

Comparative Analysis of Neural Networks and General NLP Methods on An Academic Journal Article Database

Self-collected database; NLP methods used; FNN tuned; DNN
constructed and compared, thorough analysis

(Freda) Xiaoyun Yu

Feb 2023

Template I Use

- **CM3015-Machine Learning and Neural Network.**
- I have combined the two templates (1. Deep Learning on a public dataset; 2. Gather your own dataset) in this course provided, to do my own project.
- I have done a deep learning classification project, on a topic - **academic journal articles**, use the whole article body text as the data.

Features of My Project

- Collect dataset on my own (using an API)
- Outstanding text-cleansing work
- Tune hyperparameters in FNN using controlled experiments
- Try to find the best FNN using GridSearchCV
- Have constructed DNN based on pre-trained existing models, including CNN, RNN with LSTM layer, etc.
- Use different combinations of 3 vectorizers (TF-IDF, Count Vectorizer, Bigram) and 4 classifiers (SVC, MNB, Random Forest, KNN) in NLP and compare their performance
- Have attempted to use the methods in literature review (AWD-LSTM, ULMFiT FastAI, Wikipedia2Vec, BERT-for-Task, ...)

Literature Review – Text Classification

Previous Methods	01	02	03
Paper	<i>Large Scale Subject Category Classification of Scholarly Papers with Deep Attentive Neural Networks</i>	<i>Regularizing and optimizing LSTM language models</i>	<i>Universal Language Model Fine-Tuning for Text Classification</i>
Author	Kandimalla et al.	Merity, Keskar, and Socher	Howard and Ruder
Data Source	Web of Science (WoS), year 2015	2 datasets: Penn Treebank; WikiText-2	6 datasets: TREC-6; IMDb; Yelp-bi; Yelp-full; AG; DBpedia
Main Idea	Propose a deep attentive neural network (DANN) that classifies scholarly papers using only their abstracts	Weight-dropped LSTM ; introduce various regularization techniques in LSTM	Propose a new model - universal LSTM model fine-tuning for classification (ULMFiT)
Methods	The proposed network consists of two bidirectional recurrent neural networks followed by an attention layer. Compare their models and others.	Uses a method called ' DropConnect ' to substitute the Dropout neurons in regularization, since Dropouts will weaken the long-distance ability . Other than dropouts, they do not use SGD here, but investigate ASGD (averaged SGD)	Phase 1: General-domain LM pretraining; Phase 2: Target task LM fine-tuning – skills: (1) Discriminative fine-tuning, (2) Slanted triangular learning rate; Phase 3: Target task classifier fine-tuning – skills: (1) Concat pooling, (2) Gradual unfreezing, (3) BPT3C, (4) Bidirectional language model;
Results	(1) The combination of word vectors with TFIDF outperforms character and sentence level embedding models; (2) FastText + BiGRU + Attn and FastText+BiLSTM + Attn (micro-F1 =0.76) ; (3) Retraining FastText and GloVe improves the performance; (4) Character-level embedding models often perform worse than word-level embedding models; (5) The best machine learning model (LR) is outperformed by the best DANN model by roughly 10%.	ASGD has a better effect. 'DropConnect' is effective.	ULMFiT significantly outperformed existing transfer learning techniques and the state-of-the-art on six representative text classification tasks.



Justified Based on Domain and Users

- (1) real journal platforms in academia (e.g. Nature, Science, etc.) to classify newly received articles and therefore simplify the procedure before posting new articles, and will make the news being published more timely without manual work; may stimulate an interdisciplinary collaboration between these platforms and data science platforms to add comprehensive metadata for information retrieval;
- (2) librarianship field, by boosting the development of computer-based technologies to do part of the librarian jobs, and to stimulate the debate of human work vs. computer work;
- (3) real journal platforms if I can find any obstacles or difficulties when developing my project, and will contribute furthermore if I could possibly propose methods to solve them;
- (4) philosophical field, by exploring potential difficulties and proposing possible solutions, since classification is a very basic concept in philosophy;
- (5) students studying in academia (e.g. postgraduate applicants, doctoral students) who have a strong pressure or focus on publishing papers, by receiving stimulation and inspirations found from my project. For instance, the top list of keywords shown in each academic subject can be a hint to expand the students' academic vocabulary set;
- (6) teachers working in academia who need a timely update for the recent research trend, to stimulate their own research and to give their students a prospective working direction;
- (7) data scientists and computer scientists who are also interested in text classification methods. If my models are well-tuned, they may be applied more widely as a transfer learning example, not only in my own dataset, but also in other label-classifications in NLP field, such as sentiment classification, or even image classification. Data scientists may find new inspirations on labelling articles.

Collect Data – from EventRegistry.org

- Their API: <https://www.newsapi.ai/documentation/sandbox?tab=searchArticles>
- I copied their codes from the sandbox to my own Jupyter Notebook file, and output the retrieved data into JSON files.

```
In [4]: import json

er = EventRegistry(apiKey = '5d9ca3e2-04a6-4cf5-b47c-4346d55c385a')
qStr = """
{
    "$query": {
        "$and": [
            {
                "categoryUri": "dmoz/Science/Earth_Sciences"
            },
            {
                "sourceGroupUri": "science/top15"
            },
            {
                "lang": "eng"
            }
        ]
    },
    "$filter": {
        "forceMaxDataTimeWindow": "31",
        "dataType": [
            "news"
        ]
    }
}
"""

q = QueryArticlesIter.initWithComplexQuery(qStr)
# change maxItems to get the number of results that you want
for article in q.execQuery(er, maxItems=100):
    with open('D:/University-of-London-2020/CM3070-Computer-Science-Final-Project/datasets/Event_registry_search_results/API/output.js', 'a') as f:
        f.write(json.dumps(article, ensure_ascii=True))
```


