RMBI4310 Project 1

Group 32

Simple Summary of the Project

- In this project, we have experimented mainly 4 different model architectures, including: 1 layer perceptron, CNN, RNN, and RNN+CNN.
- Among all, CNN performed the best, with all metrics passing the strong baseline, while 1 layer perceptron and RNN+CNN also performed quite well in validation accuracy.
- For our project submission, we submitted the CNN version, attached with other good performing models in the folder 'code-and-predictions-for-other-good-models'.

Model\Validation Performance	Macro-F1	Precision	Recall	Accuracy
1-Layer-Perceptron (added 'funny', 'cool', 'useful')	0.5422	0.5789	0.5305	0.6590
CNN (added 'funny', 'cool', 'useful')	0.5609	0.5688	0.5682	0.6620
RNN (Bi-directional LSTM)	0.4785	0.5101	0.4818	0.5995
RNN + CNN (dropout rate 0.4)	0.5105	0.5605	0.5114	0.6515
Strong Baseline	**0.5370**	**0.5509**	**0.5324**	**0.6500**

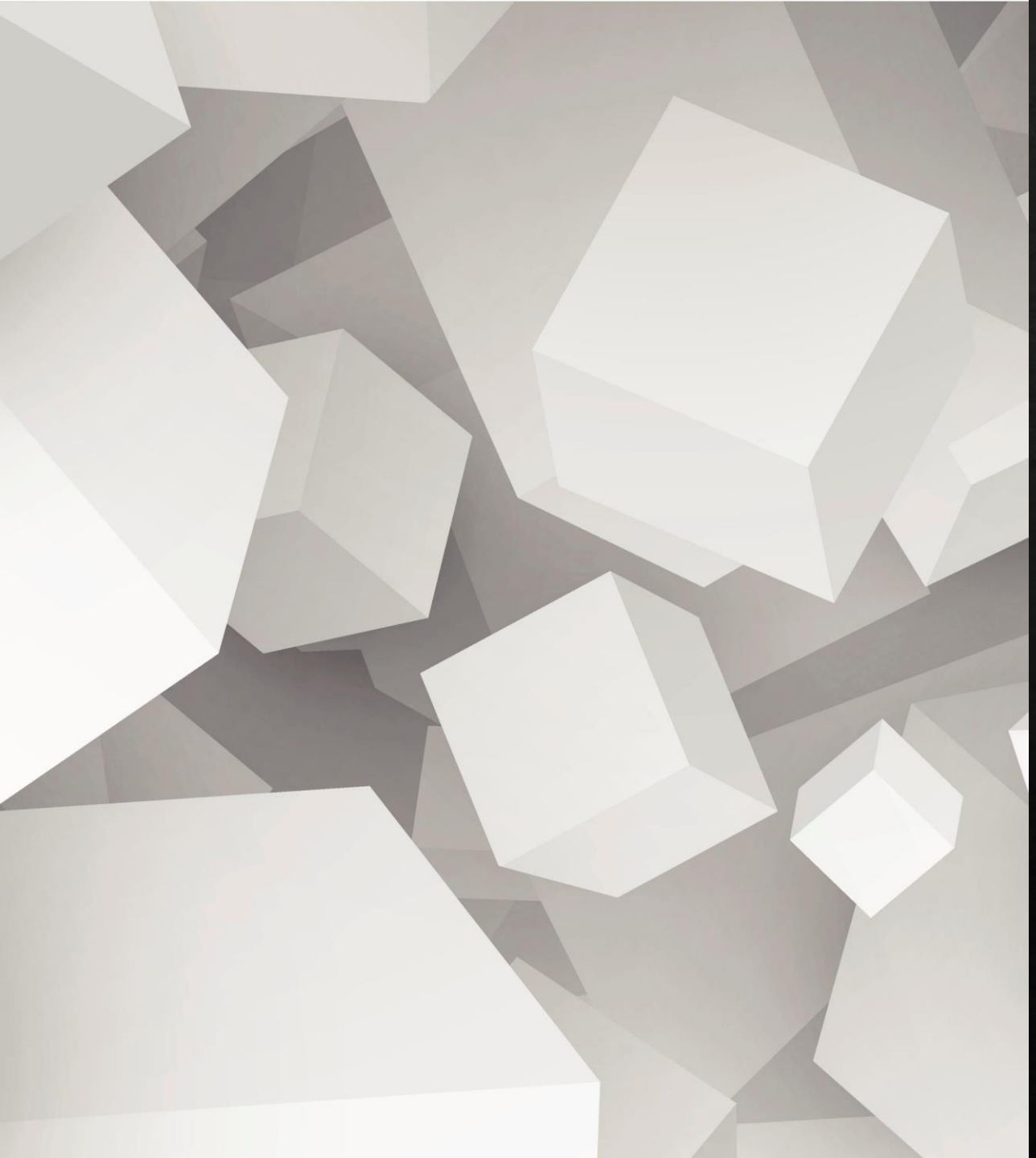


Table of Content

Submission Requirements

- Deadline: 7th April, 2022 (23:59)
- Submission:
  - Submit group32.zip file on Canvas
    - pred.csv (make sure can evaluate validation predictions on the data with help of evaluate.py)
    - PDF report (1-2 pages)
    - Code

Project Description – Sentiment Analysis

- Task: Rank the attitude (star) of the input text from 1 to 5
  - Classification OR Regression → Binary OR Ordinal Label
- Dataset:
  - Training: 18000 reviews
  - Validation: 2000 reviews
  - Test: 4000 reviews
- Baselines:

Validation Performance	Macro-F1	Precision	Recall	Accuracy
Weak baseline	0.4270	0.5420	0.4325	0.6135
Strong baseline	0.5370	0.5509	0.5324	0.6500

Project Description - Pipeline

- Data Loader
  - Load data from disk
- Data Pre-Processing
  - Data Analysis
  - Feature Extraction (e.g., Text data pre-processing)
- Learning
  - Classification via different classifiers
- Evaluation of Model
  - Using validation set
- Saving Predictions

Dataset - Attributes

Input Attributes:

- Business ID
- Cool
- Date
- Funny
- Review ID
- **Text**
- Useful
- User ID

Output Attributes:

- **Stars**

	A	B	C	D	E	F	G	H	I
1	business_id	cool	date	funny	review_id	stars	text	useful	user_id
2	JCZEK7wiazoM6xiq8YeZyw	1	16/1/2018 20:13	1	oxj0_2jKOqQFIWEYRjWi6g	5	I've been here a handful of times now and I've never	1	1fq-gL1i_8xKhc9VgOZDGw
3	ALn_0f-Usn3n0a9WBcjhhg	0	10/4/2018	0	gZITaUSvzBUijZvNGXO_Cg	1	The service was terrible. The food was just ok. Dessert w	0	wqG3PCf8ufXId2RG0oBufA
4	3tBRBsiTi6JJz3CJ7DcS_w	0	11/7/2014 19:08	0	ov2ohuP2bPJI35sscGGJpw	4	Alil pricey for the location but completly get the bang for	0	xgXVmyRpUZUwbgo519IqJw
5	eD6MH0tD1R3C1Qs1sH0wBg	0	28/4/2018 22:03	0	LFJGPIrbR7U_g3oavotkXg	1	Don't get your car washed here. Paid 11 and my car cam	1	KjhzP6W-6T7cZrPczcnKOg
6	T-TES2u1IA2THb8uBhNdCA	0	15/7/2015 17:21	0	hUoRKiGTnMV51R6pQSYovQ	5	Cute but tight. Not expensive and creative. I love the plac	0	CN5OQxL6FVT3nr7L2Ohm2w
7	K7lWdNUhCbcnEvI0NhGewg	0	9/4/2017 20:33	0	DiQufyOytFxWFRwx_J0VzQ	3	My first time to Vegas, however not my first time to a	1	iOuRNOcpE7hoL7v_ACNPwg
8	vl1aHp6pvnSmpoRQShWLjg	0	1/8/2017 23:10	0	wonq8-oCfdESM7UX0AN6IQ	5	Oh. My. God. I hurt my stomach but it was SO WORTH IT	0	LtdPQ-KS-eUoTDgCsjikyA
9	#NAME?	1	16/8/2010 2:49	0	Fvkz30VnBGJZtFQukvFYvg	4	Gigi's was around the corner from our hotel in Vegas, so	1	__2Xu2F0Z1gAodYpIdOsCQ
10	1FnwvtISSL0HxIdPLLow1g	0	29/8/2016 6:49	0	A_yYi4WSI9lC0I8FcBRg1A	3	this place is MEH... there's a weird smell at the doorway.	0	JSuHkTkKdaLAmtioY2KNtw
11	gJhMeq2nVH27tz8LqbD3eQ	0	5/7/2018 7:52	0	2Nh8AeeTHhkHFo-vs5anpw	5	This show is excellent. I'm so glad we went to see Raiding	0	ijCwAEmqzZZWlfQvKcBprw
12	BG8lvZoPTrPJqcvTh-ytzg	0	16/10/2010 1:23	0	mQSDNTdzTirmrTKcRMnxDw	4	RÃ©moulade sauce for the appizer crab cocktail was a fi	0	VHtB9pjUN2uDkgQukKtJrQ
13	Zh4RJ6mQ9RXjefij9oDueg	0	23/5/2013 19:35	0	YMI3V2bPnwM7aCGkoICx1A	4	Cheap way to get your car done, clean, not perfect but w	0	SSRrNLsHuQbDhmfCGdyXTw
14	FlJ3Add2GgFaIsWntgum0g	0	16/7/2009 6:01	0	HipPLFpsN1be-vv_zUpRpw	5	With sandwiches under 400 calories, and stellar service,	1	ddZffVYvHsCIhylU5UG-dA
15	GhWMl0J2gOOH61BhBMTAAA	0	5/2/2017 0:03	0	BYxBgw3ljuG75ndCMilNDg	5	First time here and it was awesome! I love how you can c	0	TMilGgbY0iUdiGulPFC3Xw

Data Analysis

Loading Data

- Ignore ‘business ID’, ‘user ID’, ‘review ID’, and ‘date’
  - Probably irrelevant to sentiment analysis

```
train_df
```

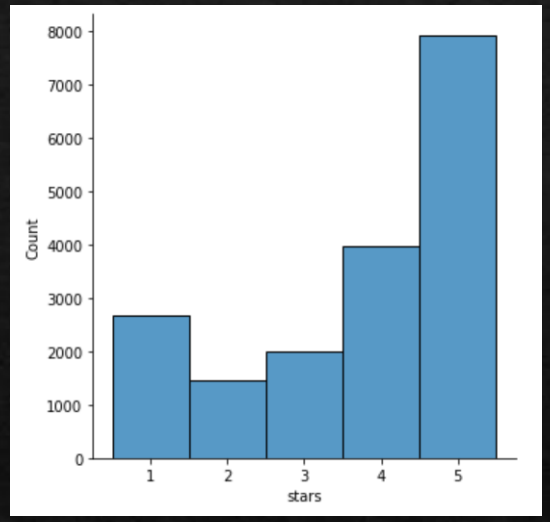
[4]:

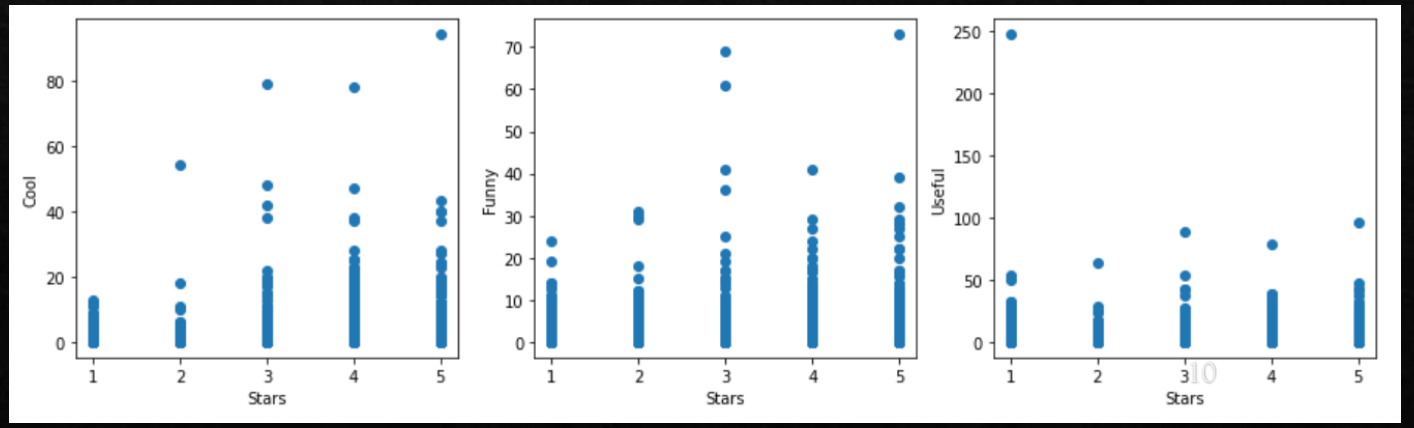
	cool	funny	useful	text	stars
0	1	1	1	I've been here a handful of times now and I've...	5
1	0	0	0	The service was terrible. The food was just ok...	1
2	0	0	0	Alil pricey for the location but completly get...	4
3	0	0	1	Don't get your car washed here. Paid 11 and my...	1
4	0	0	0	Cute but tight. Not expensive and creative. I ...	5
...	...	...	...	...	...
17995	0	0	0	DHL is not my friend. The instructions to clea...	1
17996	0	0	2	It's a nicely laid out shop with lots of jewel...	3
17997	1	1	5	Called the salon to advise the owner that my p...	1
17998	0	0	2	If you like authentic mexican seafood - than t...	4
17999	0	0	0	My husband and I stopped by for a mid-day bite...	5

18000 rows × 5 columns

Explorative Data Analysis

- Imbalance data
  - Most of the reviews are having '5 stars' and '4 stars'
  - '2 stars' and '3 stars' are rare
  - → Resampling maybe useful
- More 'useful' may means '1-3 stars' (i.e., fewer stars)
  - Review giving low stars may more likely to contain information about what is not doing well, so other people may found it useful to know

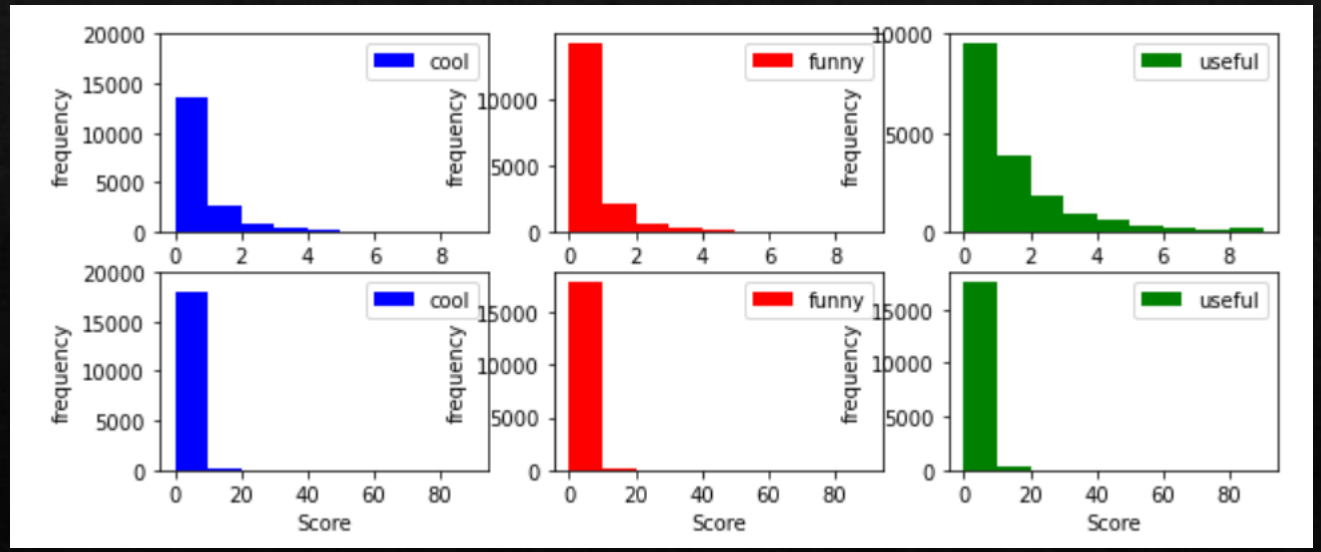




Explorative Data Analysis

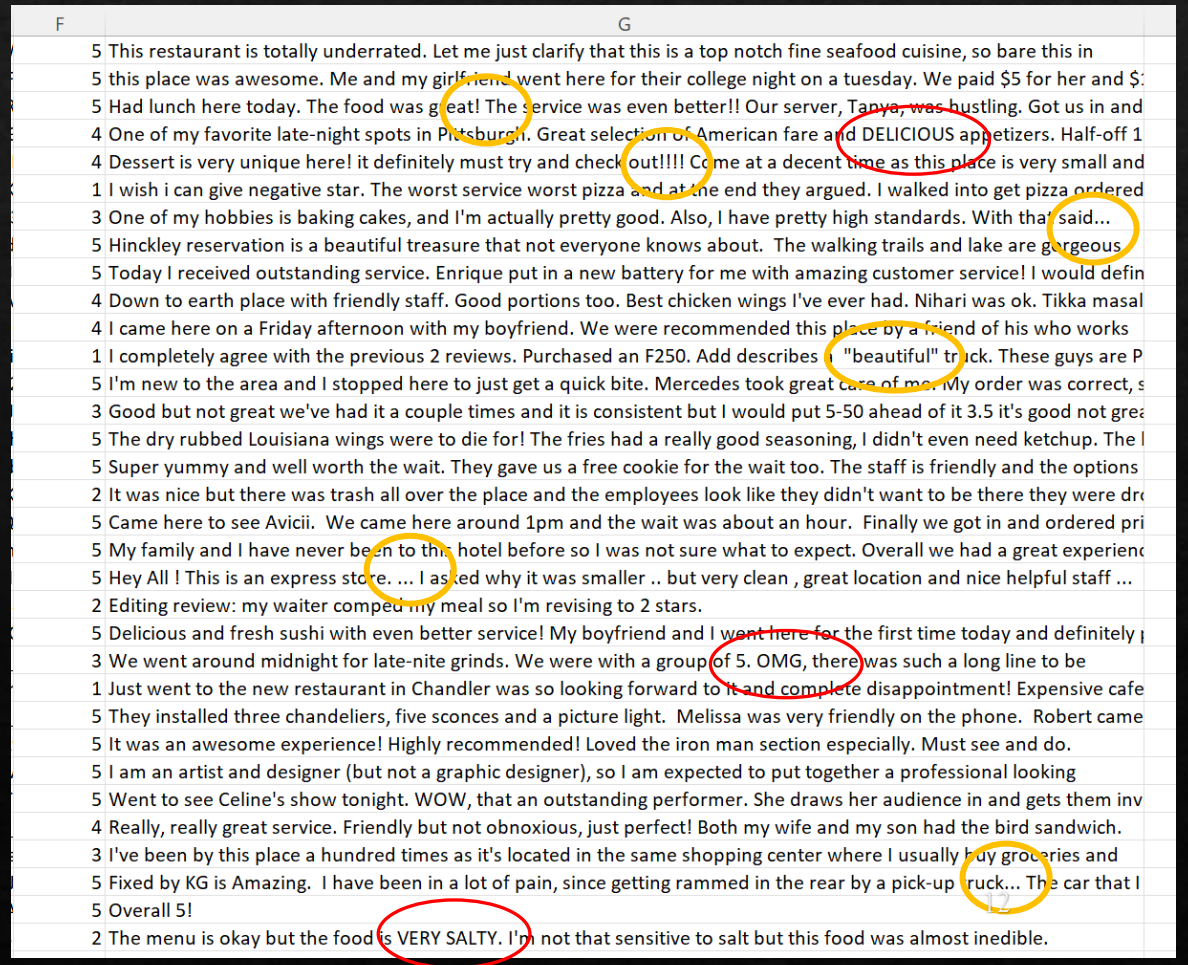
- On the left are the correlation matrix and histogram of 'cool', 'funny', and 'useful'
  - As shown in the matrix, 'cool' is having slightly positive relationship with 'stars'
  - Also, 'funny' and 'useful' are slightly negatively correlated with 'stars'
  - As shown in the graph, most (>17500/18000 for training set) scores for the three criteria lies between 0-10.
    - We will later discard value > 10 in our model feature input

	stars	cool	funny	useful
stars	1.000000	0.082473	-0.100597	-0.111312
cool	0.082473	1.000000	0.568251	0.535532
funny	-0.100597	0.568251	1.000000	0.434786
useful	-0.111312	0.535532	0.434786	1.000000



Explorative Data Analysis

- Upper Case is important information for sentiment analysis
  - E.g., DELICIOUS, HEAVEN, LEAST, VERY SALTY, WOW, OMG etc.
- Punctuation (probably could be capture with n-gram)
  - ! → emphasis/stress
    - E.g., Overpriced!!!!!, awesome experience!
    - Increase weights for words with ! ending?
  - … → negative?/positive?
    - E.g., great location and nice helpful staff…, So…
  - “” → focus
    - E.g., “beautiful”
    - Add weights on words with “”?
- Language
- Word/Sentence Count



F	G
5	This restaurant is totally underrated. Let me just clarify that this is a top notch fine seafood cuisine, so bare this in
5	this place was awesome. Me and my girlfriend went here for their college night on a tuesday. We paid $5 for her and $:
5	Had lunch here today. The food was great! The service was even better!! Our server, Tanya, was hustling. Got us in and
4	One of my favorite late-night spots in Pittsburgh. Great selection of American fare and DELICIOUS appetizers. Half-off 1
4	Dessert is very unique here! it definitely must try and check out!!!! Come at a decent time as this place is very small and
1	I wish i can give negative star. The worst service worst pizza and at the end they argued. I walked into get pizza ordered
3	One of my hobbies is baking cakes, and I'm actually pretty good. Also, I have pretty high standards. With that said…
5	Hinckley reservation is a beautiful treasure that not everyone knows about. The walking trails and lake are gorgeous
5	Today I received outstanding service. Enrique put in a new battery for me with amazing customer service! I would defin
4	Down to earth place with friendly staff. Good portions too. Best chicken wings I've ever had. Nihari was ok. Tikka masal
4	I came here on a Friday afternoon with my boyfriend. We were recommended this place by a friend of his who works
1	I completely agree with the previous 2 reviews. Purchased an F250. Add describes a "beautiful" truck. These guys are P
5	I'm new to the area and I stopped here to just get a quick bite. Mercedes took great care of me. My order was correct, s
3	Good but not great we've had it a couple times and it is consistent but I would put 5-50 ahead of it 3.5 it's good not grea
5	The dry rubbed Louisiana wings were to die for! The fries had a really good seasoning, I didn't even need ketchup. The l
5	Super yummy and well worth the wait. They gave us a free cookie for the wait too. The staff is friendly and the options
2	It was nice but there was trash all over the place and the employees look like they didn't want to be there they were dr
5	Came here to see Avicii. We came here around 1pm and the wait was about an hour. Finally we got in and ordered pri
5	My family and I have never been to this hotel before so I was not sure what to expect. Overall we had a great experienc
5	Hey All ! This is an express store. … I asked why it was smaller .. but very clean , great location and nice helpful staff …
2	Editing review: my waiter comped my meal so I'm revising to 2 stars.
5	Delicious and fresh sushi with even better service! My boyfriend and I went here for the first time today and definitely g
3	We went around midnight for late-nite grinds. We were with a group of 5. OMG, there was such a long line to be
1	Just went to the new restaurant in Chandler was so looking forward to it and complete disappointment! Expensive cafe
5	They installed three chandeliers, five sconces and a picture light. Melissa was very friendly on the phone. Robert came
5	It was an awesome experience! Highly recommended! Loved the iron man section especially. Must see and do.
5	I am an artist and designer (but not a graphic designer), so I am expected to put together a professional looking
5	Went to see Celine's show tonight. WOW, that an outstanding performer. She draws her audience in and gets them inv
4	Really, really great service. Friendly but not obnoxious, just perfect! Both my wife and my son had the bird sandwich.
3	I've been by this place a hundred times as it's located in the same shopping center where I usually buy groceries and
5	Fixed by KG is Amazing. I have been in a lot of pain, since getting rammed in the rear by a pick-up truck… The car that I
5	Overall 5!
2	The menu is okay but the food is VERY SALTY. I'm not that sensitive to salt but this food was almost inedible.

