, dictionary, sort_topics=False)

pyLDAvis.display(lda_display)
```