Collect Data – from EventRegistry.org

- Then I convert those JSON files into EXCEL format, and combine them. The following is the snapshot for the combined file.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC
1		record	uri	lang	isDuplicate	date	time	dateTime	dateTimePu	dataType	sim	url	title	body	source	uri.1	dataType.1	title.1	image	eventUri	sentiment	wgt	relevance	authors	uri.2	name	type	isAgency	category
2	0		7.22E+09	eng	FALSE	2022-10-0	17:31:00	2022-10-0	2022-10-0	news	0	https://ww	Towards a	When studying physic	nature.com	news	Nature	https://media.springe	0.011765	25	25								biology
3	1		7.23E+09	eng	FALSE	2022-10-1	16:35:00	2022-10-1	2022-10-1	news	0	https://ww	Efficient an	Human pluripotent ste	nature.com	news	Nature	https://media.springe	0.223529	24	24								biology
4	2		7.21E+09	eng	FALSE	2022-10-0	15:56:00	2022-10-0	2022-10-0	news	0.729412	https://ww	c-Myb red	When worn out, most	nature.com	news	Nature	https://me eng-80682	-0.02745	24	24								biology
5	3		7.22E+09	eng	FALSE	2022-10-1	14:17:00	2022-10-1	2022-10-1	news	0.619608	https://ww	Co-express	Here, to examine the e	nature.com	news	Nature	https://me eng-80994	0.137255	23	23								biology
6	4		7.21E+09	eng	FALSE	2022-09-2	10:31:00	2022-09-2	2022-09-2	news	0.54902	https://ww	Changes in	In our work, we have	nature.com	news	Nature	https://me eng-80570	-0.05098	23	23								biology
7	5		7.23E+09	eng	FALSE	2022-10-1	15:14:00	2022-10-1	2022-10-1	news	0.533333	https://ww	scRNA-seq	Here, in our study, we	nature.com	news	Nature	https://me eng-81077	-0.27059	22	22								biology
8	6		7.22E+09	eng	FALSE	2022-10-1	04:22:00	2022-10-1	2022-10-1	news	0	https://ww	Epigenetic	Epigenetic modificatio	nature.com	news	Nature	https://media.springe	0.160784	22	22								biology
9	7		7.21E+09	eng	FALSE	2022-09-2	15:55:00	2022-09-2	2022-09-2	news	0	https://ww	Rigid tumo	The ability for cancer c	nature.com	news	Nature	https://media.springe	-0.09804	22	22								biology
10	8		7.23E+09	eng	FALSE	2022-10-1	16:12:00	2022-10-1	2022-10-1	news	0.580392	https://ww	Establishm	During early mammali	nature.com	news	Nature	https://me eng-80994	0.207843	21	21								biology
11	9		7.22E+09	eng	FALSE	2022-10-1	10:04:00	2022-10-1	2022-10-1	news	0.678431	https://ww	Naive pluri	Cells positive for SSEA	nature.com	news	Nature	https://me eng-80994	0.145098	21	21								biology
12	10		7.21E+09	eng	FALSE	2022-09-2	20:28:00	2022-09-2	2022-09-2	news	0	https://ww	Autosis an	Historical Perspective	nature.com	news	Nature	https://media.springe	-0.30196	21	21								biology
13	11		7.23E+09	eng	FALSE	2022-10-2	15:37:00	2022-10-2	2022-10-2	news	0	https://ww	Multiplexe	In this study, we used	nature.com	news	Nature	https://media.springe	-0.2549	20	20								biology
14	12		7.23E+09	eng	FALSE	2022-10-2	15:35:00	2022-10-2	2022-10-2	news	0	https://ww	Unappreci	CD40 has long been kn	nature.com	news	Nature	https://media.springe	-0.05882	20	20								biology
15	13		7.23E+09	eng	FALSE	2022-10-1	17:01:00	2022-10-1	2022-10-1	news	0	https://ww	Characteri	Induced pluripotent st	nature.com	news	Nature	https://media.springe	0.176471	20	20								biology
16	14		7.23E+09	eng	FALSE	2022-10-1	08:22:00	2022-10-1	2022-10-1	news	0	https://ww	Online sing	SCALEX implements a	nature.com	news	Nature	https://media.springe	0.035294	20	20								biology
17	15		7.23E+09	eng	FALSE	2022-10-1	17:18:00	2022-10-1	2022-10-1	news	0.592157	https://ww	VASA prot	Our studies aimed to a	nature.com	news	Nature	https://me eng-80994	0.113725	20	20								biology
18	16		7.22E+09	eng	FALSE	2022-10-1	08:19:00	2022-10-1	2022-10-1	news	0	https://ww	Drug toxic	Different response of	nature.com	news	Nature	https://media.springe	0.2	20	20								biology
19	17		7.22E+09	eng	FALSE	2022-10-1	16:27:00	2022-10-1	2022-10-1	news	0	https://ww	Deep learn	Constructing single-ce	nature.com	news	Nature	https://media.springemature.cor		20	20								biology
20	18		7.22E+09	eng	FALSE	2022-10-1	10:19:00	2022-10-1	2022-10-1	news	0	https://ww	Corynoxine	In this study, we aime	nature.com	news	Nature	https://media.springe	-0.05882	20	20								biology
21	19		7.22E+09	eng	FALSE	2022-10-1	17:22:00	2022-10-1	2022-10-1	news	0	https://ww	Clinical imp	More definitive conclu	nature.com	news	Nature	https://media.springe	-0.07451	20	20								biology
22	20		7.21E+09	eng	FALSE	2022-10-0	16:45:00	2022-10-0	2022-10-0	news	0.72549	https://ww	Mouse em	For more than 100 yea	nature.com	news	Nature	https://me eng-80719	0.160784	20	20								biology
23	21		7.21E+09	eng	FALSE	2022-10-0	14:06:00	2022-10-0	2022-10-0	news	0.619608	https://ww	Local immi	Overwhelming system	nature.com	news	Nature	https://me eng-80682	-0.08235	20	20								biology
24	22		7.21E+09	eng	FALSE	2022-09-2	13:10:00	2022-09-2	2022-09-2	news	0.490196	https://ww	In vivo lab	Cell neighbourhoods a	nature.com	news	Nature	eng-80570	-0.19216	20	20								biology
25	23		7.2E+09	eng	FALSE	2022-09-2	21:00:00	2022-09-2	2022-09-2	news	0	https://ww	Decipherin	Evolutionary theory h	nature.com	news	Nature	https://media.springe	-0.22353	20	20								biology
26	24		7.2E+09	eng	FALSE	2022-09-2	15:36:00	2022-09-2	2022-09-2	news	0	https://ww	Advancing	The workshop focus	nature.com	news	Nature	https://media.springe	0.207843	20	20								biology
27	25		7.23E+09	eng	FALSE	2022-10-2	00:05:00	2022-10-2	2022-10-2	news	0	https://ww	New insigh	HESC cultures have be	nature.com	news	Nature	https://media.springe	-0.06667	19	19								biology
28	26		7.23E+09	eng	FALSE	2022-10-2	20:09:00	2022-10-2	2022-10-2	news	0	https://ww	Impact of t	We generated human	nature.com	news	Nature	https://media.springe	0.035294	19	19								biology



The article body (full content)

Feature X



The academic news platform



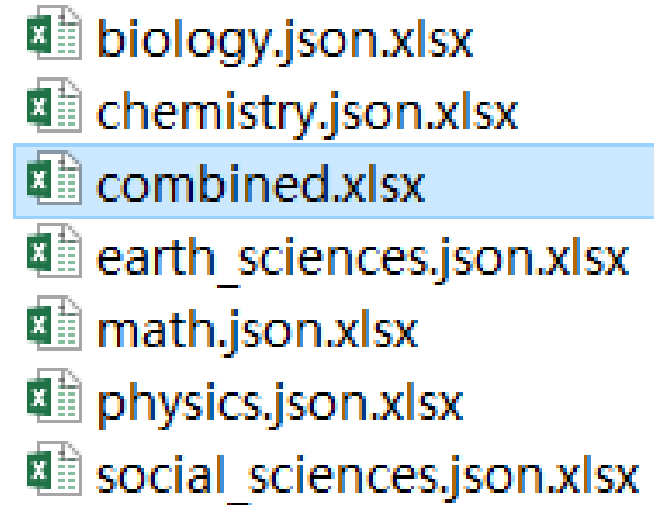
The subject col

(here it is 'biology')

Target Y

Collect Data – from EventRegistry.org

- 6 classes:
 - Biology
 - Chemistry
 - Earth Sciences
 - Math
 - Physics
 - Social Sciences



- Commonsense baseline = $1/6 = 0.167$

Codes Implemented

- Preprocessing: Remove unwanted characters; Lowercase and expand contractions; Remove stopwords & lemmatisation.

```
In [33]: def stopword_lemmatizing(word_list):  
    ''' Remove Stopwords, Punctuations, and also Lemmatize the Original Input  
        I will do two steps of lemmatization - the verb version, and the noun version.  
  
    Parameters:  
        A list which contains many word strings.  
  
    Returns:  
        The stopwords-removed and lemmatized version of this list, also containing word strings.  
    '''  
    result = []  
    # Below order is important! We should always firstly remove stopwords  
    # Otherwise, stopwords would change their form!  
  
    for word in word_list: # for each string in the list  
        word = word.lower() # to get the lower form of each word  
        if not word in stop_words: # to remove stop words.  
            if not word in string.punctuation: # Notice: here  
                word_n = wnل.lemmatize(word, pos='n') # to lemmatize the noun  
                word_v = wnل.lemmatize(word_n, pos='v') # to lemmatize the verb  
  
                result.append(word_v)  
  
    return result
```

Algorithm 4.2-2: Expand Contractions & Lowercase

Input: a text string *text*

- 1: *contractions* ← set rules for contractions; *new_text* ← an empty list to store future list of word strings
- 2: for each character *char* in *text*:
- 3: if *char* is not space:
- 4: read the *char*, and append it into a temporary variable
- 5: else if *char* is space:
- 6: join each character to form the word, and lowercase the word
- 7: if that word is in the *contractions*:
- 8: append the full version to *new_text*
- 9: else:
- 10: just append this word to *new_text*
- 11: *text* ← use space to join these words, then delete spaces at beginning and the end

Output: the new version of that string of *text*

- I have also output my cleansed text data into files, for later to use.

Codes Implemented

- Control randomness

```
In [43]: # Set the random seeds,  
# of the variables.  
import random  
random.seed(15)  
np.random.seed(15)  
tf.random.set_seed(15)
```

- Transform and feed data to Keras
- Text data → numerical data

```
In [50]: # Below codes are from https://github.com/codehax41/BBC-Text-Classification/blob/master/BBC%20using%20Keras.ipynb  
num_words_input = 600 # We should set the max number of crucial words to be identified.  
# Notice that Keras does not input all of the texts. It only uses important words.  
tok = keras.preprocessing.text.Tokenizer(num_words=num_words_input,  
                                          lower=True, # convert to lowercase  
                                          char_level=False)  
  
# The above process will filter default punctuations.  
  
tok.fit_on_texts(X_train_pre) # fit tokenizer to our training text data  
  
X_train = tok.texts_to_matrix(X_train_pre)  
X_test = tok.texts_to_matrix(X_test_pre)
```

```
In [53]: # This cell of codes is from: https://github.com/codehax41/BBC-Text-Classification/blob/master/BBC%20using%20Keras.ipynb  
# Use sklearn utility to convert label strings to numbered index  
encoder = LabelEncoder()  
encoder.fit(Y_train_pre)  
Y_train = encoder.transform(Y_train_pre)  
Y_test = encoder.transform(Y_test_pre)  
# Converts the labels to a one-hot representation  
num_classes = np.max(Y_train) + 1  
Y_train = keras.utils.to_categorical(Y_train, num_classes)  
Y_test = keras.utils.to_categorical(Y_test, num_classes)
```

```
In [54]: Y_train[300] # check. It should be a one-hot encoding array
```

```
Out[54]: array([0., 0., 0., 1., 0., 0.], dtype=float32)
```