Explorative Data Analysis

- Tags & url in some reviews
  - Maybe could clean them up since probably they are not related to sentiment

12309	ohqeJybL1	0	########	0	308OxiDo!	2	I love Einstein's, however this location has lim	0	B-8emtACLpCXUXCvnoZm5Q
12310	GVHCYFp_	0	########	0	xfM-g1c6a	5	Want to SCREAM!! No other place will compa	0	CzKv_Cawgj2Q5n_QJvHd7w
12311	aZG9Hp8f	1	########	1	Kt2MjnFNI	3	Went Saturday for the first time in forever. 1	1	K8mRAWnTtGetOPqlMb9Sow
12312	vnvQ0lD9I	0	########	0	Y_lEsL8YL6	1	@HiltonGrandVac Not impressed at Elara Las	0	2vlpFv-7Tl_ulU3j-NWBOQ
12313	4OrIaoOw	0	########	0	j-Yt64ZYm	1	First I love my Toyota, but I have much	14	6au-XoKYas6YIRLDq8Klbg
12314	mU3vlAVz	0	########	0	qI7CCSWG	4	I gave Momofuku a chance again and after all	0	DLTEjTArAa2hlKUK32lZHw
14789	PZ0zeKE_F	0	########	0	AyjGb_azc	3	Unfortunately, great food but a terrible	1	71KaOJE5rZ2OuHYM71QEA
14790	WgBV0Sm	1	########	0	LnRtbtTeY	5	The staff in this location is busy but good.	0	q_GNUKJXBlMnuezCaxTNOw
14791	GtipraWsv	0	########	0	EyaZ02RoJ	5	#180	0	4CWCIaZl-ceMmDuVLjDxUQ
14792	20HWqSal	0	########	0	kq3opgTH	5	This place is excellent!!!! Amazing prices and s	1	mz-4Ehi8yNz7o-WosKzwJA

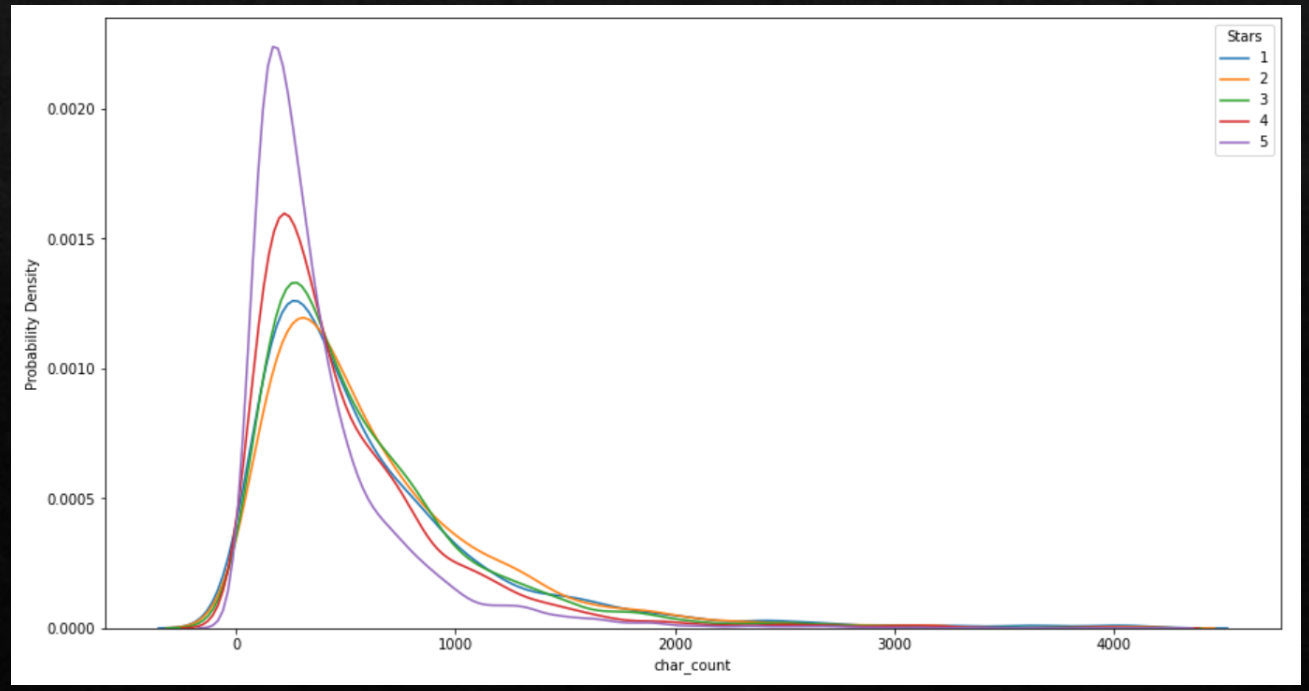
Explorative Data Analysis

- Consider the followings (do not have time to try add them in our model):
  - Character Count: sum the number of characters in each token
  - Word Count: count the number of tokens in the text
  - Sentence Count: count the number of sentences (separated by a period '.')
  - Average Word Length: character count/word count
  - Average Sentence Length: word count/sentence count

	cool	funny	useful	text	stars	word_count	char_count	sentence_count	avg_word_length	avg_sentence_length
0	1	1	1	I've been here a handful of times now and I've...	5	137	563	10	4.109489	13.70
1	0	0	0	The service was terrible. The food was just ok...	1	18	81	4	4.500000	4.50
2	0	0	0	Alil pricey for the location but completly get...	4	18	78	1	4.333333	18.00
3	0	0	1	Don't get your car washed here. Paid 11 and my...	1	37	142	4	3.837838	9.25
4	0	0	0	Cute but tight. Not expensive and creative. I ...	5	34	155	5	4.558824	6.80

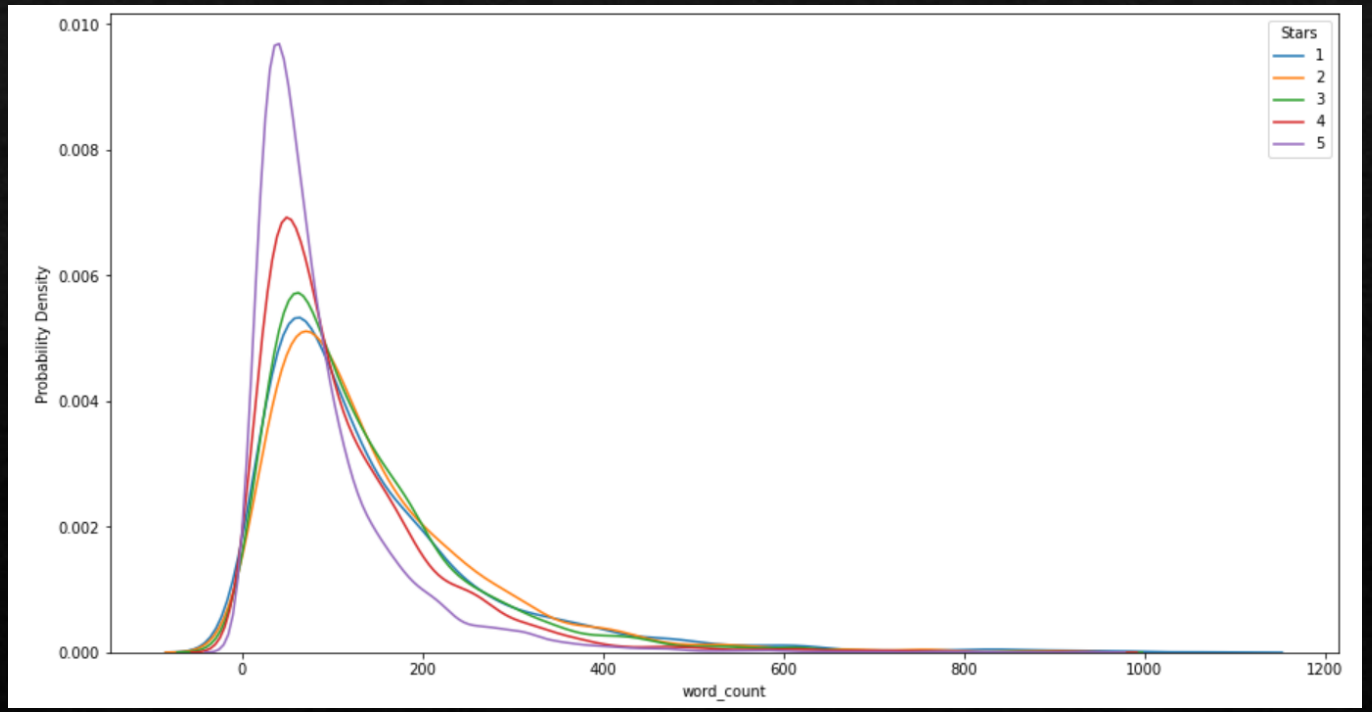
Explorative Data Analysis

- Character Count
  - Different distribution – possible feature for the model



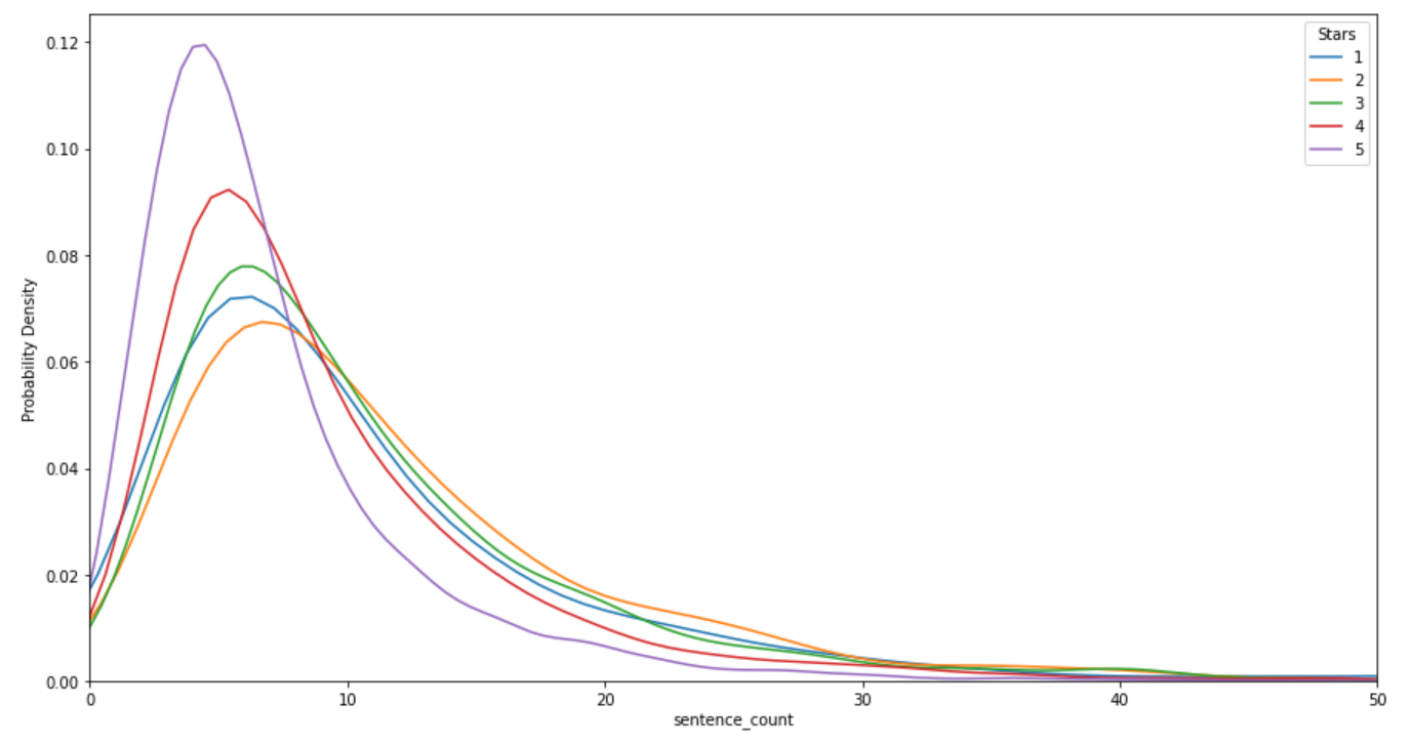
Explorative Data Analysis

- Word Count
  - Different distribution – possible feature for the model



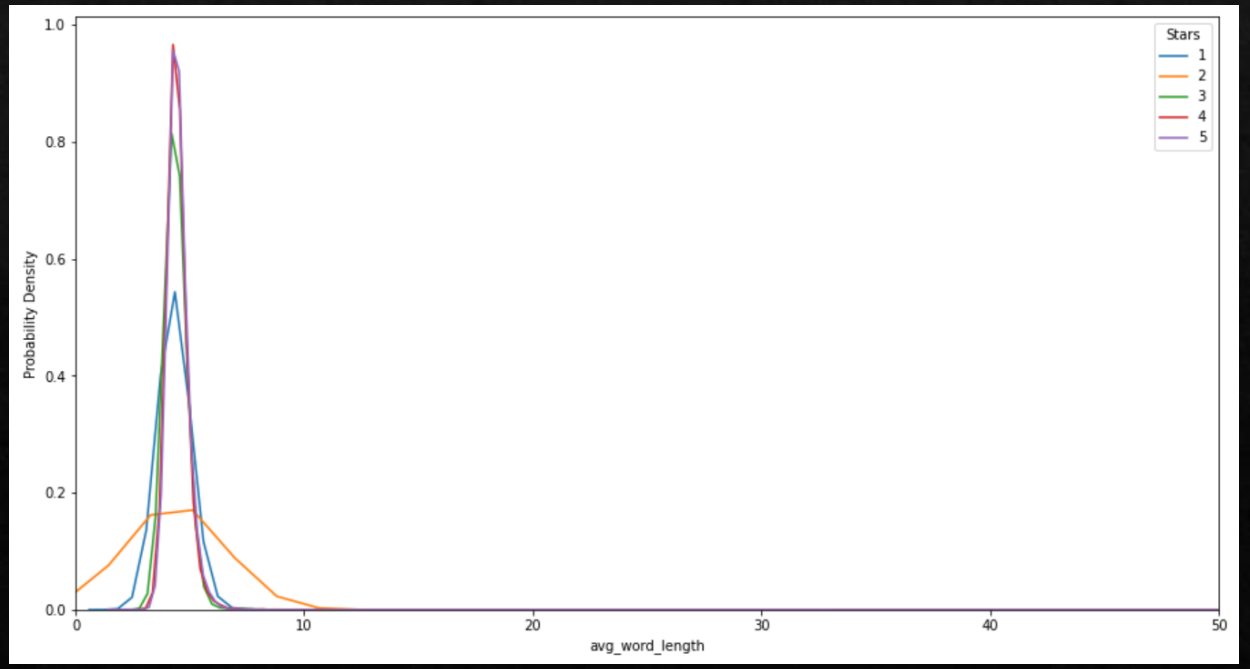
Explorative Data Analysis

- Sentence Count
  - Different distribution – possible feature for the model



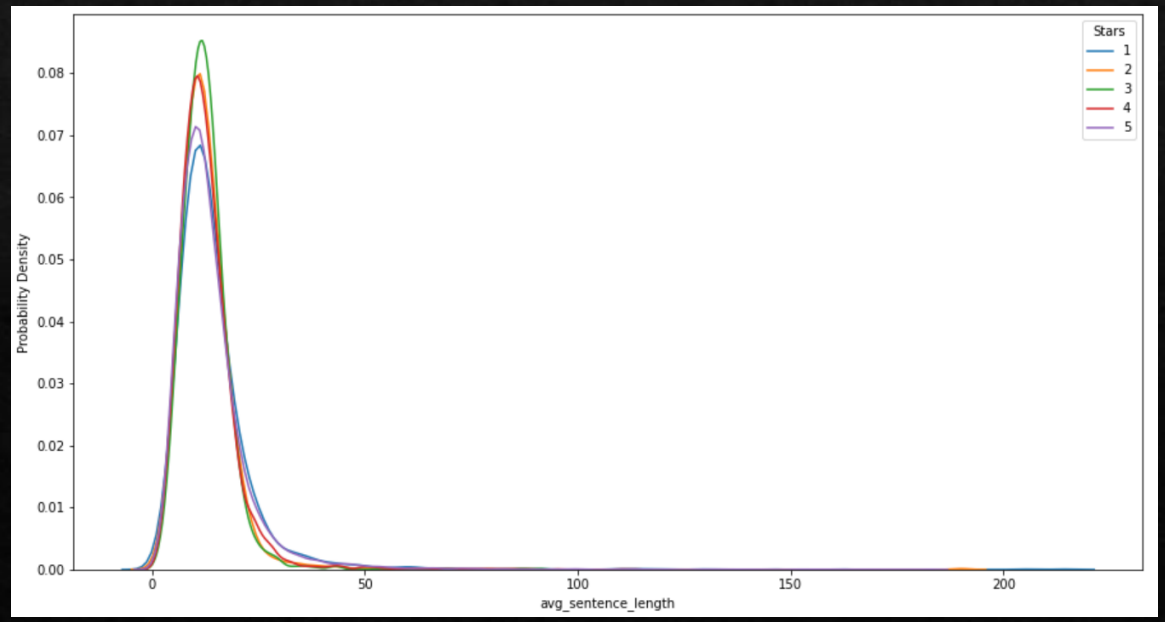
Explorative Data Analysis

- Average Word Length
  - Different distribution – possible feature for the model



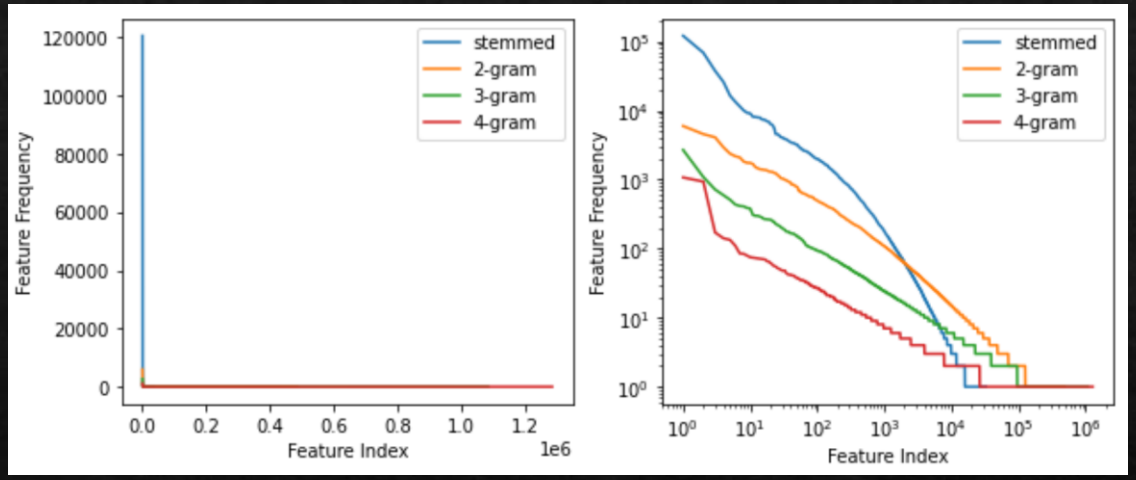
Explorative Data Analysis

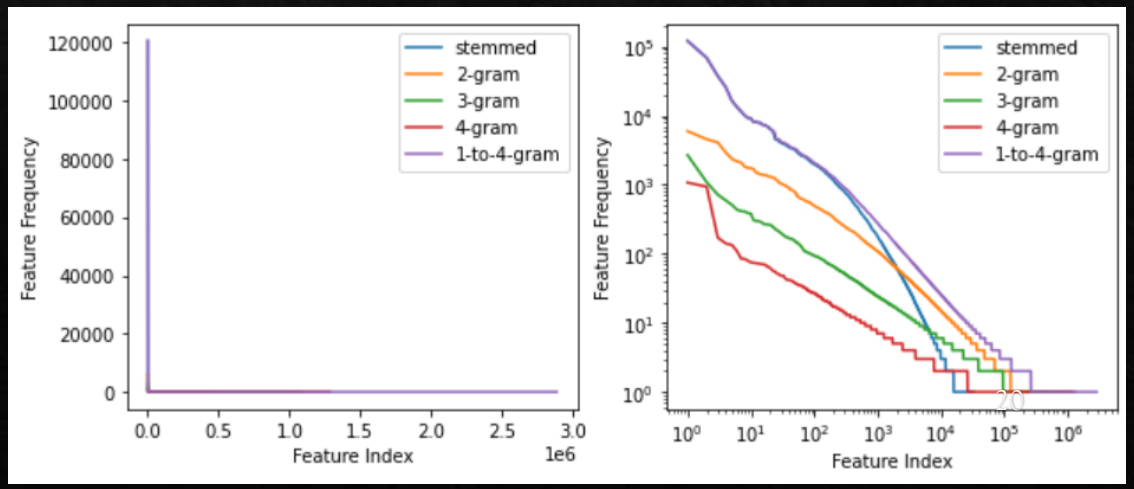
- Average Sentence Length
  - Similar Distribution



Explorative Data Analysis

- N-gram
  - 1-gram:
    - Feature size: 32647
    - Around 50% of features have low frequency
  - 2-gram:
    - Feature size: 484785
    - Around 75% of features have low frequency
  - 3-gram:
    - Feature size: 1083424
    - Around 90% of features have low frequency
  - 4-gram:
    - Feature size: 1284005
    - Around 98% of features have low frequency
  - 1-to-4-gram (all-gram):
    - Feature size: 2884861
    - Around 96% of features have low frequency