Codes Implemented

- A small underfitting ML model (Chollet Step 5)

```
In [82]: # This cell of codes is from our lecture material, the lab REUTERS.
from tensorflow.keras import models
from tensorflow.keras import layers

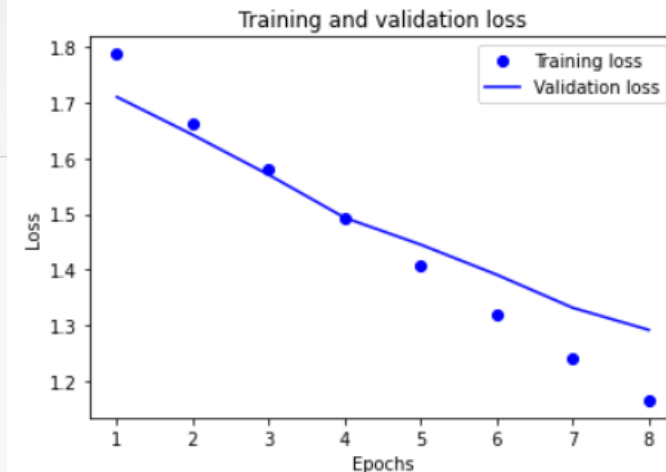
model_small = models.Sequential()
model_small.add(layers.Dense(16, activation='relu', input_shape=(num_words_input,))) # input 600
model_small.add(layers.Dense(16, activation='relu'))
model_small.add(layers.Dense(6, activation='softmax'))
# Notice that for multi-class classification task, the last layer should choose 'softmax' as the
# activation function.

model_small.compile(optimizer='rmsprop',
                    loss='categorical_crossentropy',
                    # Notice that the Chollet book emphasizes that 'categorical_crossentropy' is always used
                    # because 'It minimizes the distance between the probability distributions output by
                    # the model and the true distribution of the targets'.
                    metrics=['accuracy'])
```

```
In [83]: # train-validation splitting
X_train_partial2, X_val2, Y_train_partial2, Y_val2 = train_test_split(
    X_train2, Y_train2, test_size=0.20, random_state=15)
# we set a random state in order to repeat the experiment later

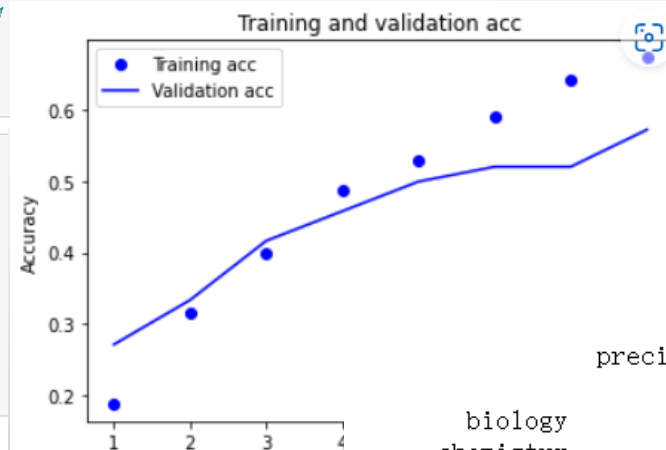
history_small = model_small.fit(X_train_partial2,
                                Y_train_partial2,
                                batch_size=64,
                                epochs=8, # we set a small epoch
                                validation_data=(X_val2, Y_val2))
```

```
Epoch 1/8
6/6 [=====] - 1s 68ms/step - loss: 1.7867 - accuracy: 0.1875 - val_loss: 1.70
98 - val_accuracy: 0.2708
Epoch 2/8
6/6 [=====] - 0s 10ms/step - loss: 1.6607 - accuracy: 0.3151 - val_loss: 1.64
19 - val_accuracy: 0.3333
Epoch 3/8
6/6 [=====] - 0s 11ms/step - loss: 1.5810 - accuracy: 0.3984 - val_loss: 1.57
02 - val_accuracy: 0.4167
Epoch 4/8
```



biology	11	1	0	1	0	2
chemistry	0	6	4	2	4	1
earth_sciences	0	2	13	0	2	2
math	4	3	1	7	8	3
physics	0	2	3	2	12	1
social_sciences	0	6	3	2	1	11
	biology	chemistry	earth_sciences	math	physics	social_sciences

predicted labels



	precision	recall	f1-score	support
biology	0.73	0.73	0.73	15
chemistry	0.30	0.35	0.32	17
earth_sciences	0.54	0.68	0.60	19
math	0.50	0.27	0.35	26
physics	0.44	0.60	0.51	20
social_sciences	0.55	0.48	0.51	23
accuracy			0.50	120
macro avg	0.51	0.52	0.51	120
weighted avg	0.51	0.50	0.49	120

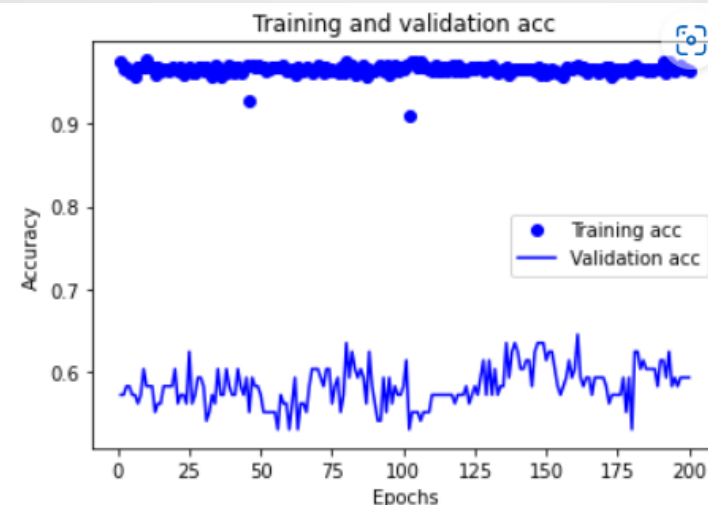
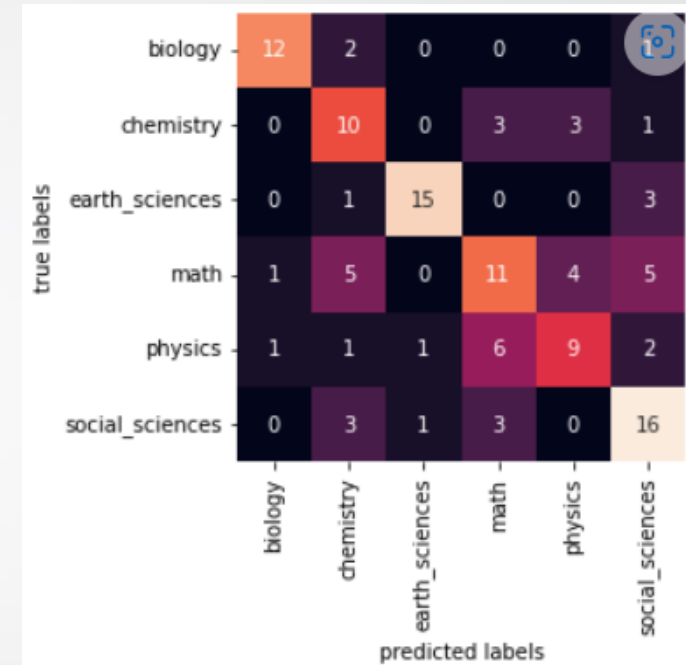
Codes Implemented

- An overfitting ML model (Chollet Step 6)

```
In [67]: # This cell of codes is from our lecture material, the lab REUTERS.
from tensorflow.keras import models
from tensorflow.keras import layers

model_overfit = models.Sequential()
model_overfit.add(layers.Dense(16, activation = 'relu', input_shape = (num_words_input, ))) # input 600
model_overfit.add(layers.Dense(128, activation = 'relu'))
model_overfit.add(layers.Dense(256, activation = 'relu'))
model_overfit.add(layers.Dense(256, activation = 'relu'))
model_overfit.add(layers.Dense(16, activation = 'relu'))
model_overfit.add(layers.Dense(6, activation = 'softmax'))
# Notice that for multi-class classification task, the last layer should choose 'softmax' as the
# activation function.

model_overfit.compile(optimizer = 'rmsprop',
                      loss = 'categorical_crossentropy',
                      # Notice that the Chollet book emphasizes that 'categorical_crossentropy' is always used
                      # because 'It minimizes the distance between the probability distributions output by
                      # the model and the true distribution of the targets'.
                      metrics = ['accuracy'])
```



	precision	recall	f1-score	support
biology	0.86	0.80	0.83	15
chemistry	0.45	0.59	0.51	17
earth_sciences	0.88	0.79	0.83	19
math	0.48	0.42	0.45	26
physics	0.56	0.45	0.50	20
social_sciences	0.57	0.70	0.63	23
accuracy			0.61	120
macro avg	0.63	0.62	0.63	120
weighted avg	0.62	0.61	0.61	120