Explorative Data Analysis

- N-gram top features:
  - Top features are mostly punctuation, as mentioned before, punctuation could be an important feature to classify sentiment
  - Including all-gram would be better since the number of punctuation, e.g., !!!!, could represent stronger sentiment

```
stemmed features top 10: ['.', ',', 'wa', '!', 'thi', "n't", "'s", 'place', 'food', 'good']
2-gram features top 10: ['! !', 'thi place', '. wa', ', wa', '. thi', ') .', ". 's", 'wa veri', 'good .', '. also']
3-gram features top 10: ['! ! !', '? ? ?', 'thi place .', '. servic wa', '. food wa', '. thi place', 'love thi place', '. ho
wev ,', '. highli recommend', ". ca n't"]
4-gram features top 10: ['! ! ! !', '? ? ? ?', '* * * *', 'love thi place !', 'love thi place .', ". ca n't wait", "ca n't g
o wrong", "ca n't wait go", '. highli recommend thi', '. highli recommend .']
```

Explorative Data Analysis

- Choosing Minimum Frequency
  - From previous discussion, we concluded that for 1-gram, 2-gram, 3-gram, 4-gram, and all-gram, 50%, 75%, 90%, 98%, 96% of the features are having low frequency
  - Therefore, we find the corresponding frequency and treat it as the minimum frequency for each feature dictionary
  - → 2, 3, 2, 3, 4 for 1-gram, 2-gram, 3-gram, 4-gram, and all-gram

```
# XXth percentile got from the graph above
print("1-gram count at 50th percentile: ", y_stemmed[16322])
print("2-gram count at 75th percentile: ", y_2gram[121195])
print("3-gram count at 90th percentile: ", y_3gram[108341])
print("4-gram count at 98th percentile: ", y_4gram[25679])
print("1-to-4-gram count at 96th percentile: ", y_allgram[115393])
# Therefore, set min_freq as 2, 3, 2, 3, 4 for 1-gram, 2-gram, 3-gram, 4-gram, and 1-to-4-gram
```

```
1-gram count at 50th percentile:  1
2-gram count at 75th percentile:  2
3-gram count at 90th percentile:  1
4-gram count at 98th percentile:  2
1-to-4-gram count at 96th percentile:  3
```

1-Layer Perceptron

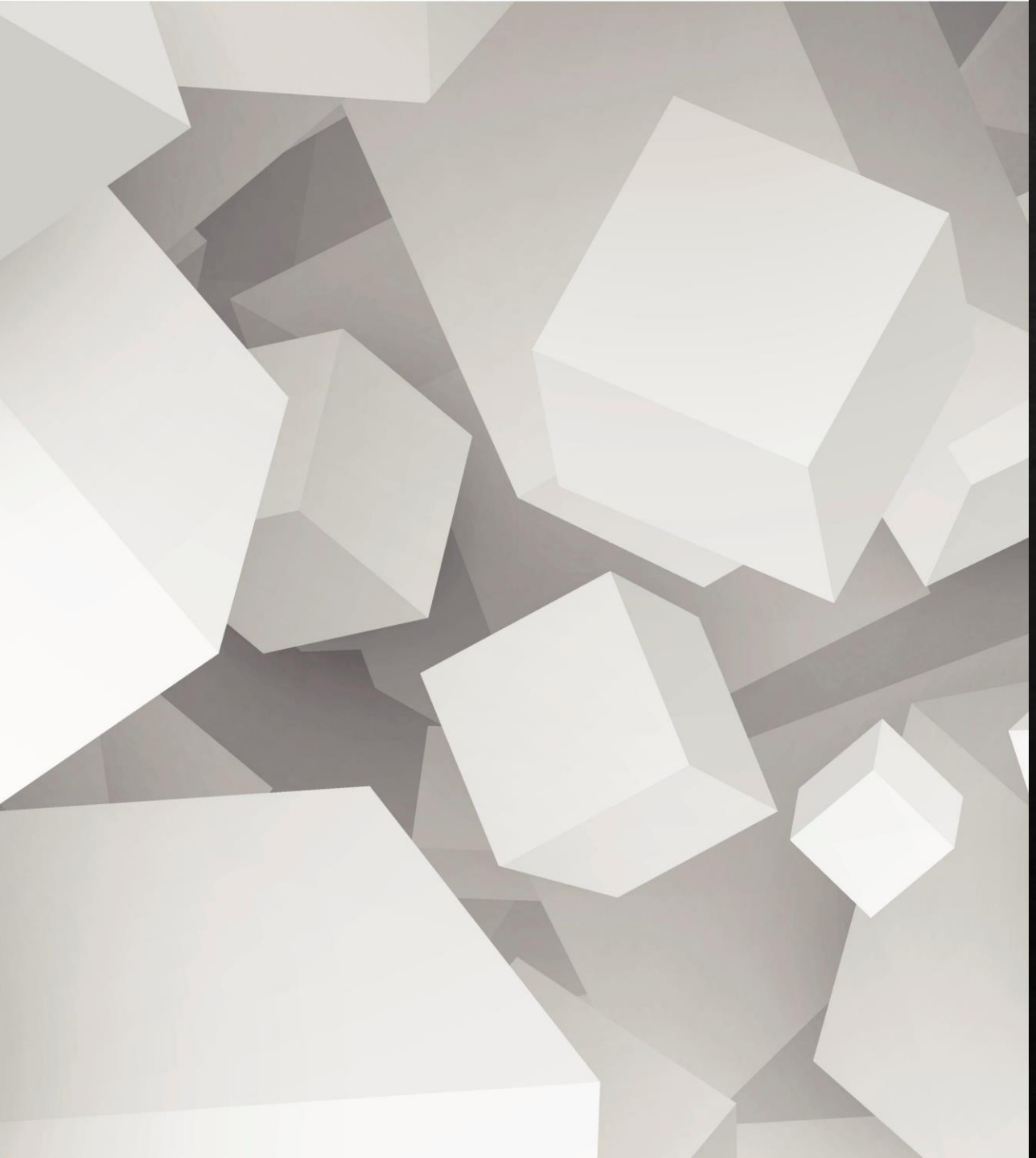


Table of Content

Classifier

- 1 dense layer
- Optimizers:
  - SGD
  - RMSprop
  - Adam
- Loss
  - Categorical_crossentropy
- Activation Function
  - Softmax
- Learning Rate: 0.1
- L2 Regularization: 0.005

Build Classifier

```
import keras as K
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
from keras import metrics
from tensorflow.keras.optimizers import SGD
import tensorflow

def build_classifier(input_size, output_size,
                     l2_reg=0.0,
                     optimizer="SGD",
                     learning_rate=0.1):

    model = Sequential()

    # the projection layer
    model.add(Dense(output_size,
                    activation="softmax",
                    input_dim=input_size,
                    kernel_initializer=K.initializers.he_normal(seed=0),
                    bias_initializer="zeros",
                    kernel_regularizer=K.regularizers.l2(l2_reg)))

    # set the loss, the optimizer, and the metric
    if optimizer == "SGD":
        optimizer = K.optimizers.SGD(lr=learning_rate)
    elif optimizer == "RMSprop":
        optmizer = K.optimizers.RMSprop(learning_rate=learning_rate)
    elif optimizer == "Adam":
        optmizer = K.optimizers.Adam(learning_rate=learning_rate)
    else:
        raise NotImplementedError
    model.compile(loss="categorical_crossentropy", optimizer=optimizer, metrics=['accuracy', tensorflow.keras.metrics.Precisi

    return model
```

Text Feature Construction – Initial Settings

- Simply one-hot vector

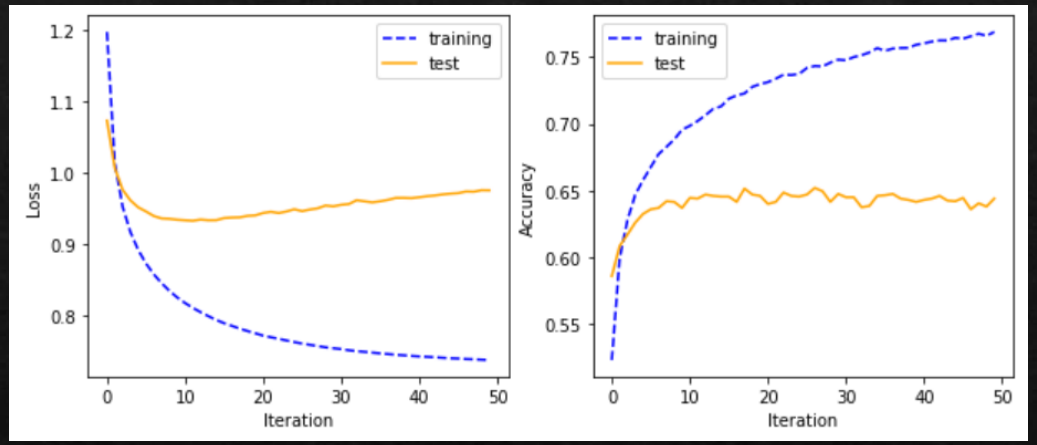
```
# build the feature list
train_feats = list()
for i in range(train_size):
    train_feats.append(train_tokens[i])

valid_feats = list()
for i in range(valid_size):
    valid_feats.append(valid_tokens[i])

# build the feature dict
feats = set()
# collect all features
for f in train_feats:
    feats.update(f)
print("Size of features:", len(feats))
# build a mapping from features to indices
feats_dict = dict(zip(feats, range(len(feats))))

train_feats_matrix = np.vstack([get_onehot_vector(f, feats_dict) for f in train_feats])
valid_feats_matrix = np.vstack([get_onehot_vector(f, feats_dict) for f in valid_feats])
train_label_matrix = tensorflow.keras.utils.to_categorical(train_labels-1, num_classes=5)
valid_label_matrix = tensorflow.keras.utils.to_categorical(valid_labels-1, num_classes=5)
```

```
Size of features: 32647
```

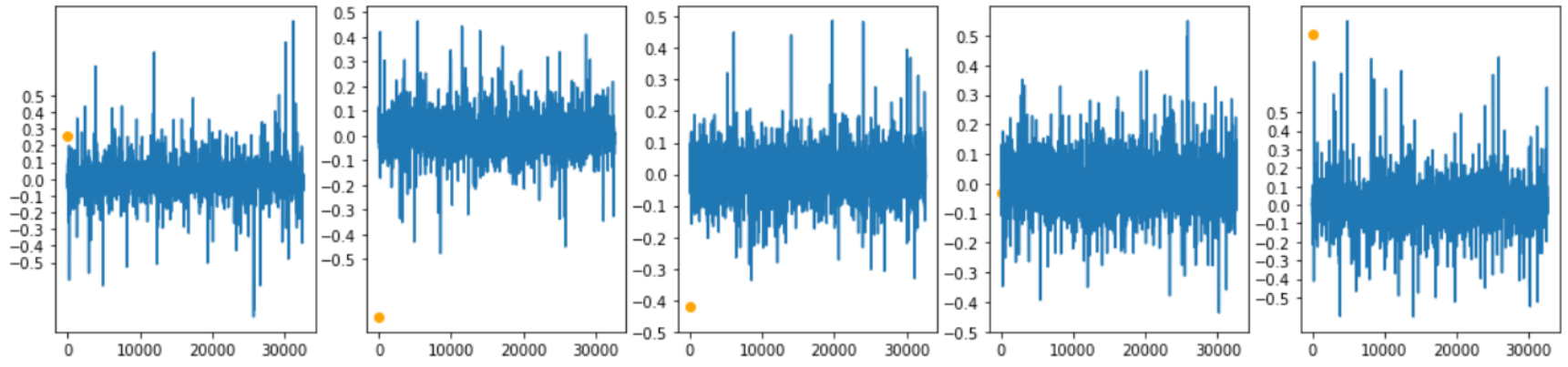
Selecting Optimizer

SGD

```
Train loss:  0.7497
Train accuracy:  0.7479
Train precision:  0.8440
Train recall:  0.6063
Train F1:  0.7057

Validation loss:  0.9483
Validation accuracy:  0.6520
Validation precision:  0.7310
Validation recall:  0.5135
Validation F1:  0.6032
```

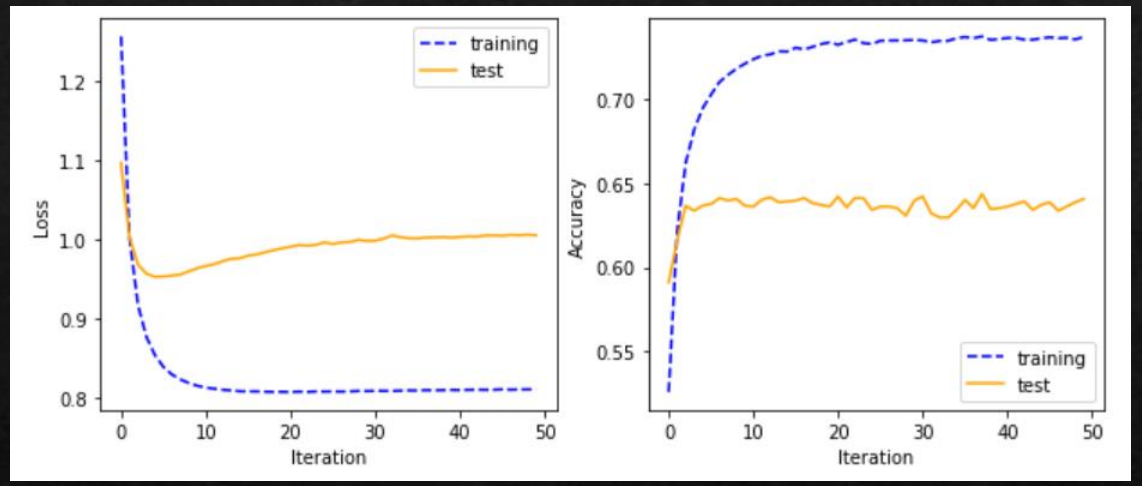



```
dense/kernel:0 (32647, 5)
dense/bias:0 (5,)
```

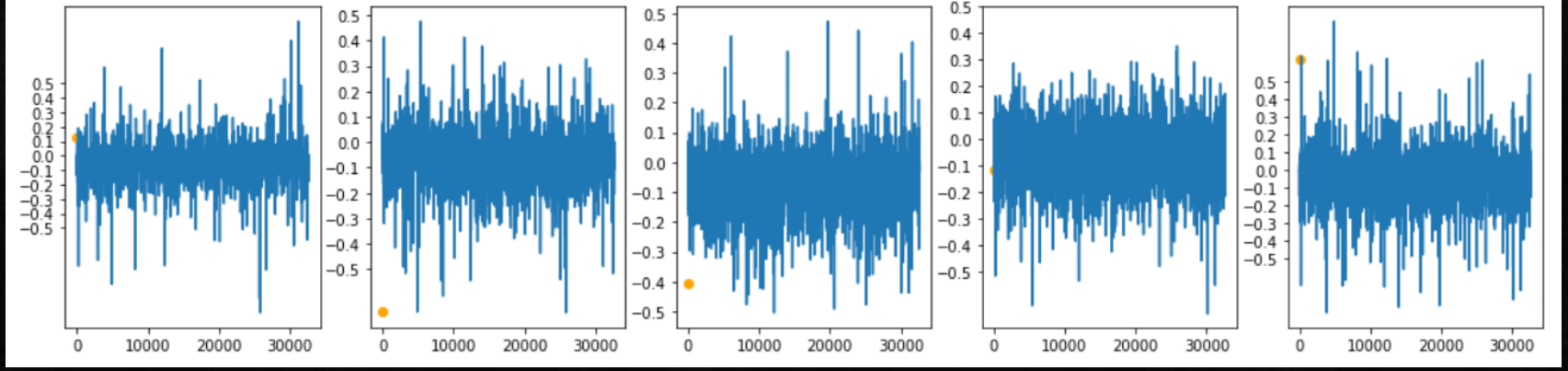


RMSprop

```
Train loss:  0.7932
Train accuracy:  0.7446
Train precision:  0.8337
Train recall:  0.6128
Train F1:  0.7064

Validation loss:  1.0028
Validation accuracy:  0.6435
Validation precision:  0.7269
Validation recall:  0.5270
Validation F1:  0.6110
```

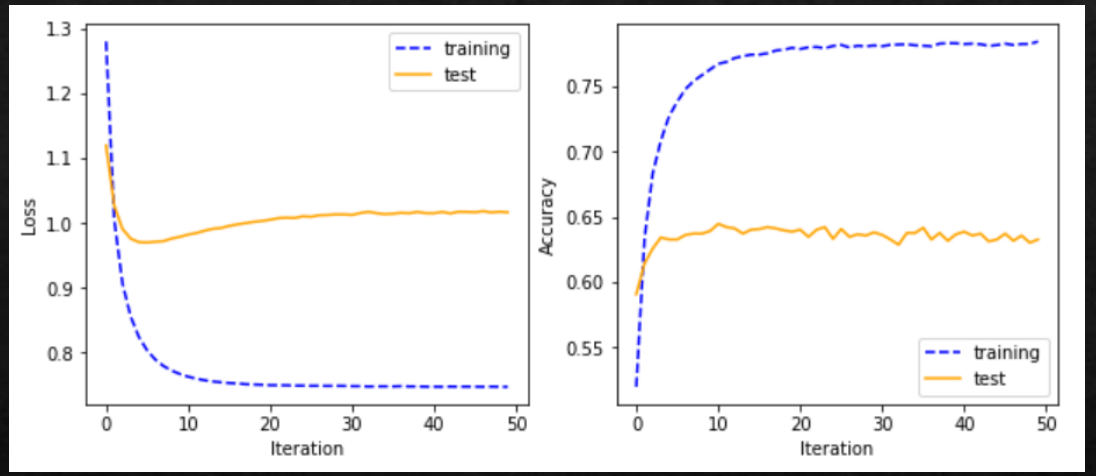



```
dense_1/kernel:0 (32647, 5)
dense_1/bias:0 (5,)
```

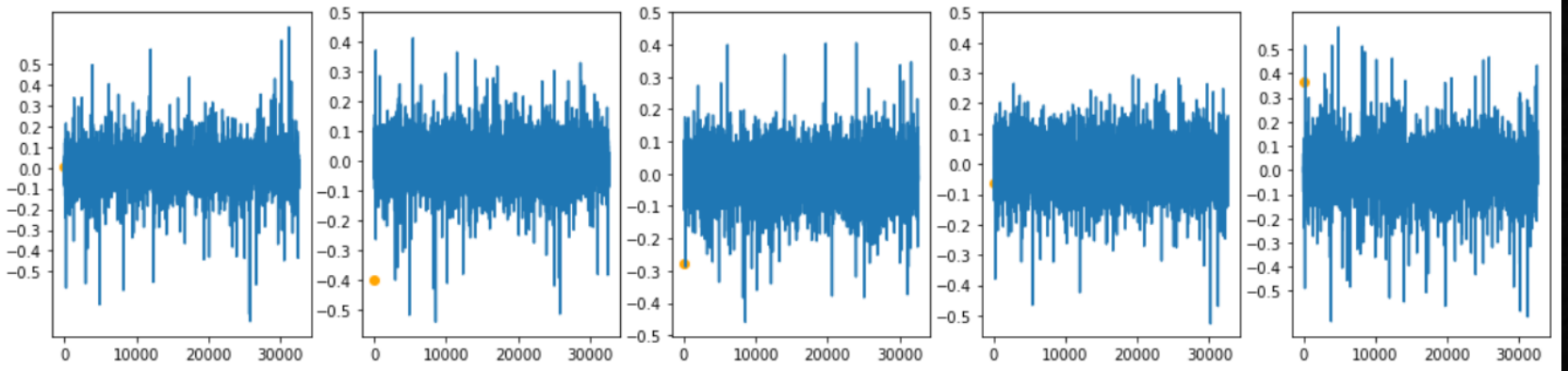


Adam

```
Train loss:  0.7377
Train accuracy:  0.7854
Train precision:  0.8806
Train recall:  0.6317
Train F1:  0.7356

Validation loss:  0.9822
Validation accuracy:  0.6445
Validation precision:  0.7341
Validation recall:  0.4970
Validation F1:  0.5927
```

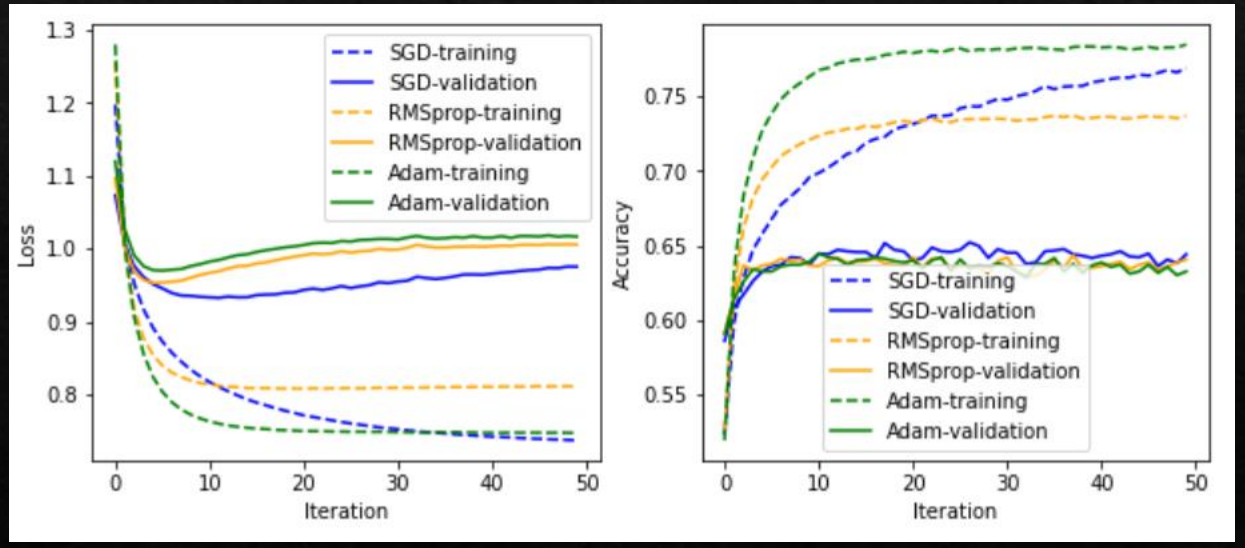



```
dense_2/kernel:0 (32647, 5)
dense_2/bias:0 (5,)
```



Comparison

- They have similar performance on validation set
- SGD is slightly better with best validation accuracy = 0.6520