- Controlled experiments results – batch size

Settings						Tuning parameters								Evaluations	
Date	Data	sys_se ed	np_se ed	tf_s eed	wor ds	Epo chs	Batch size	# hidd en layer s	Neuron nums	Activation functions	optimi zer	loss	regul erisa tion	Accuracy (test)	Loss (test)
Dec 09	Clean sed	15	15	15	600	20	1024	1	64-64-6	relu-relu- softmax	rmspro p	categorical_cr ossentropy	-	0.5667	1.1342
Dec 09	Clean sed	15	15	15	600	20	512	1	64-64-6	relu-relu- softmax	rmspro p	categorical_cr ossentropy	-	0.5750	1.1667
Dec 09	Clean sed	15	15	15	600	20	256	1	64-64-6	relu-relu- softmax	rmspro p	categorical_cr ossentropy	-	0.6250	1.0836
Dec 09	Clean sed	15	15	15	600	20	128	1	64-64-6	relu-relu- softmax	rmspro p	categorical_cr ossentropy	-	0.6333	1.0605
Dec 09	Clean sed	15	15	15	600	20	64	1	64-64-6	relu-relu- softmax	rmspro p	categorical_cr ossentropy	-	0.6750	1.1273
Dec 09	Clean sed	15	15	15	600	20	32	1	64-64-6	relu-relu- softmax	rmspro p	categorical_cr ossentropy	-	0.6250	1.2453
Dec 09	Clean sed	15	15	15	600	20	16	1	64-64-6	relu-relu- softmax	rmspro p	categorical_cr ossentropy	-	0.5500	1.3784

Based on my controlled settings, the best performing **batch size is 64** when operated on Feb 09, with accuracy=0.675. I have found an increasing-then-decreasing pattern of the accuracies when the batch size goes down.

- Controlled experiments results – number of layers

Settings						Tuning parameters								Evaluations	
Date	Data	sys_see	np_see	tf_see	words	Epo	Batch	# hidden layers	Neuron	Activation	optimizer	loss	regularisation	Accuracy (test)	Loss (test)
Jan 31	Clean sed	15	15	15	600	20	64	1	256-256-6	relu-relu-softmax	rmsprop	categorical_crossentropy	-	0.6916666 626930237	1.1270121 335983276
Jan 31	Clean sed	15	15	15	600	20	256	3	128-128-128-128-6	relu-relu-relu-relu-softmax	rmsprop	categorical_crossentropy	-	0.6999999 88079071	1.0144932 270050049
Jan 31	Clean sed	15	15	15	600	20	128	0	128-6	relu-softmax	rmsprop	categorical_crossentropy	-	0.6999999 88079071	0.9842715 859413147
Jan 31	Clean sed	15	15	15	600	20	128	1	128-128-6	relu-relu-softmax	rmsprop	categorical_crossentropy	-	0.65833336 11488342	1.0232696 533203125
Jan 31	Clean sed	15	15	15	600	20	128	2	128-128-128-6	relu-relu-relu-softmax	rmsprop	categorical_crossentropy	-	0.6999999 88079071	0.9746896 624565125

I have run 7 records * (3 settings * 2 repetitions) = 42 experiments, using different settings. The best numbers of layers are quite unpredictable. The best layer number covers all from 0 to 4 in my 6 groups of experiments.

Controlled experiments results – activation functions

Experiments Records – activation functions

Settings						Tuning parameters								Evaluations	
Date	Data	sys_see	np_see	tf_see	words	Epo	Batch	# hidden layers	Neuron	Activation	optimizer	loss	regularisation	Accuracy (test)	Loss (test)
Feb 03	Clean sed	15	15	15	600	20	128	1	64-128-6	relu-relu-softmax	rmsprop	categorical_crossentropy	-	0.65	1.0465116818745932
Feb 03	Clean sed	15	15	15	600	20	128	1	64-128-6	relu-softmax-softmax	rmsprop	categorical_crossentropy	-	0.49166667	1.6607657194137573
Feb 03	Clean sed	15	15	15	600	20	128	1	64-128-6	relu-sigmoid-softmax	rmsprop	categorical_crossentropy	-	0.675	0.9671117067337036
Feb 03	Clean sed	15	15	15	600	20	128	1	64-128-6	relu-tanh-softmax	rmsprop	categorical_crossentropy	-	0.6333333	1.1139755964279174
Feb 03	Clean sed	15	15	15	600	20	128	1	64-128-6	relu-selu-softmax	rmsprop	categorical_crossentropy	-	0.6666667	1.0172423601150513
Feb 03	Clean sed	15	15	15	600	20	128	1	64-128-6	relu-softsign-softmax	rmsprop	categorical_crossentropy	-	0.68333334	0.9269145607948304
Feb 03	Clean sed	15	15	15	600	20	128	1	64-128-6	relu-hard sigmoid-softmax	rmsprop	categorical_crossentropy	-	0.69166666	0.9540115276972453
Feb 03	Clean sed	15	15	15	600	20	128	1	64-128-6	relu-exponential-softmax	rmsprop	categorical_crossentropy	-	0.6666667	1.493983308474223

Experiments Records – activation functions

Settings						Tuning parameters									Evaluations	
Date	Data	sys_see	np_see	tf_see	words	Epochs	Batch size	# hidden layers	Neuron nums	Activation functions	optimizer	loss	regularisation	Accuracy (test)	Loss (test)	
Feb 03	Clean sed	15	15	15	600	20	128	2	64-128-128-6	relu-relu-relu-softmax	rmسprop	categorical_crossentropy	-	0.625	1.0755352258682251	
Feb 03	Clean sed	15	15	15	600	20	128	2	64-128-128-6	relu-softmax-softmax-softmax	rmسprop	categorical_crossentropy	-	0.29166666	1.7877373139063517	
Feb 03	Clean sed	15	15	15	600	20	128	2	64-128-128-6	relu-sigmoid-sigmoid-softmax	rmسprop	categorical_crossentropy	-	0.6166667	1.104865050315857	
Feb 03	Clean sed	15	15	15	600	20	128	2	64-128-128-6	relu-tanh-tanh-softmax	rmسprop	categorical_crossentropy	-	0.6666667	1.0579676866531371	
Feb 03	Clean sed	15	15	15	600	20	128	2	64-128-128-6	relu-selu-selu-softmax	rmسprop	categorical_crossentropy	-	0.64166665	1.2318978706995647	
Feb 03	Clean sed	15	15	15	600	20	128	2	64-128-128-6	relu-softsign-softsign-softmax	rmسprop	categorical_crossentropy	-	0.625	1.1883386691411337	
Feb 03	Clean sed	15	15	15	600	20	128	2	64-128-128-6	relu—hard_sigmoid-hard_sigmoid-softmax	rmسprop	categorical_crossentropy	-	0.6333333	1.1228162209192911	
Feb 03	Clean sed	15	15	15	600	20	128	2	64-128-128-6	relu-exponential-exponential-softmax	rmسprop	categorical_crossentropy	-	0.55833334	1.981887149810791	

From the two groups of experiment, I find that putting **hard_sigmoid** inside the inner layer achieves the best accuracy, and softsign and sigmoid perform the second and third best; If using two same activation functions inside the inner layers, then use **tanh**. Never try to put softmax inside inner layers, since it has got 0.49 and 0.29 when using once and twice. Others are just viable as well.

- Controlled experiments results – neuron numbers

Settings						Tuning parameters									Evaluations	
Date	Data	sys_see ed	np_see d	tf_s eed	words	Epo chs	Batc h size	# hidd en layer s	Neuron nums	Activation functions	opti mizer	loss	regul erisa tion	Accuracy (test)	Loss (test)	
Feb 03	Clean sed	15	15	15	600	15	512	2	32-32-32-6	relu-tanh-tanh-softmax	rmse rop	categorical_cr ossentropy	-	0.575	1.281682046254476	
Feb 03	Clean sed	15	15	15	600	15	512	2	32-64-64-6	relu-tanh-tanh-softmax	rmse rop	categorical_cr ossentropy	-	0.5416667	1.2656379699707032	
Feb 03	Clean sed	15	15	15	600	15	512	2	64-64-64-6	relu-tanh-tanh-softmax	rmse rop	categorical_cr ossentropy	-	0.625	1.1398384332656861	
Feb 03	Clean sed	15	15	15	600	15	512	2	64-64-32-6	relu-tanh-tanh-softmax	rmse rop	categorical_cr ossentropy	-	0.55	1.1970116376876831	
Feb 03	Clean sed	15	15	15	600	15	512	2	64-128-128-6	relu-tanh-tanh-softmax	rmse rop	categorical_cr ossentropy	-	0.55	1.1849392811457315	
Feb 03	Clean sed	15	15	15	600	15	512	2	64-128-64-6	relu-tanh-tanh-softmax	rmse rop	categorical_cr ossentropy	-	0.55833334	1.1169984579086303	
Feb 03	Clean sed	15	15	15	600	15	512	2	128-128-256-6	relu-tanh-tanh-softmax	rmse rop	categorical_cr ossentropy	-	0.68333334	0.9293365836143493	
Feb 03	Clean sed	15	15	15	600	15	512	2	128-128-128-6	relu-tanh-tanh-softmax	rmse rop	categorical_cr ossentropy	-	0.53333336	1.320115900039673	
Feb 03	Clean sed	15	15	15	600	15	512	2	128-128-64-6	relu-tanh-tanh-softmax	rmse rop	categorical_cr ossentropy	-	0.6	1.1395802736282348	
Feb 03	Clean sed	15	15	15	600	15	512	2	128-256-128-6	relu-tanh-tanh-softmax	rmse rop	categorical_cr ossentropy	-	0.6166667	1.1106221596399943	

Feb 03	Clean sed	15	15	15	600	15	512	2	128-256-64-6	relu-tanh-tanh-softmax	rmse	categorical_crossentropy	-	0.56666666	1.1338408788045247
Feb 03	Clean sed	15	15	15	600	15	512	2	512-256-128-6	relu-tanh-tanh-softmax	rmse	categorical_crossentropy	-	0.68333334	0.9334465821584066
Feb 03	Clean sed	15	15	15	600	15	512	2	512-512-256-6	relu-tanh-tanh-softmax	rmse	categorical_crossentropy	-	0.65833336	0.9807055513064067
Feb 03	Clean sed	15	15	15	600	15	512	2	512-1024-512-6	relu-tanh-tanh-softmax	rmse	categorical_crossentropy	-	0.53333336	1.1604699532190959
Feb 03	Clean sed	15	15	15	600	15	512	2	1024-1024-512-6	relu-tanh-tanh-softmax	rmse	categorical_crossentropy	-	0.65833336	0.9688995122909546
Feb 03	Clean sed	15	15	15	600	15	512	2	1024-512-256-6	relu-tanh-tanh-softmax	rmse	categorical_crossentropy	-	0.55	1.2143640756607055
Feb 03	Clean sed	15	15	15	600	15	512	2	1024-512-128-6	relu-tanh-tanh-softmax	rmse	categorical_crossentropy	-	0.55833334	1.0940533717473349

It seems that, with the same settings,
128-128-256-6 neurons, and
512-256-128-6 neurons
 performs the best, with
 accuracy=0.68333334.
 The second best are
 512-512-256-6 and
 1024-1024-512-6.

Less neurons or bottleneck
 effects (128-128-128-6) indeed
 cannot get a high score.

- Controlled experiments results – optimizer

Experiments Records – optimizers

Settings						Tuning parameters								Evaluations	
Date	Data	sys_see ed	np_see d	tf_s eed	wor ds	Epo chs	Batch size	# hidd en layer s	Neuron nums	Activation functions	optimi zer	loss	regul erisa tion	Accuracy (test)	Loss (test)
Feb 03	Clean sed	15	15	15	600	20	128	2	512-256-128-6	relu-tanh-tanh-softmax	rmsprop	categorical_crossentropy	-	0.65833336	1.137278159459432
Feb 03	Clean sed	15	15	15	600	20	128	2	512-256-128-6	relu-tanh-tanh-softmax	adam	categorical_crossentropy	-	0.7083333	1.2403667132059732
Feb 03	Clean sed	15	15	15	600	20	128	2	512-256-128-6	relu-tanh-tanh-softmax	sgd	categorical_crossentropy	-	0.425	1.4351878404617309
Feb 03	Clean sed	15	15	15	600	20	128	2	512-256-128-6	relu-tanh-tanh-softmax	adagrad	categorical_crossentropy	-	0.35833332	1.663960321744283
Feb 03	Clean sed	15	15	15	600	20	128	2	512-256-128-6	relu-tanh-tanh-softmax	adamax	categorical_crossentropy	-	0.69166666	1.0454886635144551
Feb 03	Clean sed	15	15	15	600	20	128	2	512-256-128-6	relu-tanh-tanh-softmax	adadelta	categorical_crossentropy	-	0.175	1.8860424121220907

Adam performs really well, with accuracy of nearly 0.708. Adamax and RMSprop are both good to use. Never try to use AdaDelta or AdaGrad. SGD performs with a rather low score as well.

Controlled experiments results – regularizer

Experiments Records – regularisation

Settings						Tuning parameters								Evaluations	
Date	Data	sys_see	np_see	tf_see	words	Epochs	Batch size	# hidden layers	Neuron nums	Activation functions	optimizer	loss	Regularisation (same for all 4 layers)	Accuracy (test)	Loss (test)
Feb 08	Clean sed	15	15	15	600	20	128	2	512-256-128-6	relu-tanh-tanh-softmax	adam	categorical_crossentropy	None	0.658333 36114883 42	1.291331 76803588 87
Feb 08	Clean sed	15	15	15	600	20	128	2	512-256-128-6	relu-tanh-tanh-softmax	adam	categorical_crossentropy	l1(0.00001)	0.666666 68653488 16	1.493371 36745452 88
Feb 08	Clean sed	15	15	15	600	20	128	2	512-256-128-6	relu-tanh-tanh-softmax	adam	categorical_crossentropy	l1(0.00001)	0.683333 33730697 63	2.956308 84170532 23
Feb 08	Clean sed	15	15	15	600	20	128	2	512-256-128-6	relu-tanh-tanh-softmax	adam	categorical_crossentropy	l1(0.0001)	0.658333 36114883 42	7.496500 01525878 9
Feb 08	Clean sed	15	15	15	600	20	128	2	512-256-128-6	relu-tanh-tanh-softmax	adam	categorical_crossentropy	l1(0.01)	0.166666 67163372 04	20.80710 60180664 06
Feb 08	Clean sed	15	15	15	600	20	128	2	512-256-128-6	relu-tanh-tanh-softmax	adam	categorical_crossentropy	l1(0.1)	0.208333 32836627 96	186.6714 17236328 12
Feb 08	Clean sed	15	15	15	600	20	128	2	512-256-128-6	relu-tanh-tanh-softmax	adam	categorical_crossentropy	l2(0.00001)	0.691666 66269302 37	1.300762 77256011 96
Feb 08	Clean sed	15	15	15	600	20	128	2	512-256-128-6	relu-tanh-tanh-softmax	adam	categorical_crossentropy	l2(0.00001)	0.683333 33730697 63	1.410880 56564331 05

Feb 08	Clean sed	15	15	15	600	20	128	2	512-256-128-6	relu-tanh-tanh-softmax	adam	categorical_crossentropy	l2(0.001)	0.666666 68653488 16	2.105928 89785766 6
Feb 08	Clean sed	15	15	15	600	20	128	2	512-256-128-6	relu-tanh-tanh-softmax	adam	categorical_crossentropy	l2(0.01)	0.691666 66269302 37	3.848038 67340087 9
Feb 08	Clean sed	15	15	15	600	20	128	2	512-256-128-6	relu-tanh-tanh-softmax	adam	categorical_crossentropy	l2(0.1)	0.441666 66269302 37	10.25903 51104736 33
Feb 08	Clean sed	15	15	15	600	20	128	2	512-256-128-6	relu-tanh-tanh-softmax	adam	categorical_crossentropy	l1_l2(l1=0.0001, l2=0.00001)	0.649999 97615814 21	1.477874 27902221 68
Feb 08	Clean sed	15	15	15	600	20	128	2	512-256-128-6	relu-tanh-tanh-softmax	adam	categorical_crossentropy	l1_l2(l1=0.0001, l2=0.00001)	0.658333 36114883 42	2.927745 10383605 96
Feb 08	Clean sed	15	15	15	600	20	128	2	512-256-128-6	relu-tanh-tanh-softmax	adam	categorical_crossentropy	l1_l2(l1=0.0001, l2=0.0001)	0.691666 66269302 37	7.606308 93707275 4
Feb 08	Clean sed	15	15	15	600	20	128	2	512-256-128-6	relu-tanh-tanh-softmax	adam	categorical_crossentropy	l1_l2(l1=0.1, l2=0.1)	0.191666 66269302 368	200.3531 95190429 7
Feb 08	Clean sed	15	15	15	600	20	128	2	512-256-128-6	relu-tanh-tanh-softmax	adam	categorical_crossentropy	l1_l2(l1=0.0001, l2=0.1)	0.474999 99403953 55	10.30813 78936767 58
Feb 08	Clean sed	15	15	15	600	20	128	2	512-256-128-6	relu-tanh-tanh-softmax	adam	categorical_crossentropy	l1_l2(l1=0.1, l2=0.00001)	0.191666 66269302 368	186.4034 11865234 38

I have settled 17 experiments. I find that, there are three sets of regularizers perform the best, with test accuracy of 0.6916666626930237. They are:

all layers use l2(0.00001)

all layers use l2(0.01)

all layers use l1_l2(l1=0.001, l2=0.001)

Notice: should never use l1(0.01) or l1(0.1) or l1_l2(l1=0.1, l2=0.1) or l1_l2(l1=0.1, l2=0.00001) in all layers.