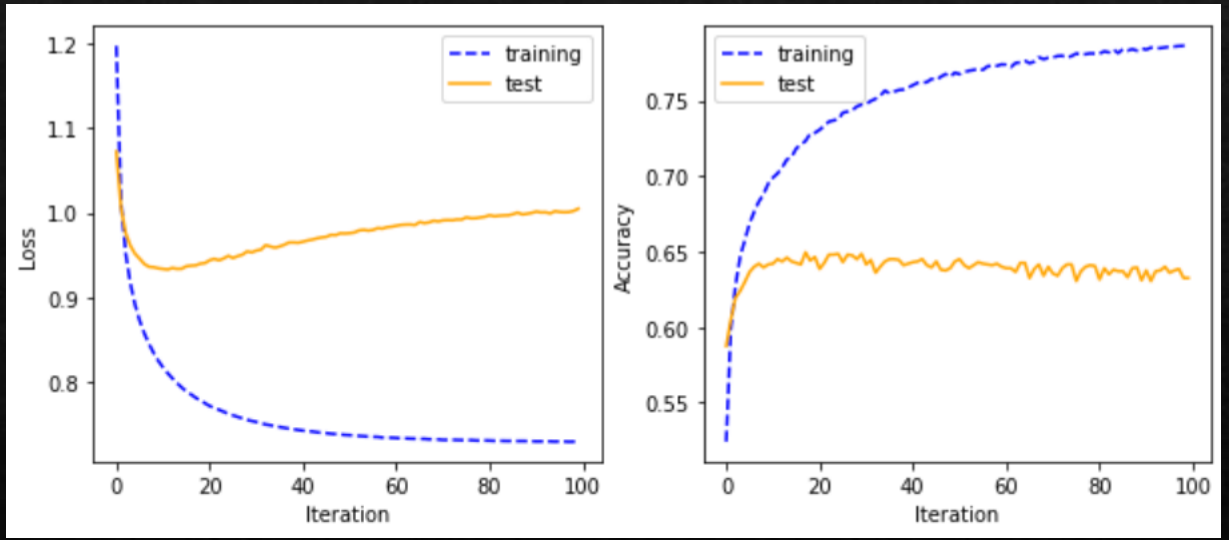



```
Summary:

SGD validation accuracy: 0.6520
RMSprop validation accuracy: 0.6435
Adam validation accuracy: 0.6445

SGD validation precision: 0.7310
RMSprop validation precision: 0.7269
Adam validation precision: 0.7341

SGD validation recall: 0.5135
RMSprop validation recall: 0.5270
Adam validation recall: 0.4970

SGD validation f1: 0.6032
RMSprop validation f1: 0.6110
Adam validation f1: 0.5927
```

N-Gram

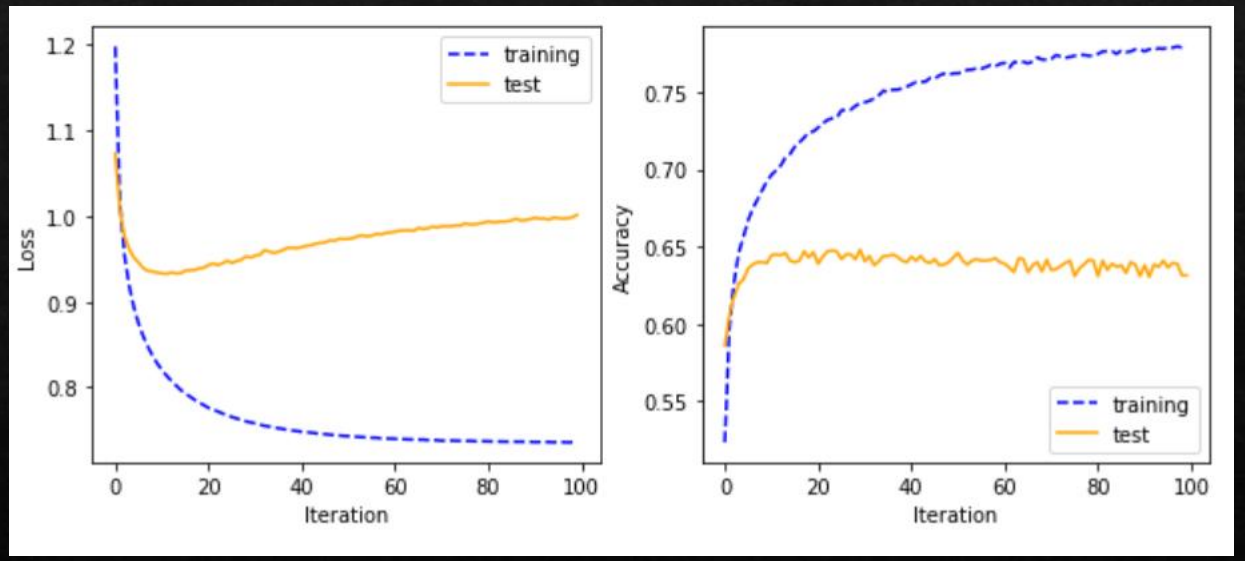
1-gram

```
Train loss:  0.7706
Train accuracy:  0.7317
Train precision:  0.8350
Train recall:  0.5790
Train F1:  0.6838

Validation loss:  0.9377
Validation accuracy:  0.6495
Validation precision:  0.7416
Validation recall:  0.4995
Validation F1:  0.5970
```



High-Frequency 1-gram

```
Train loss:  0.7506
Train accuracy:  0.7482
Train precision:  0.8419
Train recall:  0.6075
Train F1:  0.7057

Validation loss:  0.9517
Validation accuracy:  0.6480
Validation precision:  0.7329
Validation recall:  0.5145
Validation F1:  0.6046
```



Comparison – 1-gram vs hf-1-gram

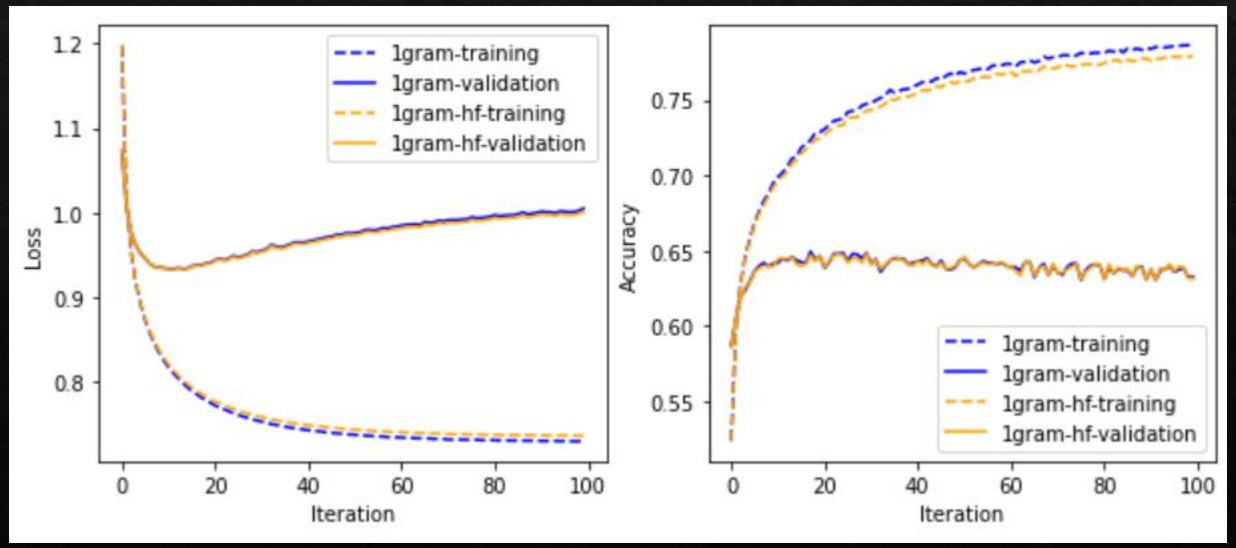
- Similar performance – simply 1-gram is slightly better

1-gram

```
Train loss:  0.7706
Train accuracy:  0.7317
Train precision:  0.8350
Train recall:  0.5790
Train F1:  0.6838

Validation loss:  0.9377
Validation accuracy:  0.6495
Validation precision:  0.7416
Validation recall:  0.4995
Validation F1:  0.5970
```

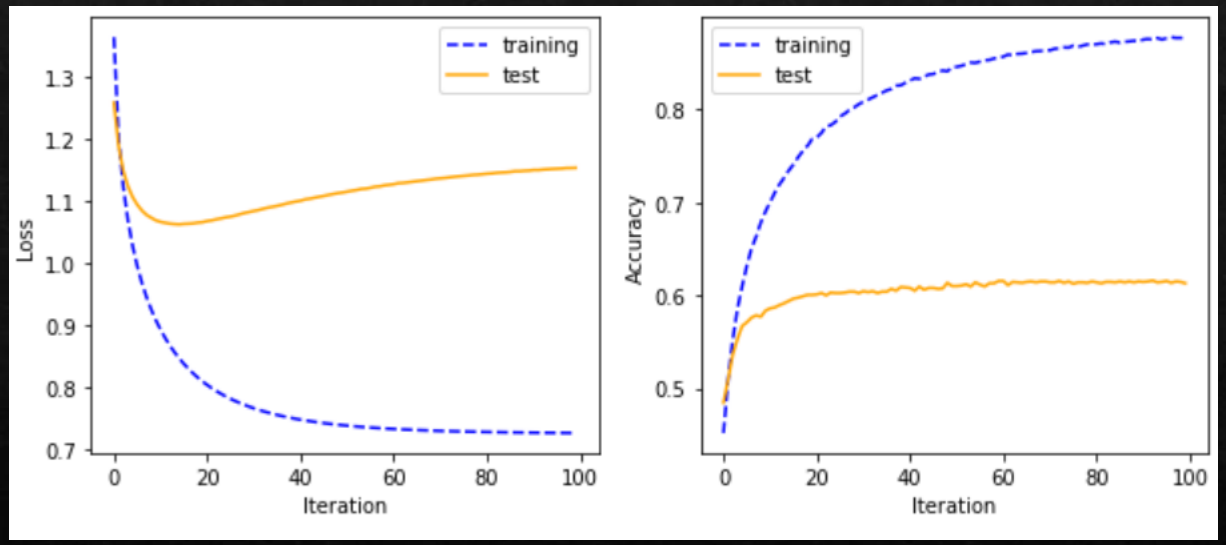
High Frequency 1-gram

```
Train loss:  0.7506
Train accuracy:  0.7482
Train precision:  0.8419
Train recall:  0.6075
Train F1:  0.7057

Validation loss:  0.9517
Validation accuracy:  0.6480
Validation precision:  0.7329
Validation recall:  0.5145
Validation F1:  0.6046
```



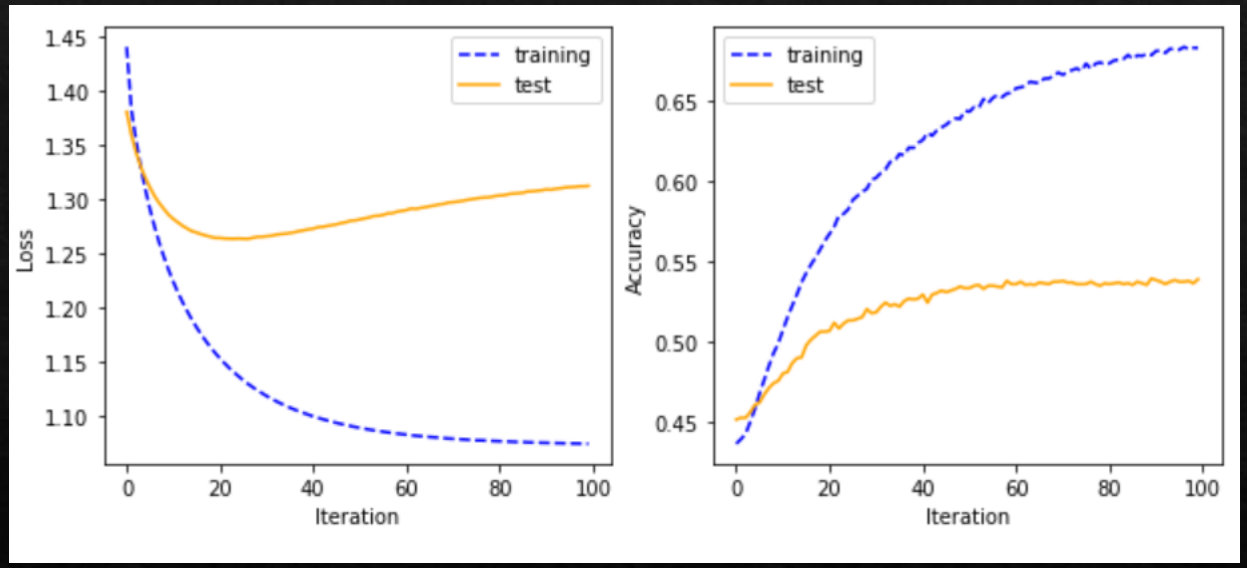
High-Frequency 2-gram

```
Train loss:  0.7219
Train accuracy:  0.8790
Train precision:  0.9604
Train recall:  0.7339
Train F1:  0.8320

Validation loss:  1.1510
Validation accuracy:  0.6160
Validation precision:  0.7386
Validation recall:  0.4520
Validation F1:  0.5608
```



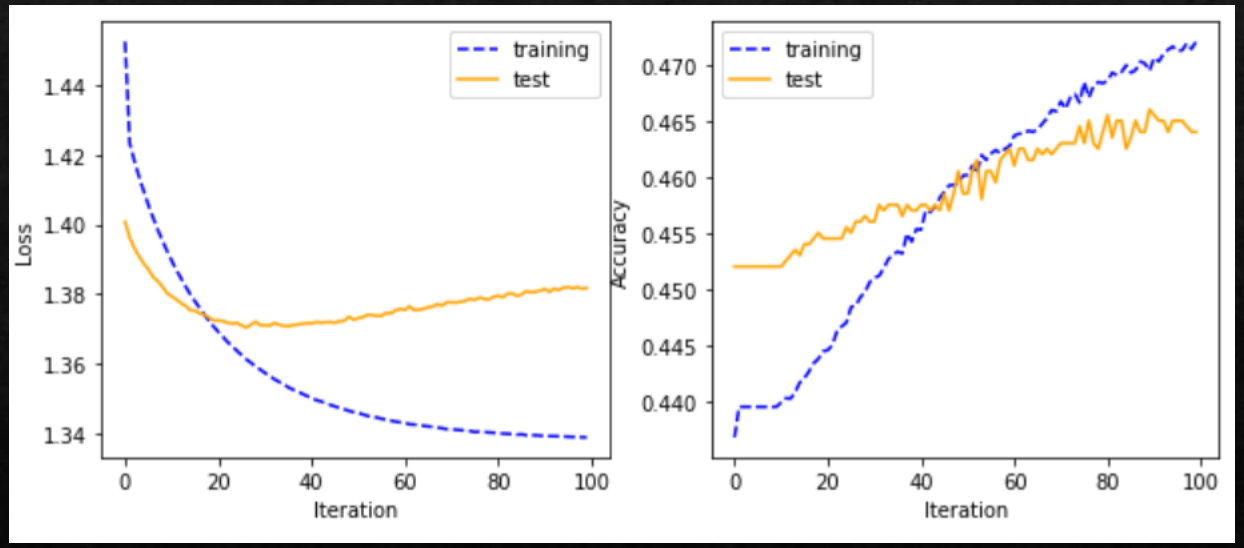
High-Frequency 3-gram

```
Train loss:  1.0724
Train accuracy:  0.6876
Train precision:  0.9349
Train recall:  0.4109
Train F1:  0.5709

Validation loss:  1.3082
Validation accuracy:  0.5395
Validation precision:  0.7724
Validation recall:  0.2715
Validation F1:  0.4018
```



High-Frequency 4-gram

```
Train loss:  1.3385
Train accuracy:  0.4732
Train precision:  0.8840
Train recall:  0.1033
Train F1:  0.1849

Validation loss:  1.3809
Validation accuracy:  0.4660
Validation precision:  0.7868
Validation recall:  0.0775
Validation F1:  0.1411
```



Comparison – 1-/2-/3-/4-gram

- High Frequency 1-gram is better

1-gram

```
Train loss:  0.7506
Train accuracy:  0.7482
Train precision:  0.8419
Train recall:  0.6075
Train F1:  0.7057

Validation loss:  0.9517
Validation accuracy:  0.6480
Validation precision:  0.7329
Validation recall:  0.5145
Validation F1:  0.6046
```

2-gram

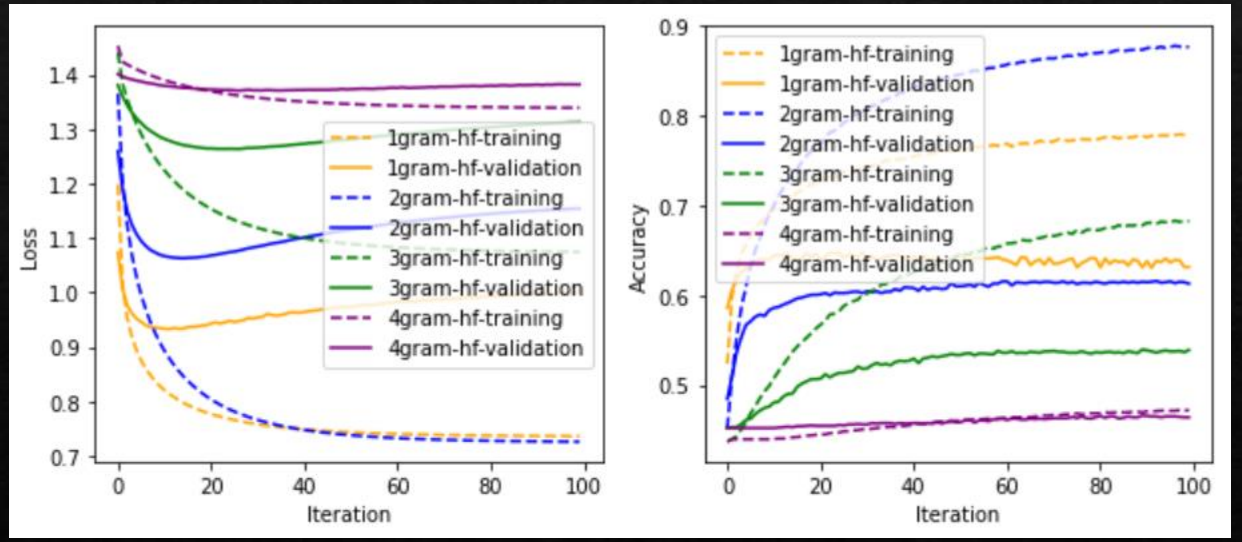
```
Train loss:  0.7219
Train accuracy:  0.8790
Train precision:  0.9604
Train recall:  0.7339
Train F1:  0.8320

Validation loss:  1.1510
Validation accuracy:  0.6160
Validation precision:  0.7386
Validation recall:  0.4520
Validation F1:  0.5608
```

3-gram

```
Train loss:  1.0724
Train accuracy:  0.6876
Train precision:  0.9349
Train recall:  0.4109
Train F1:  0.5709

Validation loss:  1.3082
Validation accuracy:  0.5395
Validation precision:  0.7724
Validation recall:  0.2715
Validation F1:  0.4018
```

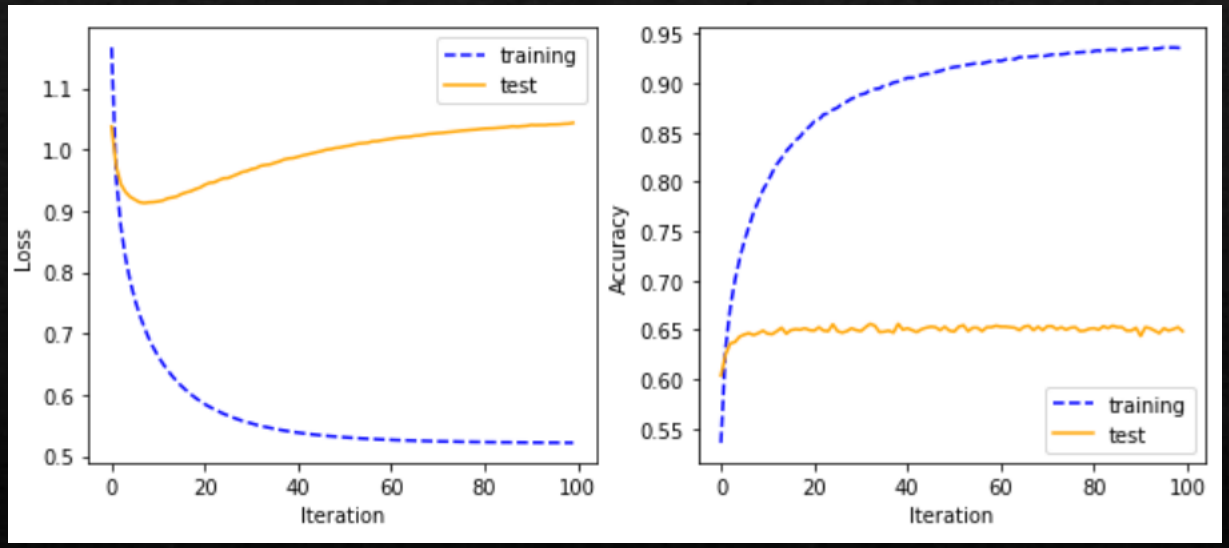
4-gram

```
Train loss:  1.3385
Train accuracy:  0.4732
Train precision:  0.8840
Train recall:  0.1033
Train F1:  0.1849

Validation loss:  1.3809
Validation accuracy:  0.4660
Validation precision:  0.7868
Validation recall:  0.0775
Validation F1:  0.1411
```



High-Frequency 1-to-4-gram (all-gram)

```
Train loss:  0.5411
Train accuracy:  0.8993
Train precision:  0.9507
Train recall:  0.7989
Train F1:  0.8683

Validation loss:  0.9733
Validation accuracy:  0.6560
Validation precision:  0.7329
Validation recall:  0.5350
Validation F1:  0.6185
```



Comparison – 1-gram vs all-gram

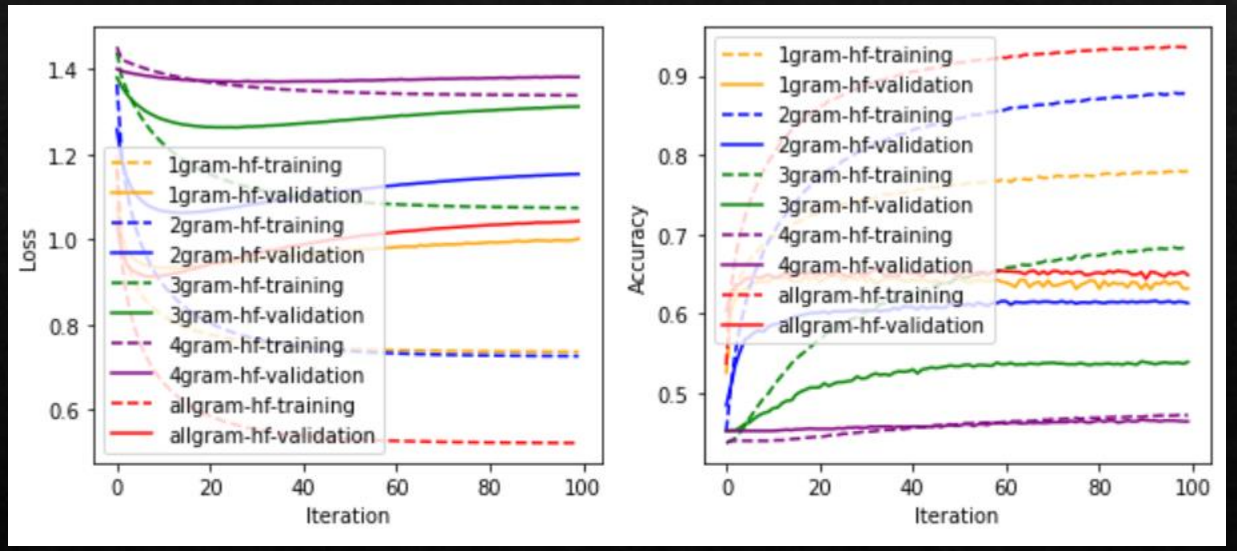
- Red line is the highest → all-gram is the best
- Validation accuracy: 0.6560

High Frequency 1-gram

```
Train loss:  0.7506
Train accuracy:  0.7482
Train precision:  0.8419
Train recall:  0.6075
Train F1:  0.7057

Validation loss:  0.9517
Validation accuracy:  0.6480
Validation precision:  0.7329
Validation recall:  0.5145
Validation F1:  0.6046
```

High Frequency all-gram

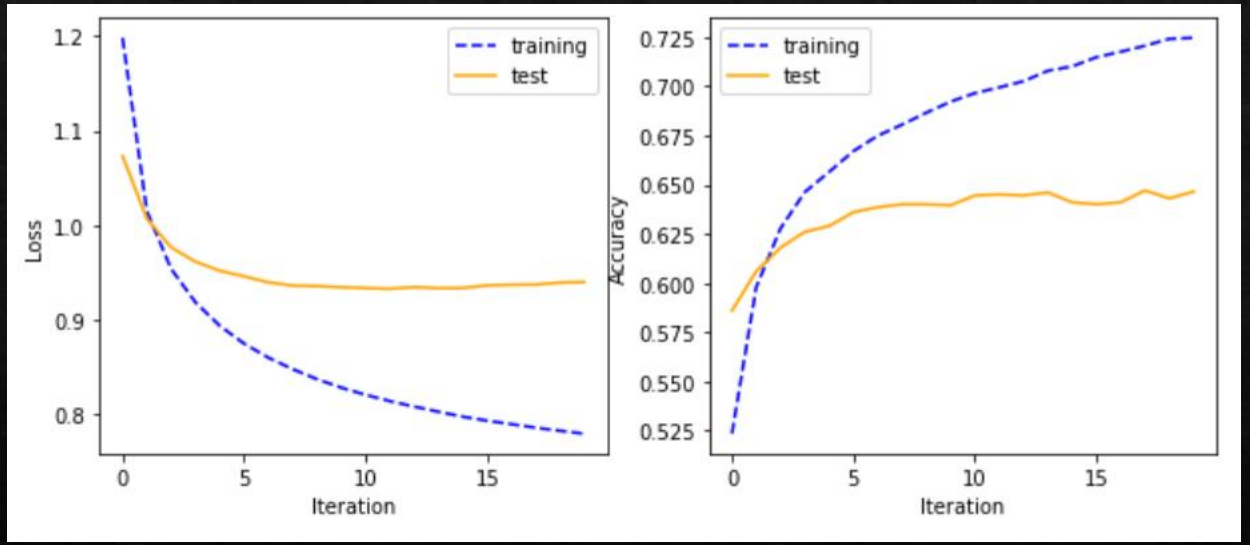
```
Train loss:  0.5411
Train accuracy:  0.8993
Train precision:  0.9507
Train recall:  0.7989
Train F1:  0.8683

Validation loss:  0.9733
Validation accuracy:  0.6560
Validation precision:  0.7329
Validation recall:  0.5350
Validation F1:  0.6185
```



Feature Representation

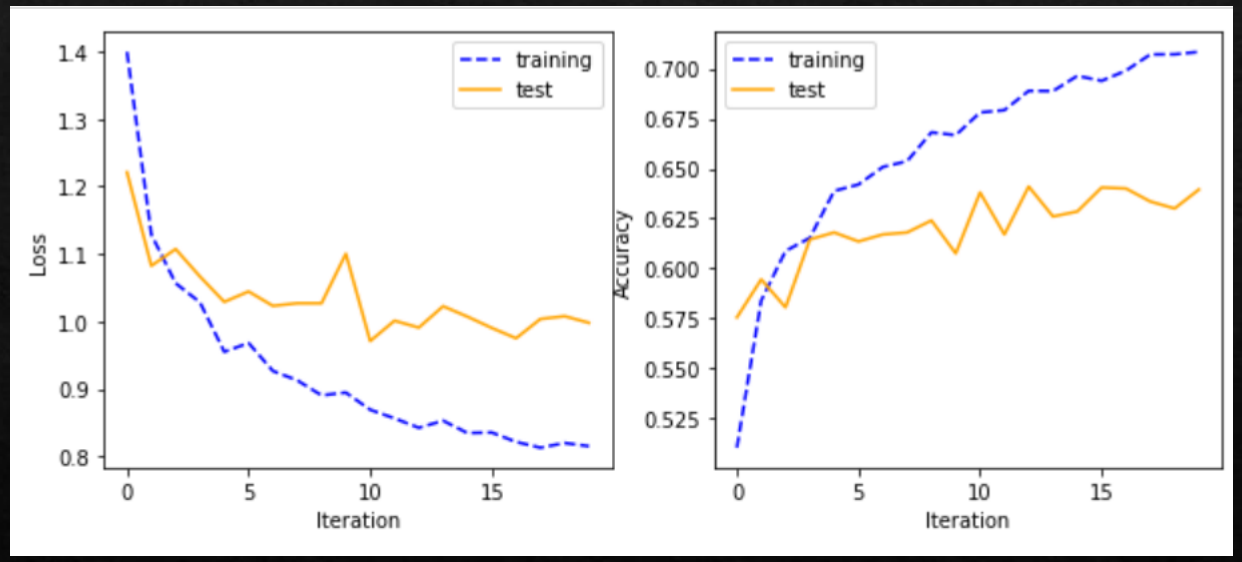
One-Hot (high frequency 1-gram)

```
Train loss:  0.7750
Train accuracy:  0.7284
Train precision:  0.8317
Train recall:  0.5757
Train F1:  0.6804

Validation loss:  0.9370
Validation accuracy:  0.6470
Validation precision:  0.7426
Validation recall:  0.5020
Validation F1:  0.5990
```



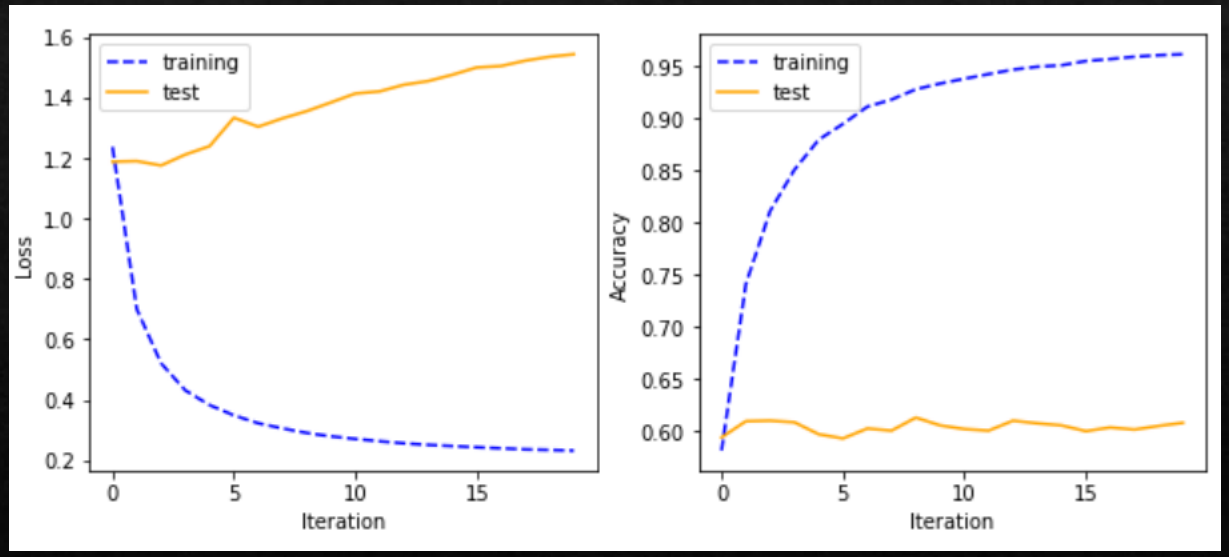
Count (high frequency 1-gram)

```
Train loss:  0.7857
Train accuracy:  0.7139
Train precision:  0.8129
Train recall:  0.5799
Train F1:  0.6769

Validation loss:  0.9910
Validation accuracy:  0.6410
Validation precision:  0.7373
Validation recall:  0.5080
Validation F1:  0.6015
```



TF-IDF (high frequency 1-gram)

```
Train loss:  0.2489
Train accuracy:  0.9464
Train precision:  0.9626
Train recall:  0.9243
Train F1:  0.9430

Validation loss:  1.3546
Validation accuracy:  0.6135
Validation precision:  0.6386
Validation recall:  0.5850
Validation F1:  0.6106
```



Comparison (high frequency 1-gram)

One-Hot

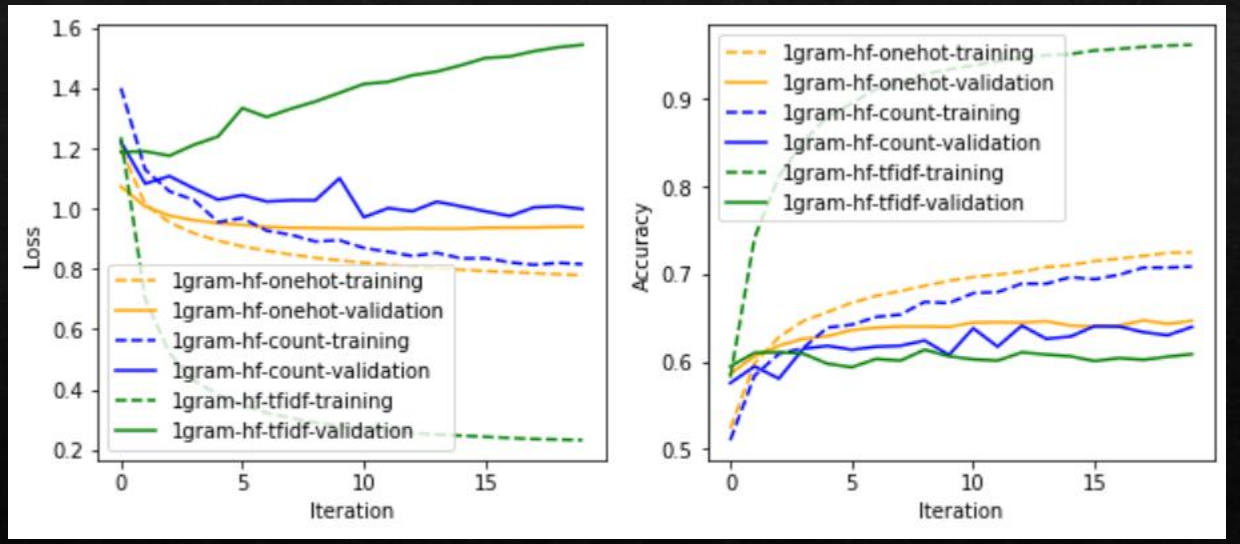
```
Train loss:  0.7750
Train accuracy:  0.7284
Train precision:  0.8317
Train recall:  0.5757
Train F1:  0.6804

Validation loss:  0.9370
Validation accuracy:  0.6470
Validation precision:  0.7426
Validation recall:  0.5020
Validation F1:  0.5990
```

TF-IDF

```
Train loss:  0.2489
Train accuracy:  0.9464
Train precision:  0.9626
Train recall:  0.9243
Train F1:  0.9430

Validation loss:  1.3546
Validation accuracy:  0.6135
Validation precision:  0.6386
Validation recall:  0.5850
Validation F1:  0.6106
```

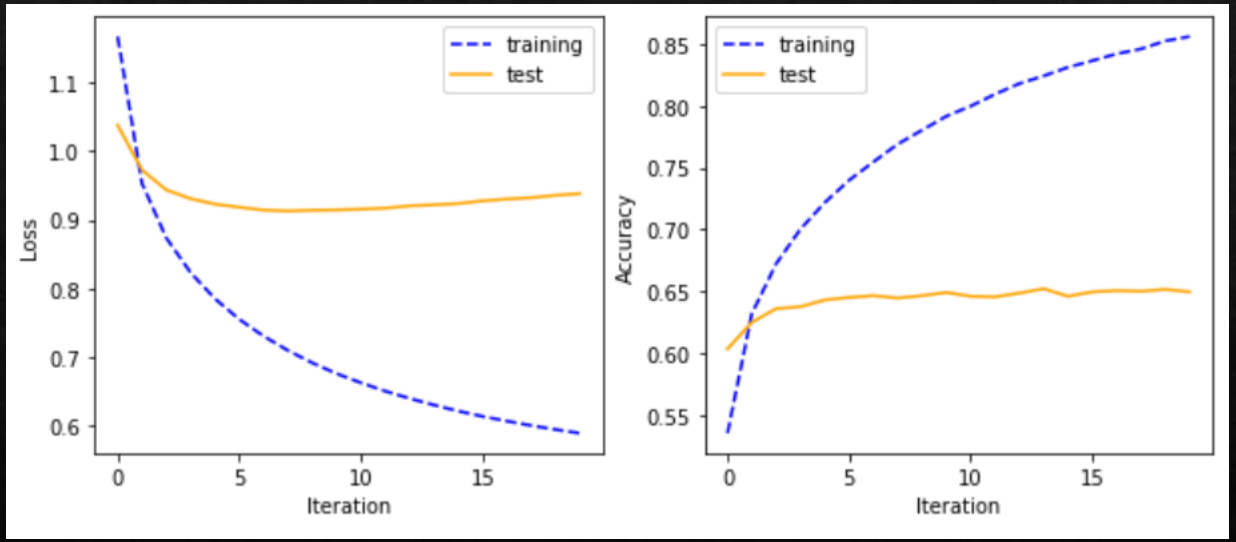
Count

```
Train loss:  0.7857
Train accuracy:  0.7139
Train precision:  0.8129
Train recall:  0.5799
Train F1:  0.6769

Validation loss:  0.9910
Validation accuracy:  0.6410
Validation precision:  0.7373
Validation recall:  0.5080
Validation F1:  0.6015
```



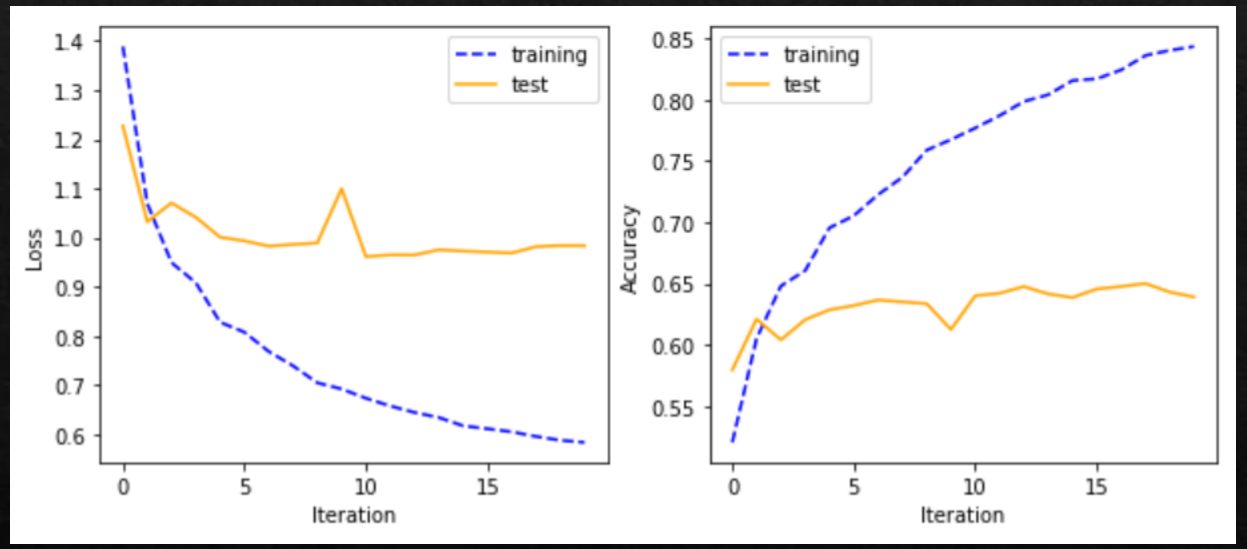
One-Hot (high frequency all-gram)

```
Train loss:  0.6143
Train accuracy:  0.8349
Train precision:  0.9124
Train recall:  0.7006
Train F1:  0.7926

Validation loss:  0.9216
Validation accuracy:  0.6520
Validation precision:  0.7425
Validation recall:  0.5205
Validation F1:  0.6120
```



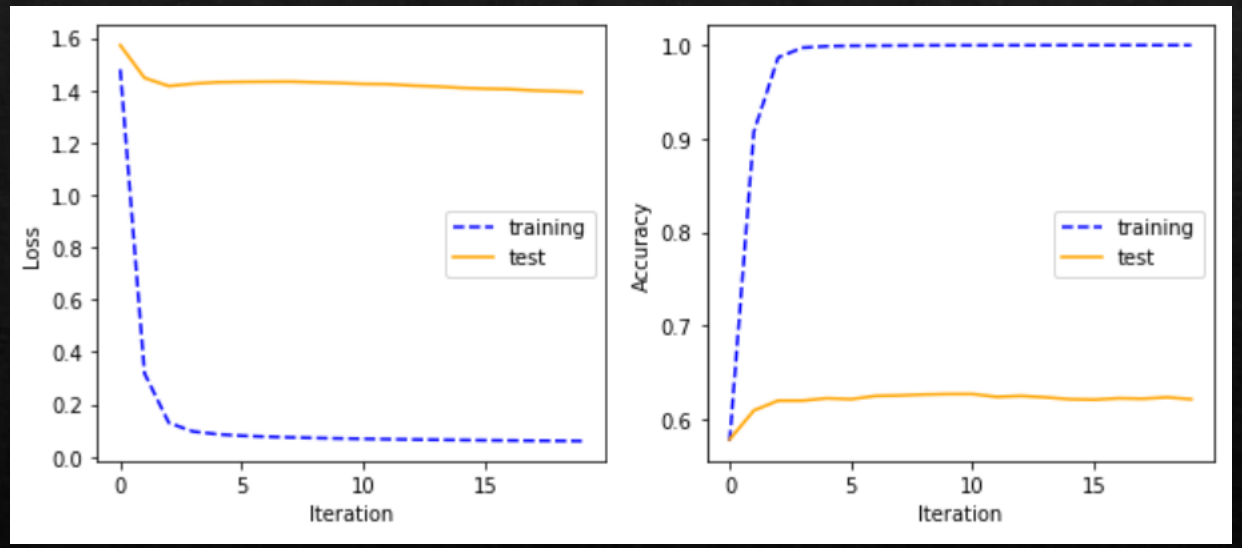
Count (high frequency all-gram)

```
Train loss:  0.5731
Train accuracy:  0.8569
Train precision:  0.9258
Train recall:  0.7364
Train F1:  0.8203

Validation loss:  0.9815
Validation accuracy:  0.6500
Validation precision:  0.7335
Validation recall:  0.5285
Validation F1:  0.6144
```



TF-IDF (high frequency all-gram)

```
Train loss:  0.0669
Train accuracy:  0.9997
Train precision:  1.0000
Train recall:  0.9993
Train F1:  0.9996

Validation loss:  1.4273
Validation accuracy:  0.6270
Validation precision:  0.6478
Validation recall:  0.6060
Validation F1:  0.6262
```



Comparison (high frequency all-gram)

One-Hot

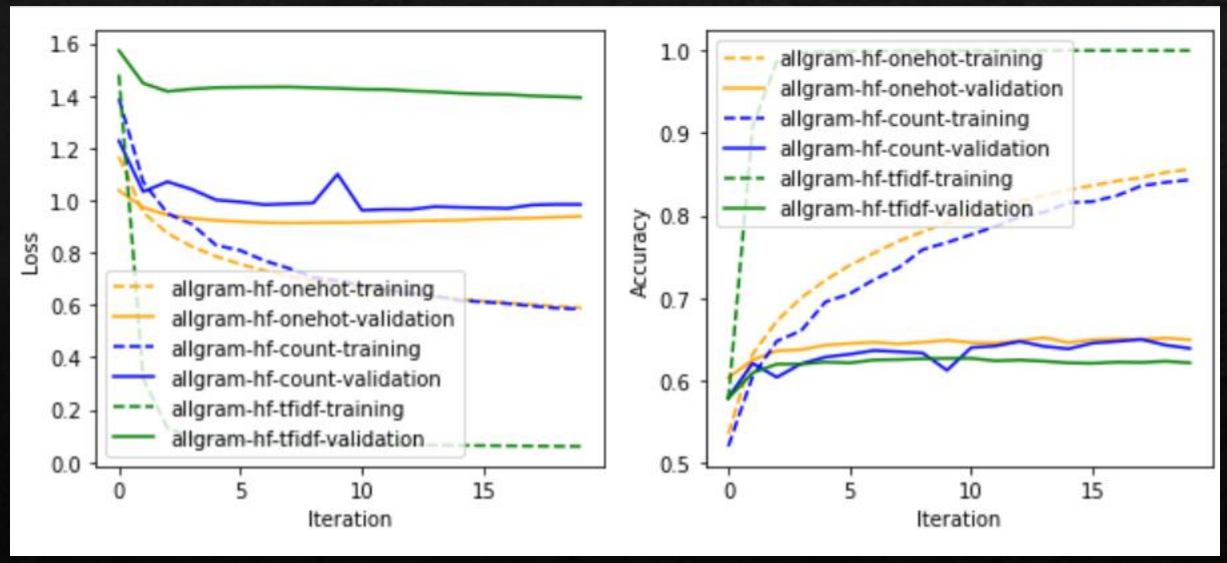
```
Train loss:  0.6143
Train accuracy:  0.8349
Train precision:  0.9124
Train recall:  0.7006
Train F1:  0.7926

Validation loss:  0.9216
Validation accuracy:  0.6520
Validation precision:  0.7425
Validation recall:  0.5205
Validation F1:  0.6120
```

TF-IDF

```
Train loss:  0.0669
Train accuracy:  0.9997
Train precision:  1.0000
Train recall:  0.9993
Train F1:  0.9996

Validation loss:  1.4273
Validation accuracy:  0.6270
Validation precision:  0.6478
Validation recall:  0.6060
Validation F1:  0.6262
```

Count

```
Train loss:  0.5731
Train accuracy:  0.8569
Train precision:  0.9258
Train recall:  0.7364
Train F1:  0.8203

Validation loss:  0.9815
Validation accuracy:  0.6500
Validation precision:  0.7335
Validation recall:  0.5285
Validation F1:  0.6144
```