L2 can be allowed to be big, but l1 can never be big.

Codes

Sklearn.model_selection.GridSearchCV on FNN

```
def my_model3(act_func, neuron_num1, neuron_num2, num_layers, loss_f, layers_num):
    # train the model:
    model = models.Sequential()
    # just pick any parameters here to control the other variables:
    model.add(layers.Dense(neuron_num1, activation = act_func, input_shape = (600,)))

    # append identical inner layers:
    i = 0
    while i < num_layers:
        model.add(layers.Dense(neuron_num2, activation =act_func))
        i += 1

    # the output layer:
    model.add(layers.Dense(6, activation = 'softmax')) # must be 6 categories, should use softmax
    model.compile(optimizer = 'adam', loss = loss_f, metrics = ['accuracy'])

    return model

from sklearn.model_selection import GridSearchCV
# from tensorflow.keras.wrappers.scikit_learn import KerasRegressor
import scikeras
from scikeras.wrappers import KerasClassifier, KerasRegressor

# SciKeras renamed the constructor argument build_fn to model
classifier3 = KerasClassifier(model = my_model3, act_func='relu', neuron_num1=32,
                             neuron_num2=128, num_layers=0, loss_f='categorical_crossentropy',
                             layers_num=3)

# hyperparameters:
hyperparameters3 = {
    'neuron_num1': [32, 64],
    'neuron_num2': [128, 256, 512],
    'num_layers': [0, 1, 2, 3],
    'act_func': ['relu', 'tanh', 'selu'],
    'batch_size': [128, 256, 512],
    'epochs': [20, 40, 60],
    'loss_f': ['categorical_crossentropy', 'mean_absolute_error'],
    'optimizer': ['adam', 'rmsprop', 'adamax']
}
```

Best hyperparameters are:
{ 'act_func': 'relu',
'batch_size': 256, 'epochs':
60, 'loss_f':
'mean_absolute_error',
'neuron_num1': 64,
'neuron_num2': 256,
'num_layers': 0, 'optimizer':
'adam' } Best score is: 0.5625

```
# train-validation splitting:
from sklearn.model_selection import train_test_split
# this time I use the cleansed dataset
X_train_partial2, X_val2, Y_train_partial2, Y_val2 = train_test_split(
    X_train2, Y_train2, test_size=0.20, random_state=15)
# we set a random state in order to repeat the experiment later

grid_search_trial3 = GridSearchCV(estimator = classifier3, param_grid = hyperparameters3, scoring = 'accu
# In sklearn, any machine learning model is an estimator. estimator.get_params()

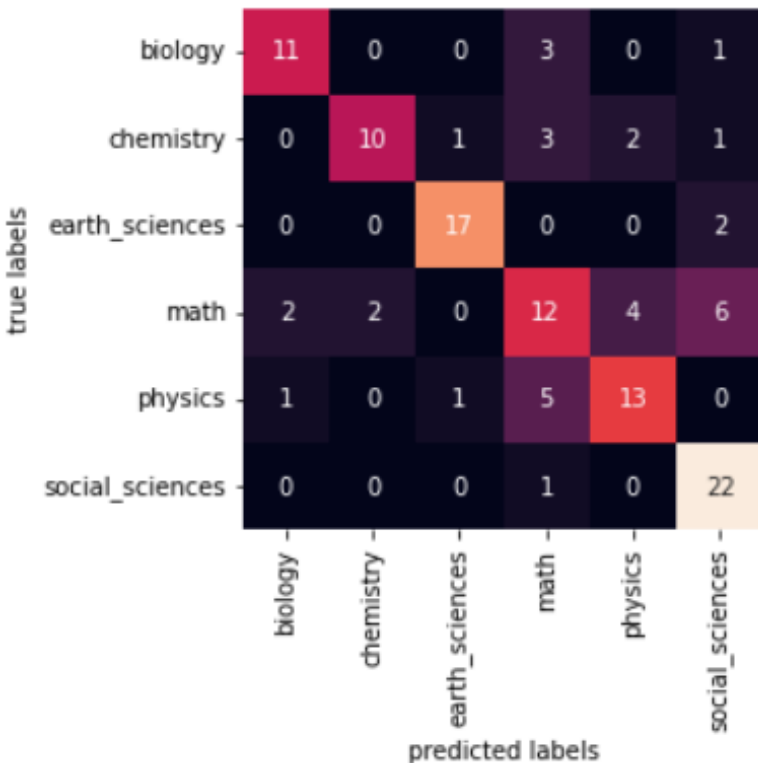
# run on Feb 02
grid_search_fit3 = grid_search_trial3.fit(X_train_partial2, Y_train_partial2, verbose=0, validation_data=

best_parameters3 = grid_search_fit3.best_params_
best_score3 = grid_search_fit3.best_score_

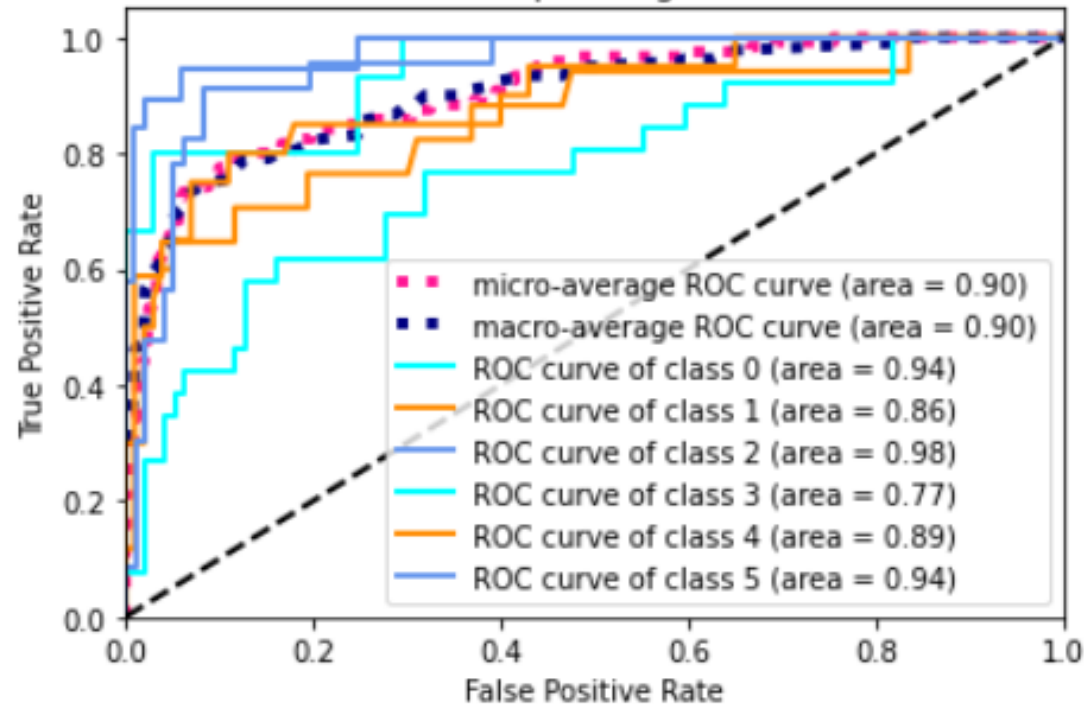
print("Best hyperparameters are: " + str(best_parameters3), "Best score is: ", best_score3)
```

Repetition of The Best FNN Model

Date	Data	sys_se ed	np_see d	tf_s eed	wor ds	Epo chs	Batc h size	# hidden layers	Neuron nums	Activation functions	optimiz er	loss	regul erisa tion	Accuracy (test)	Loss (test)
Feb 03	Clean sed	15	15	15	600	20	64	0	256-6	relu-softmax	rmsprop	categorical_c rossentropy	-	0.73333334 92279053	0.9777682 423591614



Some extension of Receiver operating characteristic to multi-class



Accuracy=0.708

Lack of replicability,
may caused by
randomness of
Dropout,
randomness of
weight, ...


```
In [68]: # convert to a numerical vector
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import LinearSVC
from sklearn.pipeline import make_pipeline

model_TFIDFSVC = make_pipeline(TfidfVectorizer(), LinearSVC(random_state=0, tol=1e-5))

In [70]: # Apply model to the training data, then predict labels for test data
model_TFIDFSVC.fit(X_train_pre2, Y_train_pre2)
labels_TFIDFSVC = model_TFIDFSVC.predict(X_test_pre2)
```

Codes Implemented

- Classification Using Classical NLP methods

They score really high!