Conclusion

- One-Hot is the best among the 3 feature representation methods

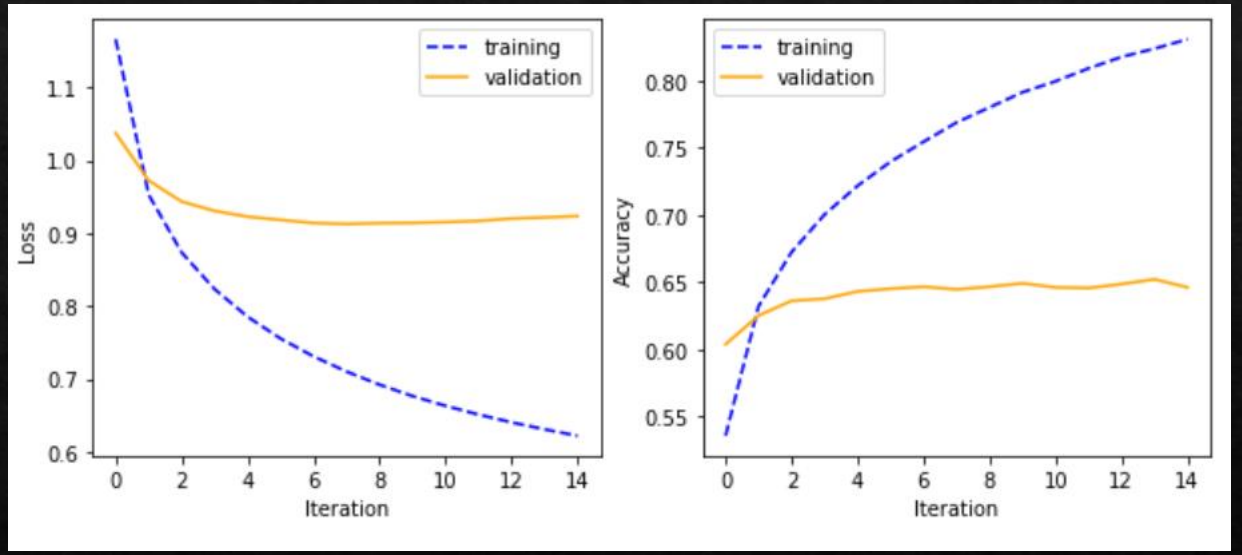
Combining All Previous Observations

Model Settings

- Features: All-gram one-hot vectors
- Loss: Categorical_CrossEntropy
- Optimizer: SGD (learning rate = 0.1)
- Activation Function: softmax
- L2 Regularization: 0.005
- Epoch: 15

Result

```
Train loss:  0.6143
Train accuracy:  0.8349
Train precision:  0.9124
Train recall:  0.7006
Train F1:  0.7926

Validation loss:  0.9216
Validation accuracy:  0.6520
Validation precision:  0.7425
Validation recall:  0.5205
Validation F1:  0.6120
```



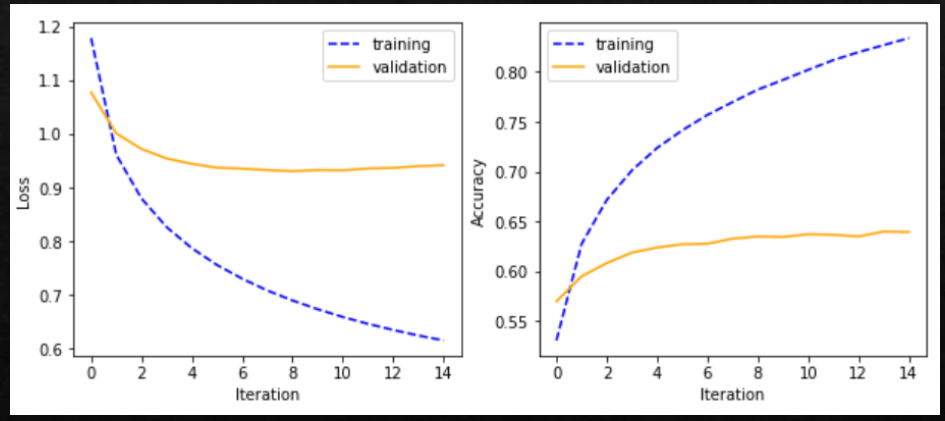
Model\Validation Performance	Macro-F1	Precision	Recall	Accuracy
1-Layer-Perceptron	0.6120	0.7425	0.5205	0.6520
Strong Baseline	**0.5370**	**0.5509**	**0.5324**	**0.6500**

Using Validation Split to Evaluate Performance

Use Validation Split to Evaluate Performance

- Here, we try to use a validation split = 0.1 in the training dataset, and treat the validation dataset given as the test set
  - Conclusion: Similar result – test accuracy = 0.6530

```
Train loss:  0.6426
Train accuracy:  0.8195
Train precision:  0.9037
Train recall:  0.6839
Train F1:  0.7786

Test loss:  0.9286
Test accuracy:  0.6530
Test precision:  0.7324
Test recall:  0.5145
Test F1:  0.6044
```



Model\Validation Performance	Macro-F1	Precision	Recall	Accuracy
1-Layer-Perceptron (Validation Split)	0.6044	0.7324	0.5145	0.6530
Strong Baseline	**0.5370**	**0.5509**	**0.5324**	**0.6500**

Final Model
– 1 layer perceptron

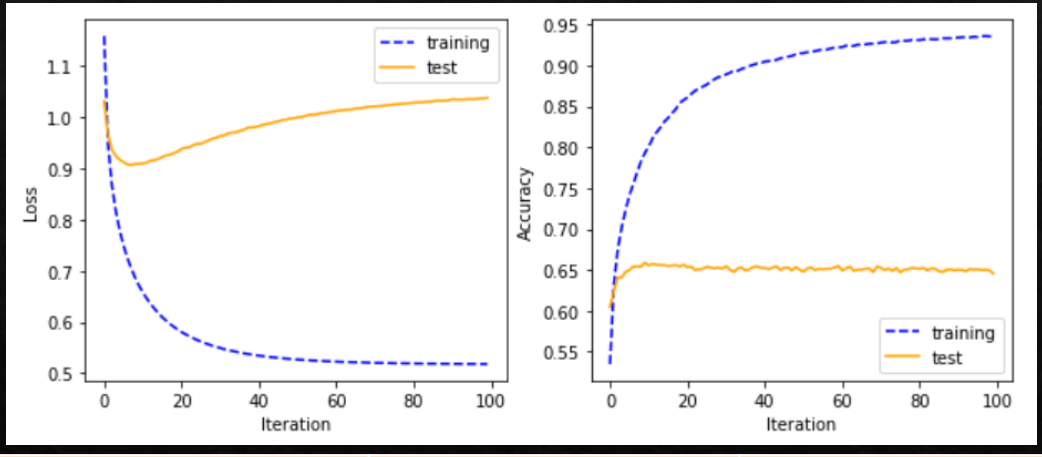
Background

- From previous experiments, we can see that the model using high-frequency all-gram text data performed the best, with validation accuracy = 0.6560
- In our final 1-layer perceptron model, we will add ‘cool’, ‘funny’, and ‘useful’ as features since in EDA we observed that there might be some correlation between them and the target variable ‘stars’
- We try to discard the data with them > 10 since from EDA we see that most of the training data have value <=10
- We then append the processed data into the training features

Final Model Result

- Result: best performance among all experiments for 1-layer perceptron
  - Validation accuracy: 0.6590

```
Train loss:  0.6506
Train accuracy:  0.8087
Train precision:  0.9020
Train recall:  0.6625
Train F1:  0.7639

Validation loss:  0.9093
Validation accuracy:  0.6590
Validation precision:  0.7511
Validation recall:  0.5085
Validation F1:  0.6064
```



Model\Validation Performance	Macro-F1	Precision	Recall	Accuracy
1-Layer-Perceptron	0.6064	0.7511	0.5085	0.6590
Strong Baseline	**0.5370**	**0.5509**	**0.5324**	**0.6500**

CNN

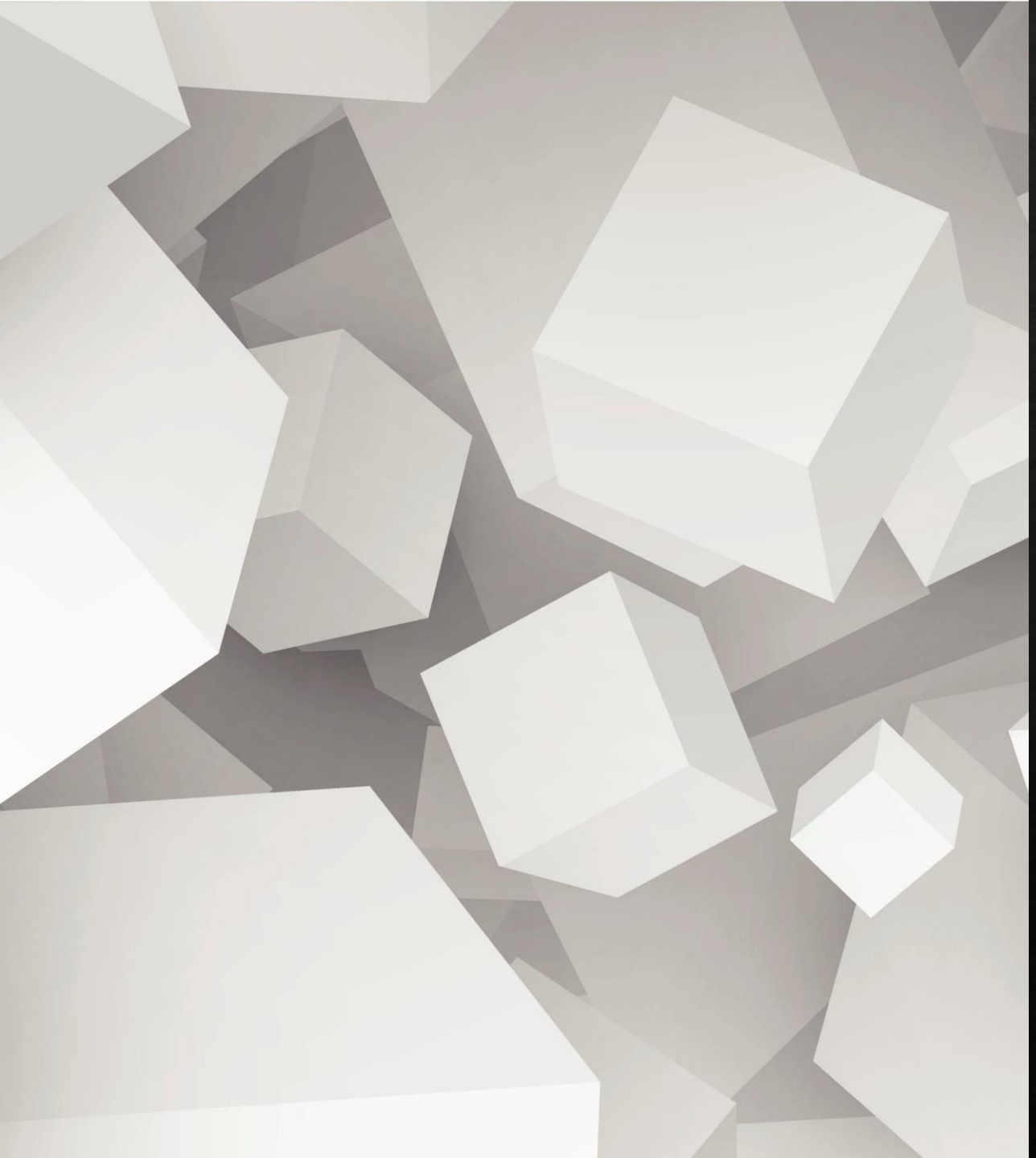
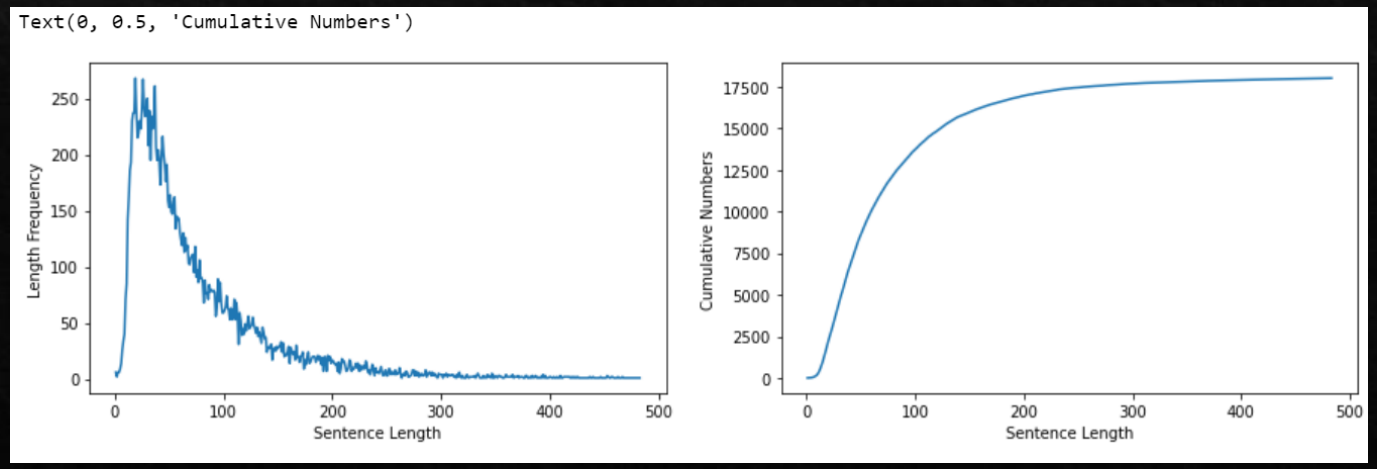


Table of Content

1st Trial – follow tutorial

1st Trial – follow tutorial

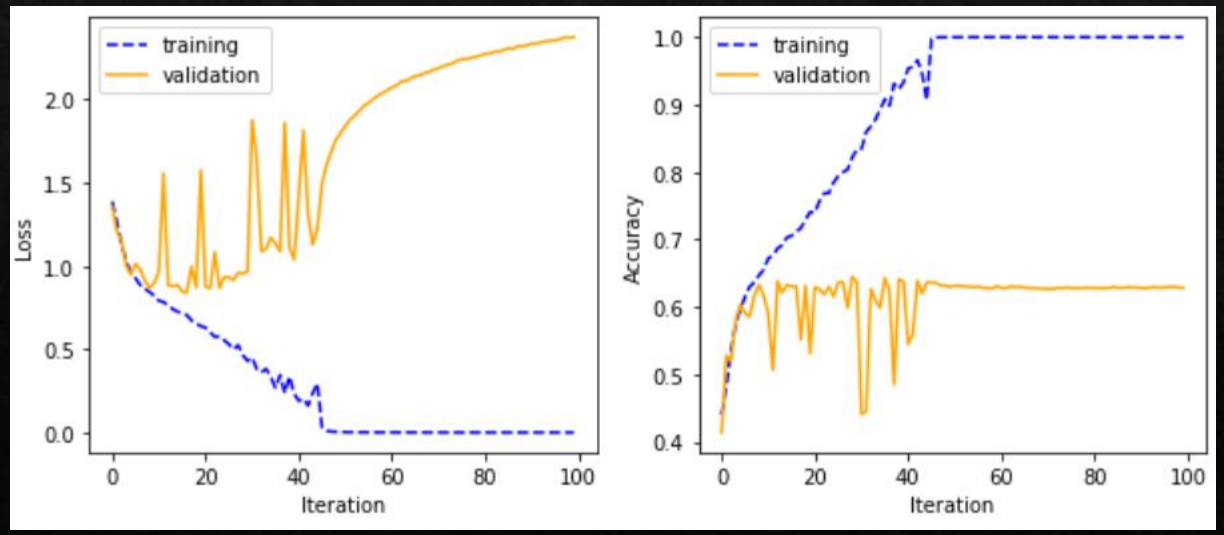
- Setup
  - The convolution with a filter can extract the n-gram-level features
    - Only select the stemmed features whose frequencies are no less than 4
  - From the graph below, we can see that around half of them are having length below 75, and most of them below length 150. Therefore, set Vocab size = 150.
  - Parameters:
    - Input_length=150, vocab_size=len(feats_dict), embedding_size=100, hidden_size=100, output_size=5, kernel_sizes=[1,2,3,4], num_filters=100, num_mlp_layers=3, activation="relu“
  - Use Validation Split = 0.1



1st Trial – follow tutorial

- Result: Unstable training process, overfitting
  - Should add L2 regularization, batch normalization, dropout

```
Train loss:  0.3570
Train accuracy:  0.8801
Train precision:  0.9000
Train recall:  0.8556
Train F1:  0.8772

Validation loss:  0.9841
Validation accuracy:  0.6315
Validation precision:  0.6665
Validation recall:  0.5985
Validation F1:  0.6307
```

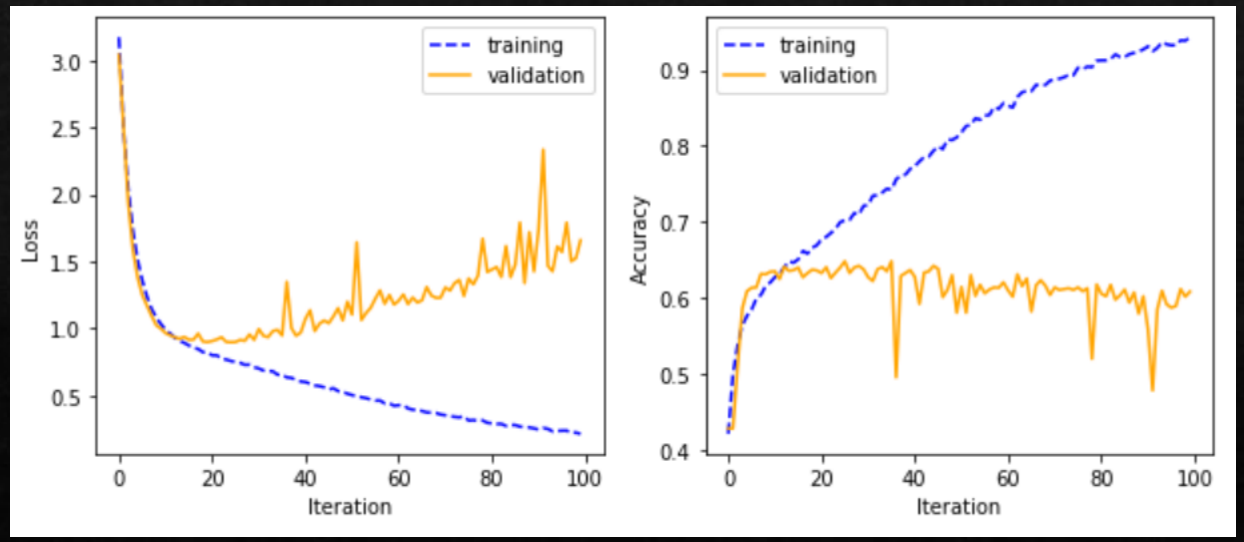


2nd Trial – follow tutorial

2nd Trial – follow tutorial

- Added L2 regularization, batch normalization, dropout
  - Result: performance improved, validation accuracy = 0.6530

```
Train loss:  0.6183
Train accuracy:  0.7740
Train precision:  0.8338
Train recall:  0.6779
Train F1:  0.7478

Validation loss:  0.9016
Validation accuracy:  0.6530
Validation precision:  0.7263
Validation recall:  0.5665
Validation F1:  0.6365
```



Model Comparison

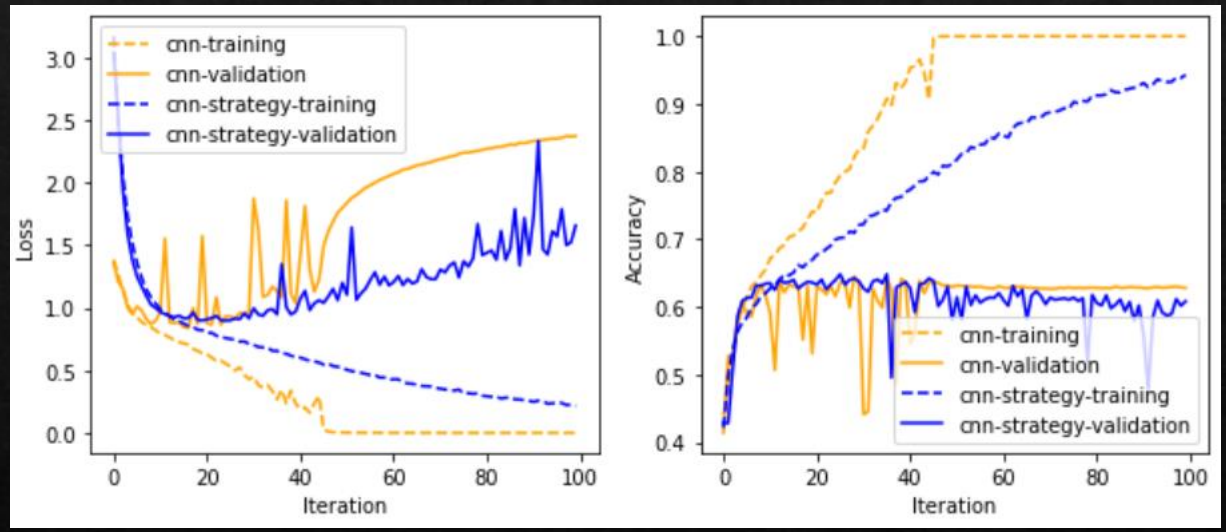
Model Comparison

1st Trial

```
Train loss:  0.3570
Train accuracy:  0.8801
Train precision:  0.9000
Train recall:  0.8556
Train F1:  0.8772

Validation loss:  0.9841
Validation accuracy:  0.6315
Validation precision:  0.6665
Validation recall:  0.5985
Validation F1:  0.6307
```

2nd Trial

```
Train loss:  0.6183
Train accuracy:  0.7740
Train precision:  0.8338
Train recall:  0.6779
Train F1:  0.7478

Validation loss:  0.9016
Validation accuracy:  0.6530
Validation precision:  0.7263
Validation recall:  0.5665
Validation F1:  0.6365
```



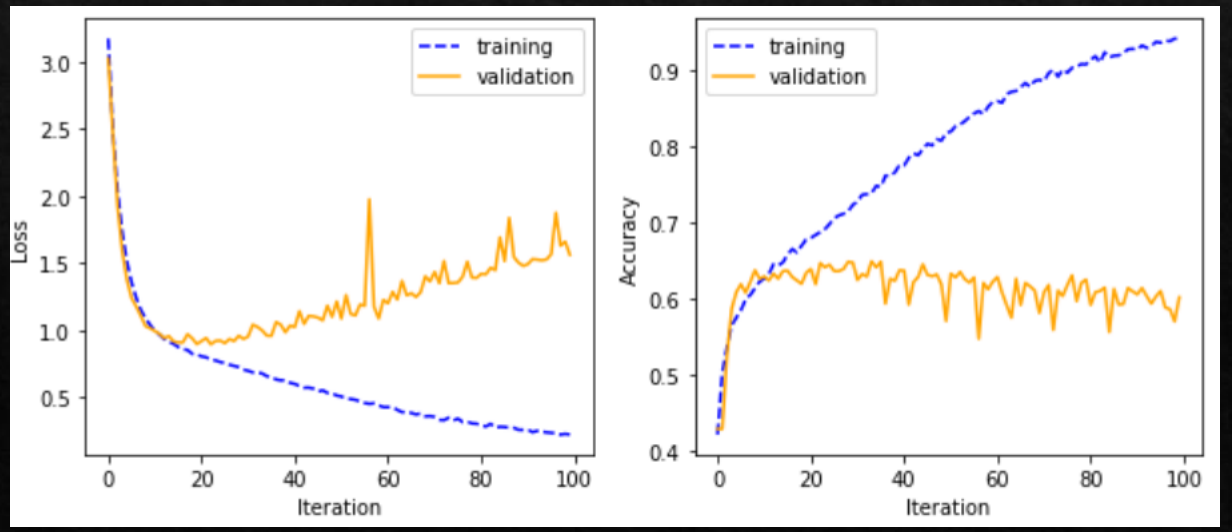
Model\Validation Performance	Macro-F1	Precision	Recall	Accuracy
CNN (2nd Trial)	0.6365	0.7263	0.5665	0.6530
Strong Baseline	**0.5370**	**0.5509**	**0.5324**	**0.6500**

CNN with Text Cleaning

CNN with Text Cleaning

- Trying to remove the tags, single-character removal, remove space before tokenize
  - Performance is lowered, decreased from 0.6530 to 0.6505

```
Train loss:  0.5526
Train accuracy:  0.7874
Train precision:  0.8254
Train recall:  0.7174
Train F1:  0.7676

Validation loss:  0.9728
Validation accuracy:  0.6505
Validation precision:  0.6940
Validation recall:  0.5830
Validation F1:  0.6337
```



CNN – Add ‘useful’ as Feature

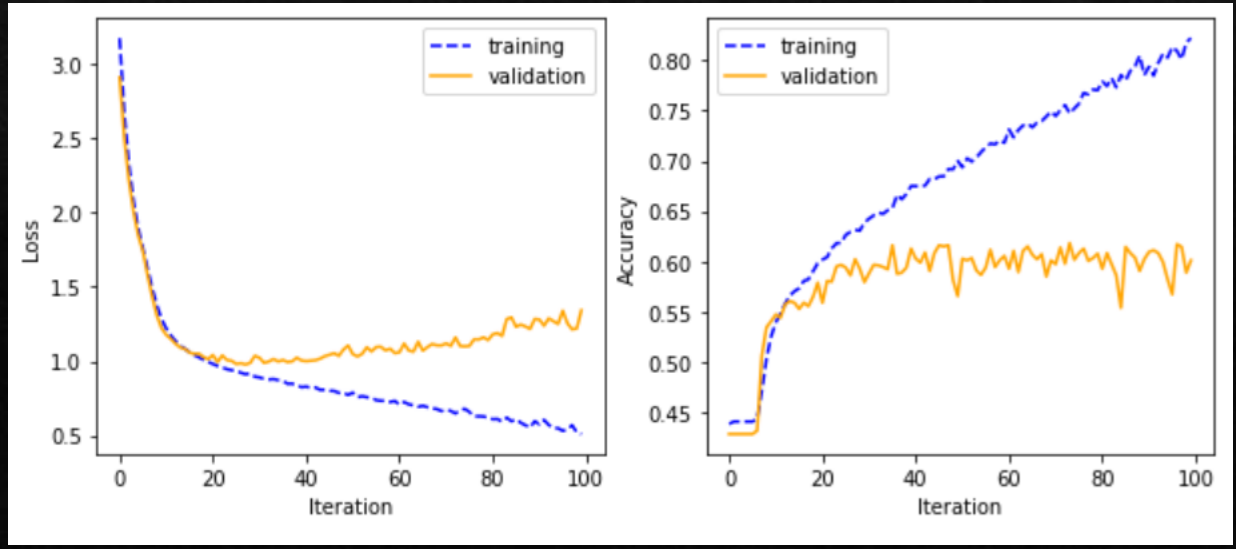
CNN – Add ‘useful’ as Feature

- Setup:
  - Convert ‘useful’ with keras.utils.to_categorical method
  - Model includes 2 branches:
    - 1st branch: CNN from tutorial 4 with text input
    - 2nd branch: 1-Layer Perceptron
  - The model combines to branches using Add() from keras and finally pass the result to a Multi-Layer Perceptron to do classification
  - Other settings are same as before

CNN – Add 'useful' as Feature

- Result:
  - Performance dropped a lot
- Conclusion: need to try other ways to add 'useful' as feature

```
Train loss:  0.5642
Train accuracy:  0.8377
Train precision:  0.9000
Train recall:  0.7198
Train F1:  0.7999

Validation loss:  1.0943
Validation accuracy:  0.6205
Validation precision:  0.6909
Validation recall:  0.5330
Validation F1:  0.6017
```

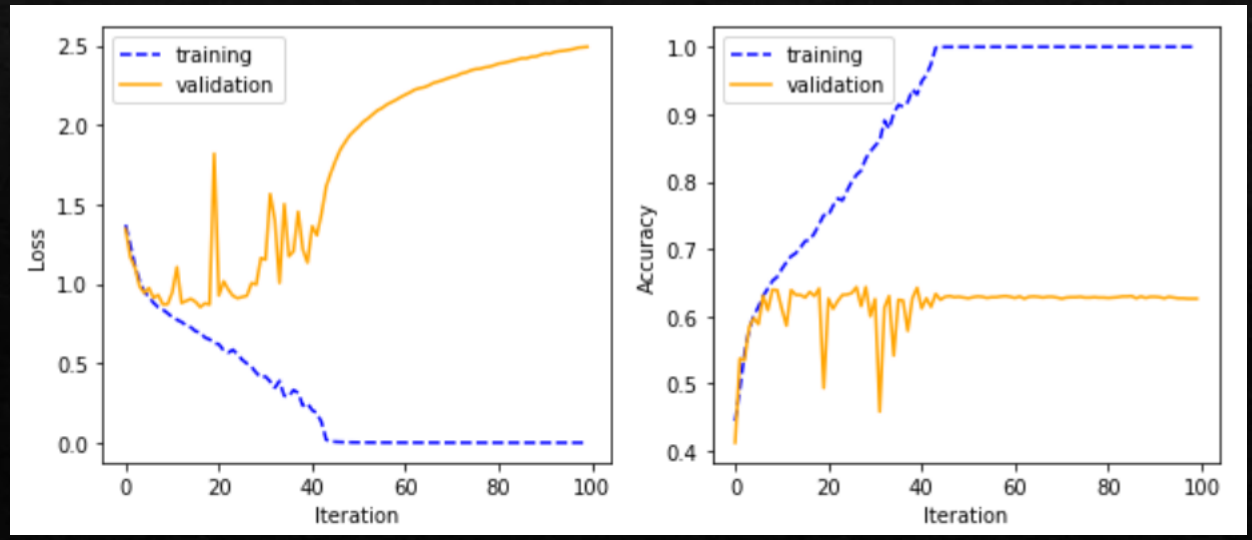


CNN
– Add ‘useful’, ‘funny’, ‘cool’

CNN – Add ‘useful’, ‘funny’, ‘cool’

- Setup:
  - Only consider with count <=10, add ‘useful’, ‘funny’, ‘cool’ into the training features
  - Not add L2 regularization, batch normalization, dropout (settings similar to CNN Trial 1)
- Result: Overfitting

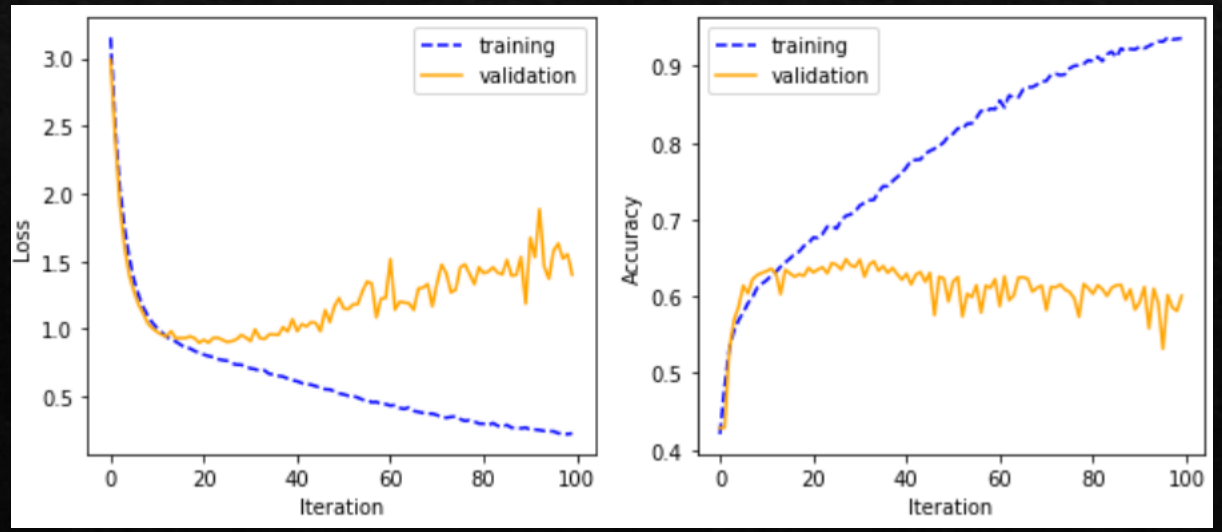
```
Train loss:  0.3688
Train accuracy:  0.8841
Train precision:  0.9061
Train recall:  0.8529
Train F1:  0.8787

Validation loss:  0.9374
Validation accuracy:  0.6315
Validation precision:  0.6690
Validation recall:  0.5810
Validation F1:  0.6219
```



CNN – Add ‘useful’, ‘funny’, ‘cool’

- Settings:
  - Only consider with count <=10, add ‘useful’, ‘funny’, ‘cool’ into the training features
  - Adding L2 regularization, batch normalization, dropout (settings similar to CNN Trial2)
- Result: Performance **enhanced** with validation accuracy = 0.6620

```
Train loss:  0.6094
Train accuracy:  0.7746
Train precision:  0.8350
Train recall:  0.6631
Train F1:  0.7392

Validation loss:  0.9304
Validation accuracy:  0.6620
Validation precision:  0.7372
Validation recall:  0.5610
Validation F1:  0.6371
```

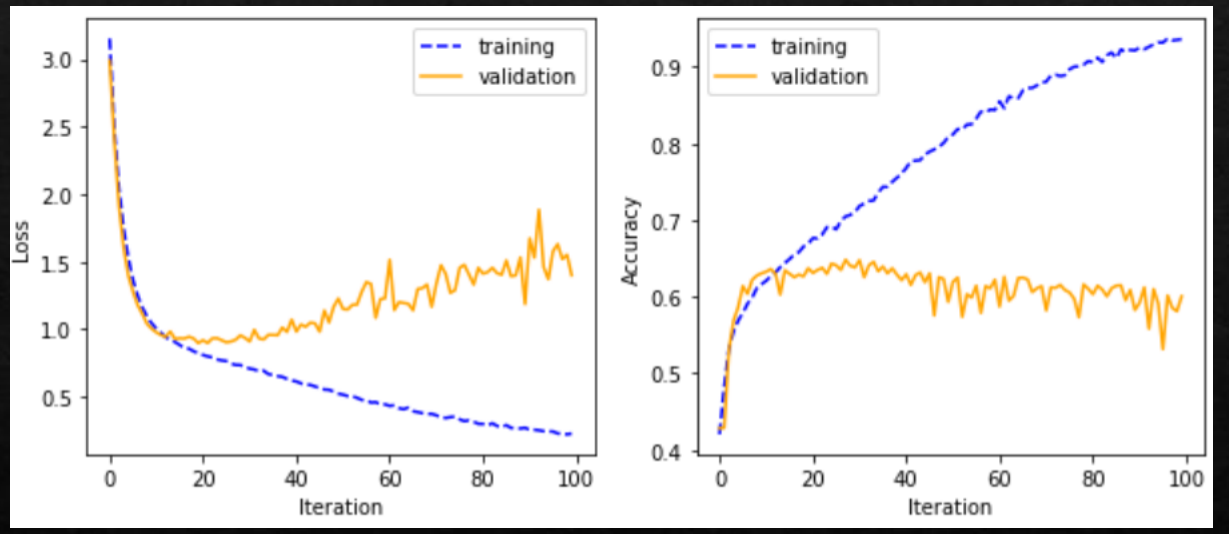


Best Model - CNN

Best Model - CNN

- Adding ‘useful’, ‘funny’, and ‘cool’ into the training features, performance is the best

```
Train loss:  0.6094
Train accuracy:  0.7746
Train precision:  0.8350
Train recall:  0.6631
Train F1:  0.7392

Validation loss:  0.9304
Validation accuracy:  0.6620
Validation precision:  0.7372
Validation recall:  0.5610
Validation F1:  0.6371
```



Model\Validation Performance	Macro-F1	Precision	Recall	Accuracy
CNN (add ‘useful’, ‘funny’, ‘cool’)	0.6371	0.7372	0.5610	0.6620
Strong Baseline	**0.5370**	**0.5509**	**0.5324**	**0.6500**

RNN

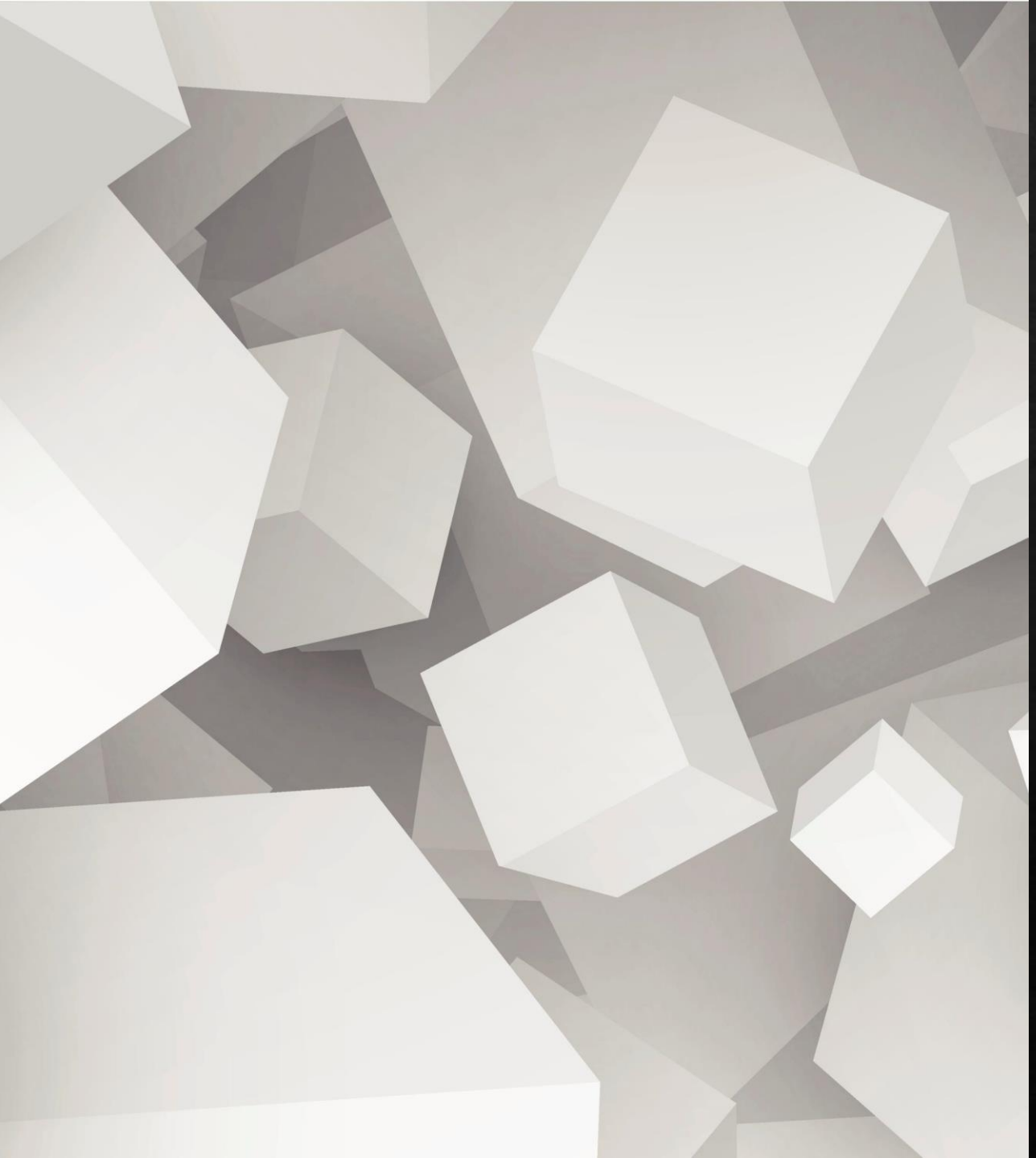


Table of Content

RNN – follow tutorial

LSTM – follow tutorial

GRU – follow tutorial

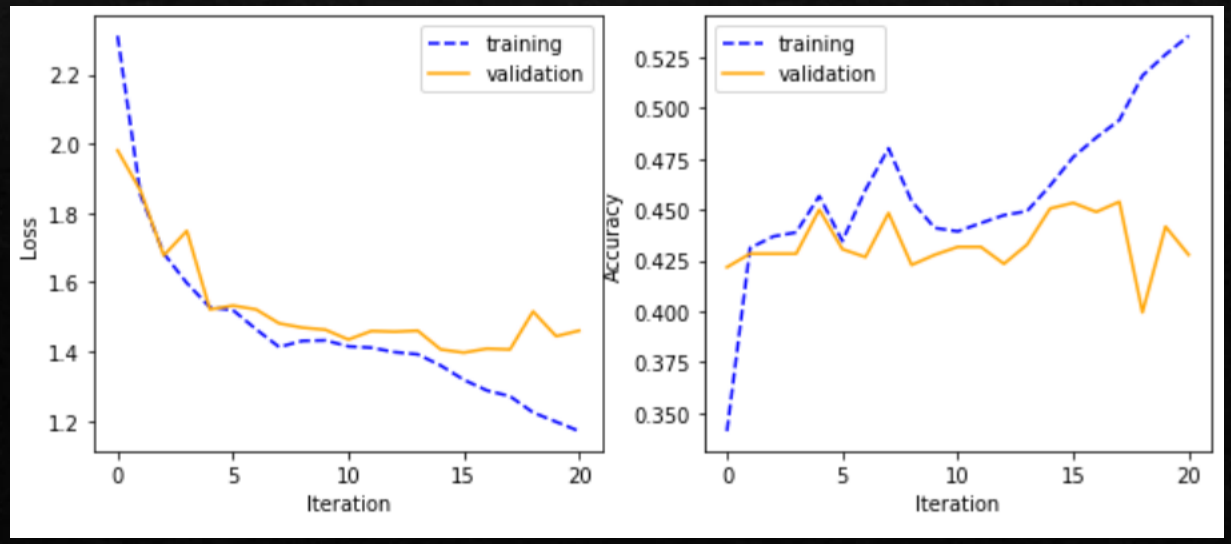
Bidirectional LSTM – follow tutorial

Model Comparison

RNN – follow tutorial

- Setup
  - input_length: 100, embedding_size: 100, num_rnn_layers: 1, hidden_size: 100, num_mlp_layers: 2, activation: tanh, l2_reg: 0.005, dropout_rate: 0.5, and batch_norm: True
  - Here input_length is reduced to 100 to shorten the training time

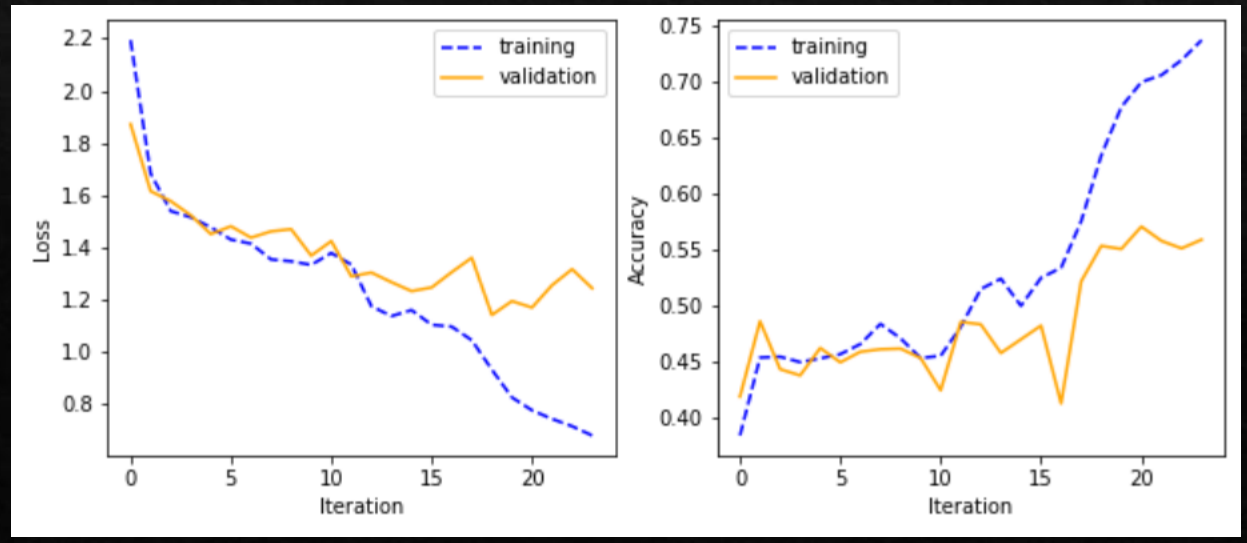
```
RNN:
Train loss:  1.2197
Train accuracy:  0.5097

Validation loss:  1.3347
Validation accuracy:  0.4740
```



LSTM – follow tutorial

- Setup
  - input_length: 100, embedding_size: 100, num_rnn_layers: 1, hidden_size: 100, num_mlp_layers: 2, activation: tanh, l2_reg: 0.005, dropout_rate: 0.5, and batch_norm: True