Table 5.1.2-1 Performances of My NLP Models

Date	Random seeds				Vectorizer	Classifier	Make pipeline?	Accuracy	Precision (weighted avg)	Recall (weighted avg)	F1 (weighted avg)
	random. seed	np.random. seed	tf.random. set_seed	classifier random_state							
Dec 09	15	15	15	0	TF-IDF	Support Vector Classifier	Yes	0.86	0.87	0.86	0.85
Dec 09	15	15	15	0	Count Vectorizer	Support Vector Classifier	Yes	0.86	0.87	0.86	0.85
Dec 09	15	15	15	0	Bigram	Support Vector Classifier	Yes	0.68	0.70	0.68	0.68
Jan31	15	15	15	-	TF-IDF	Multinomial Naive Bayes	Yes	0.78	0.80	0.78	0.76
Jan31	15	15	15	-	Count Vectorizer	Multinomial Naive Bayes	Yes	0.78	0.79	0.78	0.78
Jan31	15	15	15	-	Bigram	Multinomial Naive Bayes	Yes	0.74	0.75	0.74	0.72
Feb21	15	15	15	15	TF-IDF	Random Forest	Yes	0.73	0.74	0.72	0.71
Feb21	15	15	15	15	Count Vectorizer	Random Forest	Yes	0.69	0.68	0.69	0.68
Feb21	15	15	15	15	Bigram	Random Forest	Yes	0.54	0.64	0.54	0.56
Feb21	15	15	15	-	TF-IDF	K-Nearest Neighbors	Yes	0.82	0.86	0.82	0.82
Feb21	15	15	15	-	Count Vectorizer	K-Nearest Neighbors	Yes	0.55	0.60	0.55	0.56
Feb21	15	15	15	-	Bigram	K-Nearest Neighbors	Yes	0.26	0.54	0.26	0.18

Pre-trained model source: Bilibili video tutorial

https://www.bilibili.com/video/BV1u7411d7zU/?share_source=copy_web&vd_source=296c14837e03501foo801a512d70f87e

```
model_1DCNN.add(layers.Embedding(input_dim=2000,      # size of the vocabulary
                                output_dim=128,
                                input_length=600))
model_1DCNN.add(layers.Conv1D(256,      # output size
                              3,      # conv core size
                              padding='same',
                              activation='relu'))
# The above layer creates a convolution kernel that is convolved with the layer input over
# spatial (or temporal) dimension to produce a tensor of outputs.
model_1DCNN.add(layers.MaxPooling1D(3, 3, padding='same'))
model_1DCNN.add(layers.Conv1D(32, 3, padding='same', activation='relu'))
model_1DCNN.add(layers.Flatten())
model_1DCNN.add(layers.Dropout(0.3))
model_1DCNN.add(layers.BatchNormalization()) # the layer of batch normalization
model_1DCNN.add(layers.Dense(256, activation='relu'))
model_1DCNN.add(layers.Dropout(0.2))

model_1DCNN.add(layers.Dense(6, activation='softmax'))

model_1DCNN.compile(optimizer='rmsprop', loss='categorical_crossentropy',
                    metrics=['accuracy'])

model_1DCNN.summary()
```

train-validation splitting

```
X_train_partial2, X_val2, Y_train_partial2, Y_val2 = train_test_split(
    X_train2, Y_train2, test_size=0.20, random_state=15)
```

we set a random state in order to repeat the experiment later

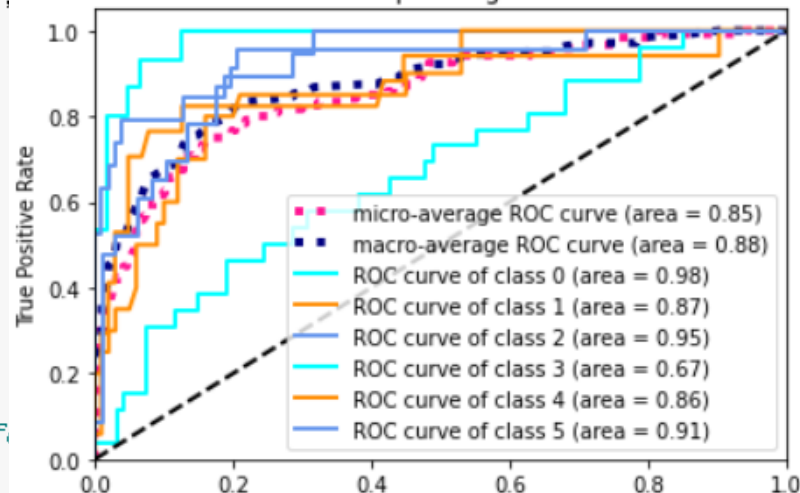
```
history_1DCNN = model_1DCNN.fit(X_train_partial2,
                                Y_train_partial2,
                                batch_size=64, # set a fixed batch size
                                epochs=31,    # we just set any fixed number for epochs
                                validation_data=(X_val2, Y_val2))
```

Codes Implemented

conv1d_16 (Conv1D)	(None, 600, 256)	98560
max_pooling1d_8 (MaxPooling1D)	(None, 200, 256)	0
conv1d_17 (Conv1D)	(None, 200, 32)	24608
flatten_8 (Flatten)	(None, 6400)	0
dropout_16 (Dropout)	(None, 6400)	0
batch_normalization_8 (Batch Normalization)	(None, 6400)	25600
dense_16 (Dense)	(None, 256)	1638656
dropout_17 (Dropout)	(None, 256)	0
dense_17 (Dense)	(None, 6)	1542

```
results_1DCNN = model_1DCNN.evaluate(X_test2, Y_test2)
results_1DCNN
```

Some extension of Receiver operating characteristic to multi-class



loss: 1.3253 - accuracy: 0.6333

- The best CNN

Accuracy=0.633

Codes Implemented

- RNN with LSTM layer

```
model_RNN = Sequential()
# the embedding layer has the shape of (samples, input_length, 8):
model_RNN.add(Embedding(600, 128))
# model_RNN.add(SimpleRNN(64))
model_RNN.add(LSTM(64, activation='tanh',
                  recurrent_activation='hard_sigmoid',
                  use_bias=True,
                  kernel_initializer='glorot_uniform',
                  recurrent_initializer='orthogonal',
                  bias_initializer='zeros',
                  unit_forget_bias=True,
                  dropout=0.0, recurrent_dropout=0.0,
                  return_sequences=True))
```

```
model_RNN.add(Dropout(0.4))
model_RNN.add(LSTM(64, activation='tanh'))
model_RNN.add(Dropout(0.4))

model_RNN.add(Dense(6, activation='softmax'))

model_RNN.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
model_RNN.summary()
```

```
# train-validation splitting
X_train_partial2, X_val2, Y_train_partial2, Y_val2 = train_test_split(
    X_train2, Y_train2, test_size=0.20, random_state=15)
# we set a random state in order to repeat the experiment later

history_RNN = model_RNN.fit(X_train_partial2,
                            Y_train_partial2,
                            batch_size=128, # set a fixed batch size
                            epochs=15, # we just set any fixed number for co
                            validation_data=(X_val2, Y_val2))
```

```
final_LSTM = model_RNN.evaluate(X_test2, Y_test2)
final_LSTM
```

```
4/4 [=====] - 0s 98ms/step - loss: 1.7375 - accuracy: 0.2667
[1.7375144958496094, 0.26666666805744171]
```

Accuracy=0.267

Model: "sequential_7"

Layer (type)	Output Shape	Param #
embedding_7 (Embedding)	(None, None, 128)	76800
lstm_10 (LSTM)	(None, None, 64)	49408
dropout_6 (Dropout)	(None, None, 64)	0
lstm_11 (LSTM)	(None, 64)	33024
dropout_7 (Dropout)	(None, 64)	0
dense_3 (Dense)	(None, 6)	390

Pre-trained model source: Chollet's ML book Ch.6

Codes Implemented

- Latent Dirichlet Allocation

(Oct 25)

```
Topic 0      ['study', 'year', 'research', 'one', 'university', 'say', 'find', 'also', 'social', 'people']
Topic 1      ['cell', 'fig', 'expression', 'gene', 'study', 'also', 'show', 'result', 'group', 'level']
Topic 2      ['quantum', 'state', 'phase', 'fig', 'show', 'system', 'two', 'time', 'use', 'energy']
Topic 3      ['use', 'cell', '10', 'sample', 'min', 'flow', 'perform', 'mm', 'study', 'medium']
Topic 4      ['hurricane', 'storm', 'water', 'flood', 'change', 'increase', 'crystal', 'temperature', 'show', 'plasma']
Topic 5      ['data', 'use', 'model', 'method', 'fig', 'structure', 'magnetic', 'result', 'field', 'study']
```

(Fig 5.1.5-1 LDA Result)

Final Comparison

- Although I have put great efforts on tuning hyperparameters, the best FNN I can bring to you is with accuracy=0.733

Settings						Tuning parameters								Evaluations	
Date	Data	sys_seed	np_seed	tf_seed	words	Epochs	Batch size	# hidden layers	Neuron nums	Activation functions	optimizer	loss	regularisation	Accuracy (test)	Loss (test)
Feb 03	Clean sed	15	15	15	600	20	64	0	256-6	relu-softmax	rmsprop	categorical_crossentropy	-	0.7333333492279053	0.9777682423591614
Feb 03	Clean sed	15	15	15	600	20	128	2	512-256-128-6	relu-tanh-tanh-softmax	adam	categorical_crossentropy	-	0.7083333	1.2403667132059732
Jan 31	Clean sed	15	15	15	600	20	256	3	128-128-128-128-6	relu-relu-relu-relu-softmax	rmsprop	categorical_crossentropy	-	0.699999988079071	1.0144932270050049



Settings 4.9-1: 1D CNN

Layer types: (1) Embedding(2000, 128, 600); (2) Conv1D(256, 3); (3) MaxPooling1D(3, 3); (4) Conv1D(32, 3, 'relu'); (5) Flatten; (6) Dropout(0.3); (7) BatchNormalization; (8) Dense(256, 'relu'); (9) Dropout(0.2); (10) Dense(6, 'softmax').
np_seed: 15; tf_seed: 15; words: 600; batch size: 64; epoch: 31



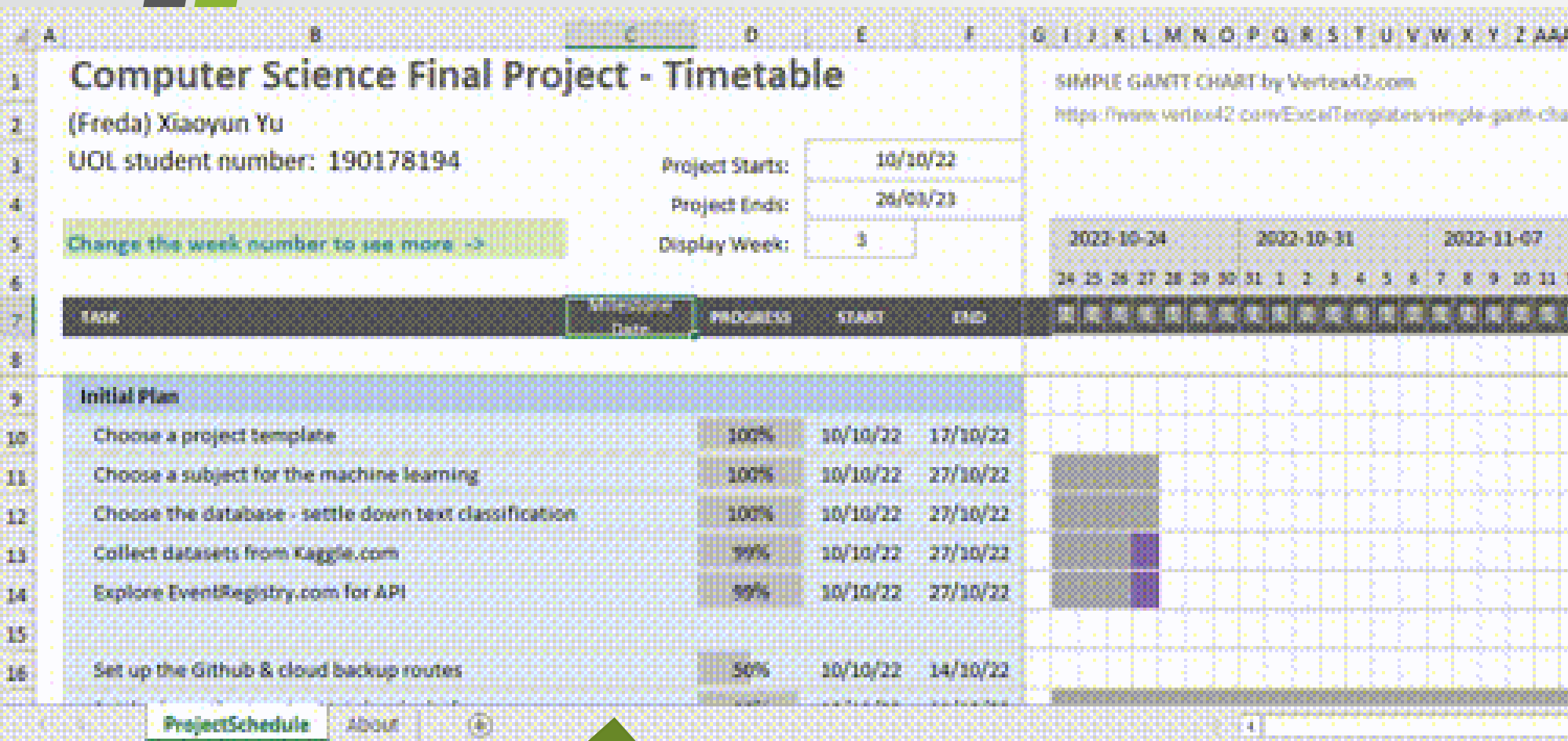
The best DNN model is a CNN model for which I get accuracy=0.633, fine-tuned by a pre-trained model mentioned by a video in Bilibili website.

- For other NLP models,
- (1) Classifier: SVC is No.1, MNB is No.2, both are impressive. Random Forest provides just a passing performance; while KNN must combine with TF-IDF vectorizer to be outstanding.
- (2) Vectorizer: TF-IDF is always excellent with all classifiers; Count Vectorizer almost the same but should not combine with KNN; Bigram performs always the worst, and should never combine with KNN.

Final Comparison

- It is interesting why SVC and MNB Classifiers outperform all the Neural Network models.
- TF-IDF and Count Vectorizer apparently help its model to get a high accuracy. Can I implement them into a NN as a layer?
- Inspired by LDA result, can I purify my subject-classification scheme by eliminating the classification based on research methods?
- Journal editors may adapt my model(s), but should manually scan for errors. Editors' work is irreplaceable.

Arrangement - Gantt Chart, Milestone Tasks



Milestone
dates

Milestone
tasks

Literature papers and books collected	Dec 01	Done
Literature Review file	Dec 20	Done before midterm
Project Design file	Dec 19	
Slides for pitch video	Dec 25	Done, on Dec 19
Pitch video for midterm	Dec 31	Done before midterm
Prototype: use a vectoriser and a classifier	Dec 10	Done, on Dec 09. Built 3 NLP models for SVC
Prototype: use > 5 NLP methods	Dec 31	
Prototype: use a small neural network	Dec 01	Done.
Prototype: use a larger neural network	Dec 28	Done, on Dec 09
*Tuning hyper-parameters: plan (grid search?)	Dec 15	Done, on Dec 09 by installing NNI, Feb 01-02 run GridSearchCV
Tuning hyper-parameters: practice	Dec 25	
Try LSTM	Feb 10	LSTM layer on Feb 03
Try CNN	Dec 17	Tried on Dec 09
*CNN learning	Jan 20	
*RNN learning	Jan 20	
Report: write methods intro	Jan 31	
Report: experiment record	Keep along all the way till the end	
GridSearchCV codes	Feb 05	Done on Feb 01-02
Build a strong CNN	Feb 10	Feb 03, 0.57 accuracy
Try ULMFIT (FastAI)	Feb 18	Feb 02-04 stuck
Try AWD-LSTM	Feb 18	Feb 03, stuck on cuda
Try Wikipedia2vec	Feb 18	Feb 05-06 too slow stuck
Try Bert-for-Task	Feb 18	Feb 06 reform data, meet utf-8 decoding error
At least one pre-training	Feb 13	?
Report: write implementation	Feb 12	
Report: write evaluations	Feb 12	
Report: write key techs	Feb 12	
Report: change references to ACM style	Mar 12	
Exam: prepare	Mar 04	
Video: slides	Feb 25	

Thank You for reviewing my project!