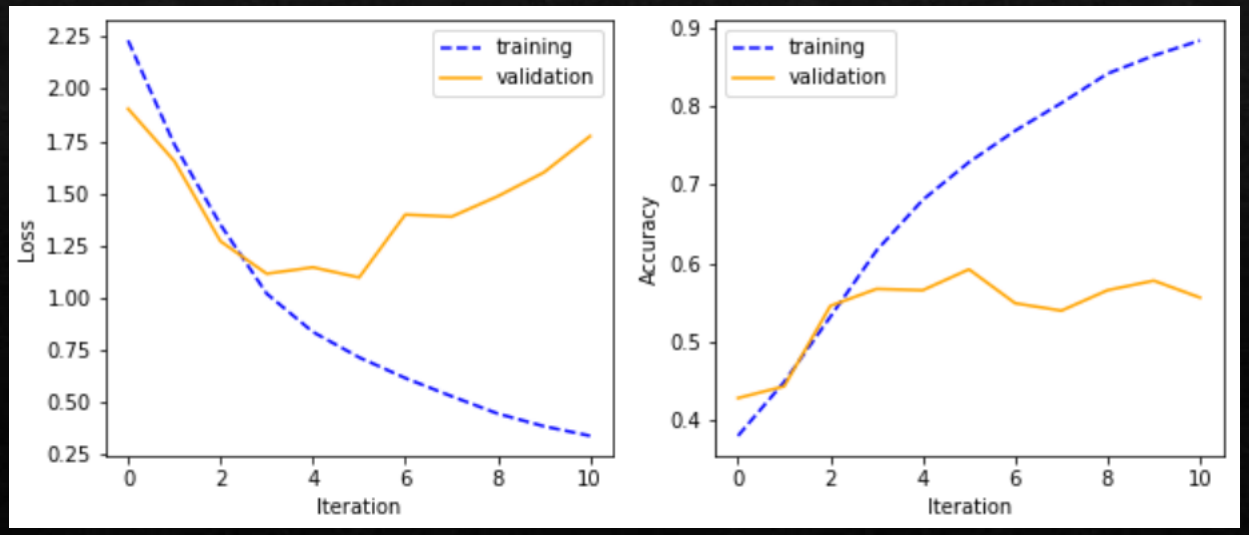
```
LSTM:
Train loss:  0.7270
Train accuracy:  0.7241

Validation loss:  1.1997
Validation accuracy:  0.5495
```



GRU – follow tutorial

- Setup
  - input_length: 100, embedding_size: 100, num_rnn_layers: 1, hidden_size: 100, num_mlp_layers: 2, activation: tanh, l2_reg: 0.005, dropout_rate: 0.5, and batch_norm: True

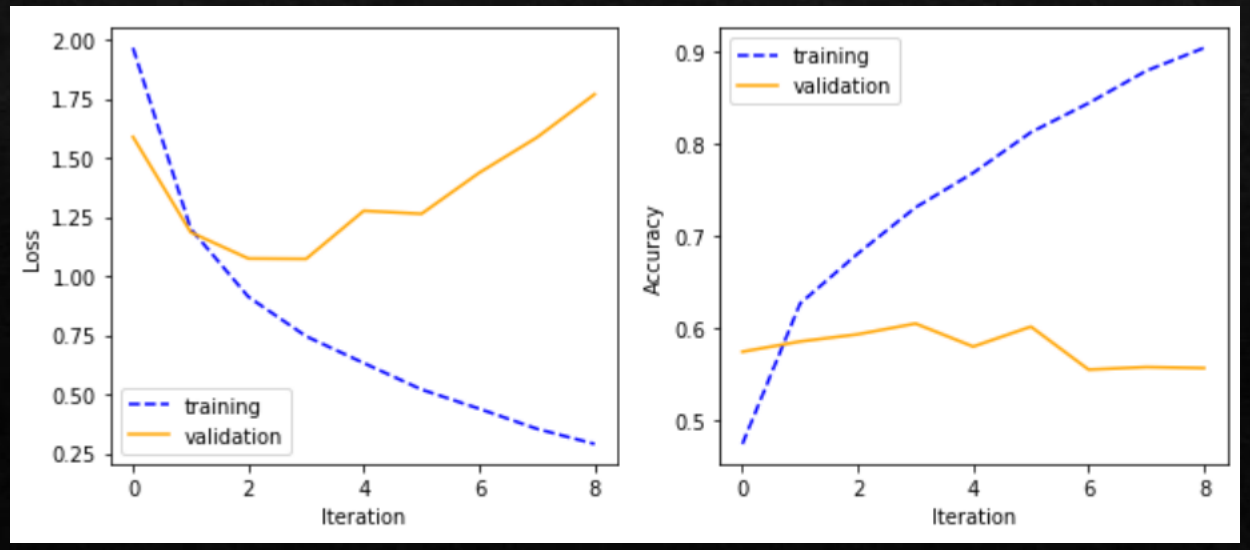
```
GRU:
Train loss:  0.6364
Train accuracy:  0.7688

Validation loss:  1.0456
Validation accuracy:  0.6025
```



Bidirectional LSTM – follow tutorial

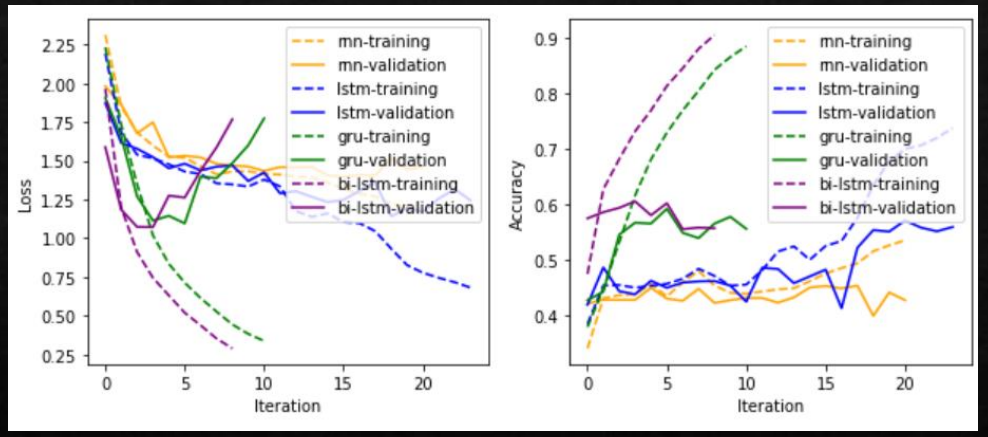
- Setup
  - input_length: 100, embedding_size: 100, num_rnn_layers: 1, hidden_size: 100, num_mlp_layers: 2, activation: tanh, l2_reg: 0.005, dropout_rate: 0.5, and batch_norm: True

```
Bidirectional LSTM:
Train loss:  0.6599
Train accuracy:  0.7776

Validation loss:  1.0323
Validation accuracy:  0.5995
```



Model Comparison

- Result: Bi-direction LSTM performed the best
  - But still, it is not performing very well compare with 1-layer perceptron and CNN




```
RNN:
Train loss:  1.2197
Train accuracy:  0.5097

Validation loss:  1.3347
Validation accuracy:  0.4740

LSTM:
Train loss:  0.7270
Train accuracy:  0.7241

Validation loss:  1.1997
Validation accuracy:  0.5495

GRU:
Train loss:  0.6364
Train accuracy:  0.7688

Validation loss:  1.0456
Validation accuracy:  0.6025

Bidirectional LSTM:
Train loss:  0.6599
Train accuracy:  0.7776

Validation loss:  1.0323
Validation accuracy:  0.5995
```

Model\Validation Performance	Macro-F1	Precision	Recall	Accuracy
RNN (Bidirectional LSTM)	0.4785	0.5101	0.4818	0.5995
Strong Baseline	**0.5370**	**0.5509**	**0.5324**	**0.6500**

RNN+CNN

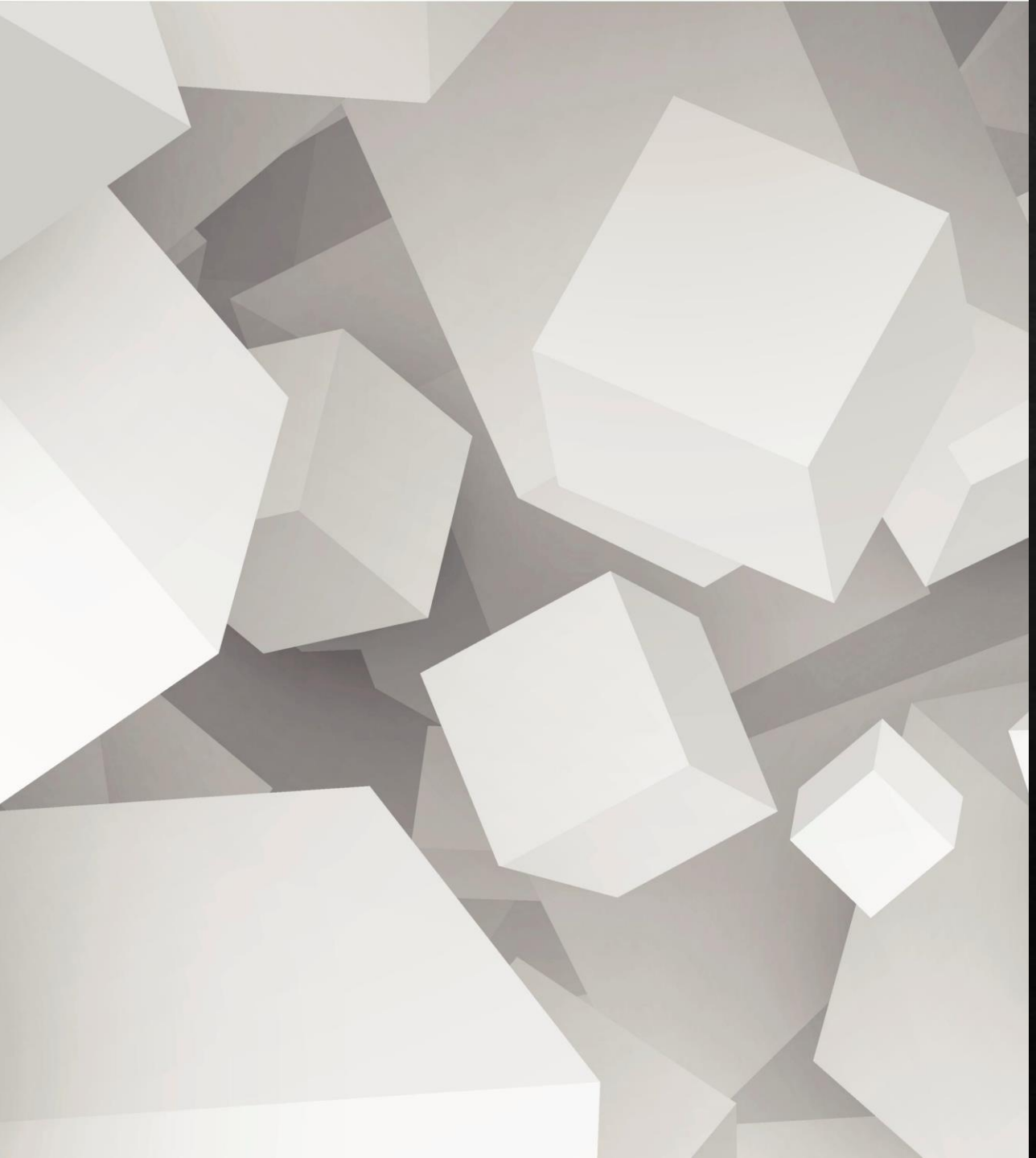


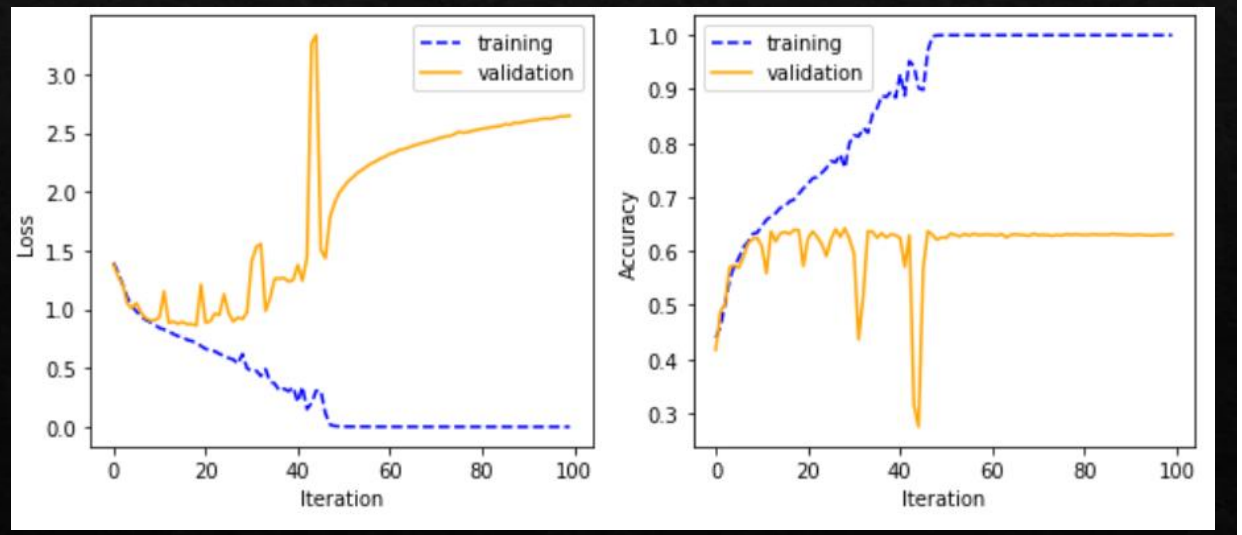
Table of Content

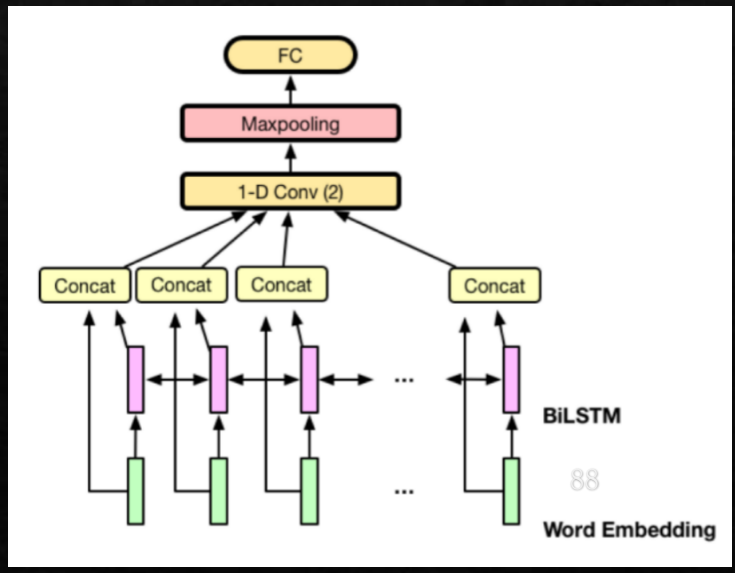
RNN+CNN

```
model_RNN_CNN = build_model(input_length=150,
                            vocab_size=len(feats_dict),
                            embedding_size=100,
                            hidden_size=100,
                            output_size=5,
                            kernel_sizes=[1,2,3,4],
                            num_filters=100,
                            num_mlp_layers=3,
                            dropout_rate=0.5,
                            batch_norm=True,
                            l2_reg=0.005)
```

- Aim: Leverage the pros of both BiLSTM (memory) and CNN (extract n-gram features)
  - Trial 1: Learning rate: 0.1, Activation="relu", Optimizer="SGD", Batch size =100
    - Poor performance (Validation accuracy: 0.6325) → Overfitting
    - Without dropout, batch norm, and regularization

```
Train loss:  0.4632
Train accuracy:  0.8224
Train precision:  0.8549
Train recall:  0.7682
Train F1:  0.8092

Validation loss:  0.9439
Validation accuracy:  0.6325
Validation precision:  0.6788
Validation recall:  0.5790
Validation F1:  0.6249
```





Hyperparameter Tuning

Background

- From the previous trial, we tried without batch normalization, l2 regularization, and dropout, and overfitting problem seems exist. So from now on, all the models we try will include batch normalization, l2 regularization = 0.05 (same as tutorial), and dropout rate = 0.5 (same as tutorial)
- We will examine learning rate, optimizer, activation function, and batch size.
- Since training takes time, we will do a bunch of experiments at the same time, and investigation will be conducted after every batch of experiments finished.

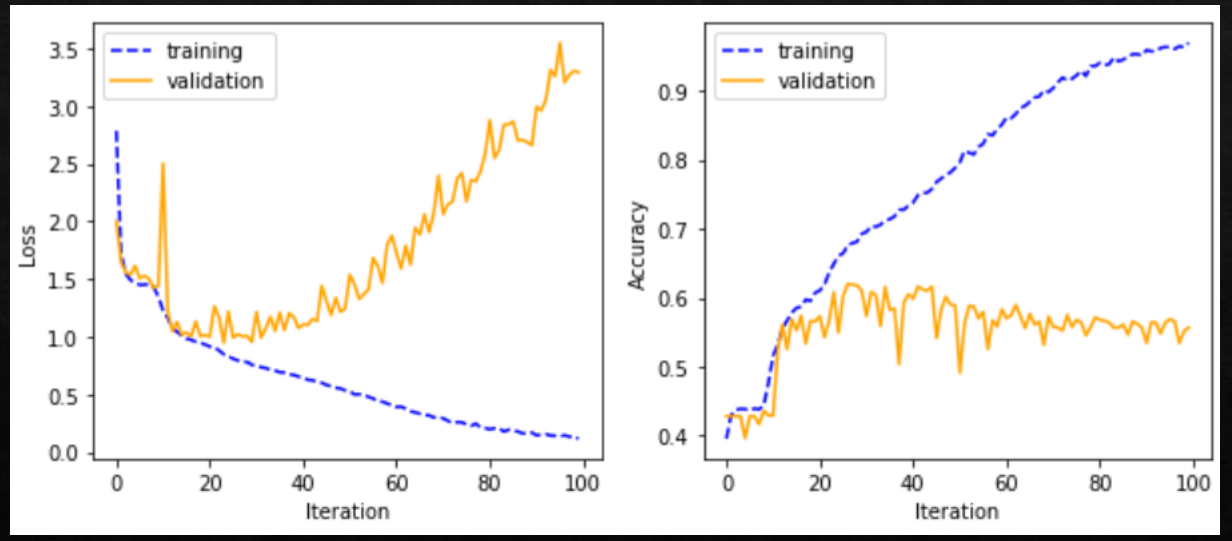
1st Batch of Experiments

Learning Rate

RNN-CNN with anti-overfitting

- Learning rate = 0.5, Optimizer = "SGD", Epoch = 100
  - Result: performance dropped

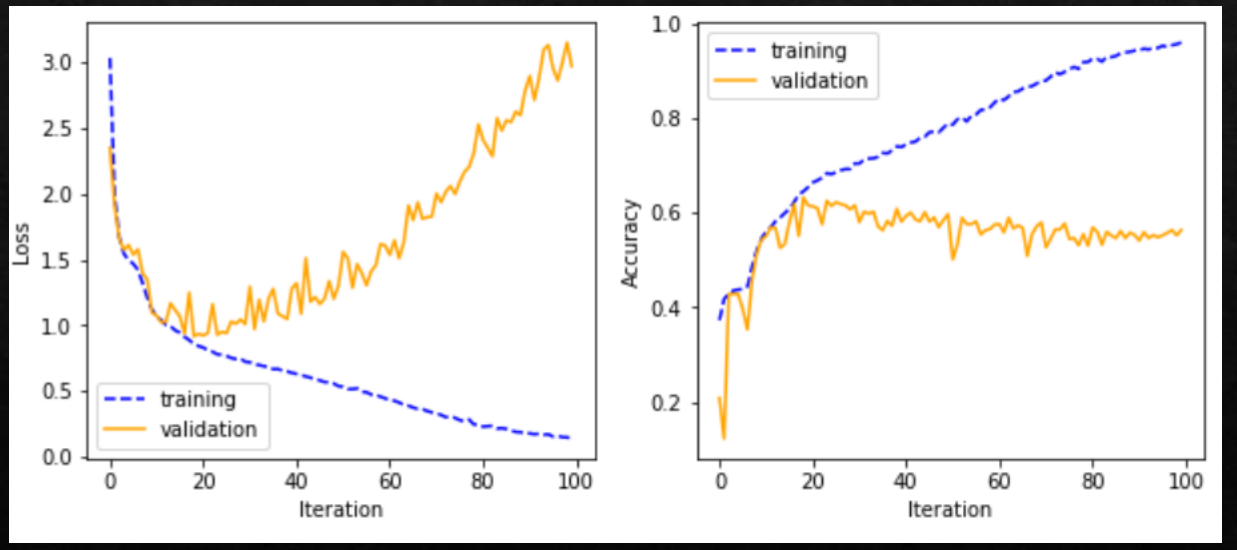
```
Train loss:  0.7918
Train accuracy:  0.6646
Train precision:  0.7189
Train recall:  0.6082
Train F1:  0.6589

Validation loss:  0.9788
Validation accuracy:  0.6205
Validation precision:  0.6729
Validation recall:  0.5575
Validation F1:  0.6098
```



RNN-CNN with anti-overfitting

- Learning rate = 0.3, Optimizer = "SGD", Epoch =100
  - Result: performance dropped

```
Train loss:  0.7214
Train accuracy:  0.7144
Train precision:  0.7862
Train recall:  0.5979
Train F1:  0.6792

Validation loss:  0.9758
Validation accuracy:  0.6150
Validation precision:  0.7050
Validation recall:  0.5175
Validation F1:  0.5969
```

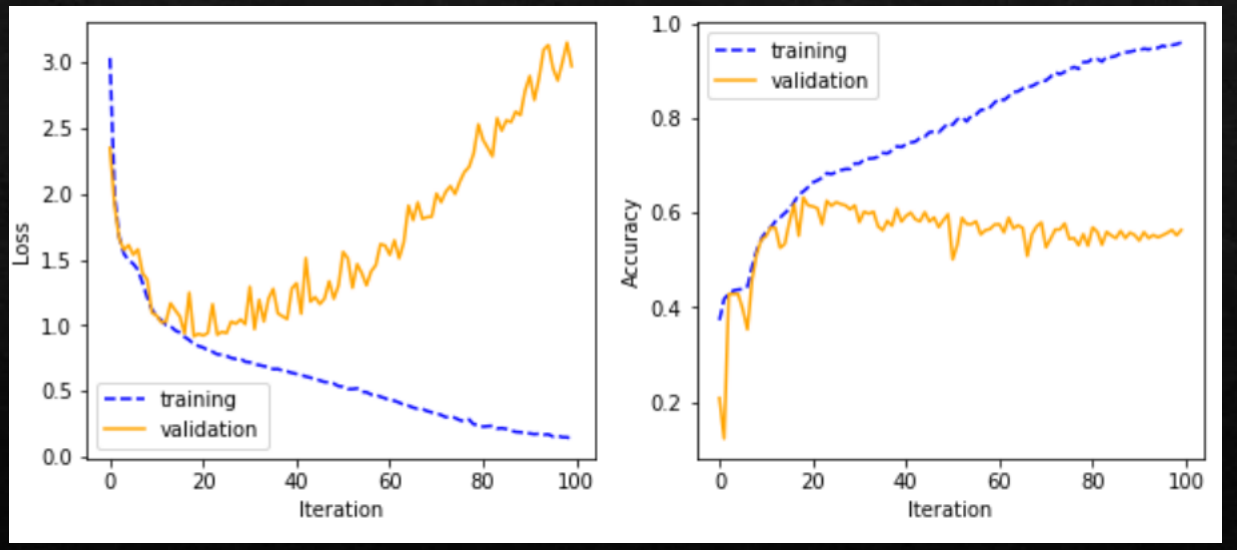


Optimizer

RNN-CNN with anti-overfitting

- As shown before, Learning rate = 0.3, **Optimizer = "SGD",** Epoch =100

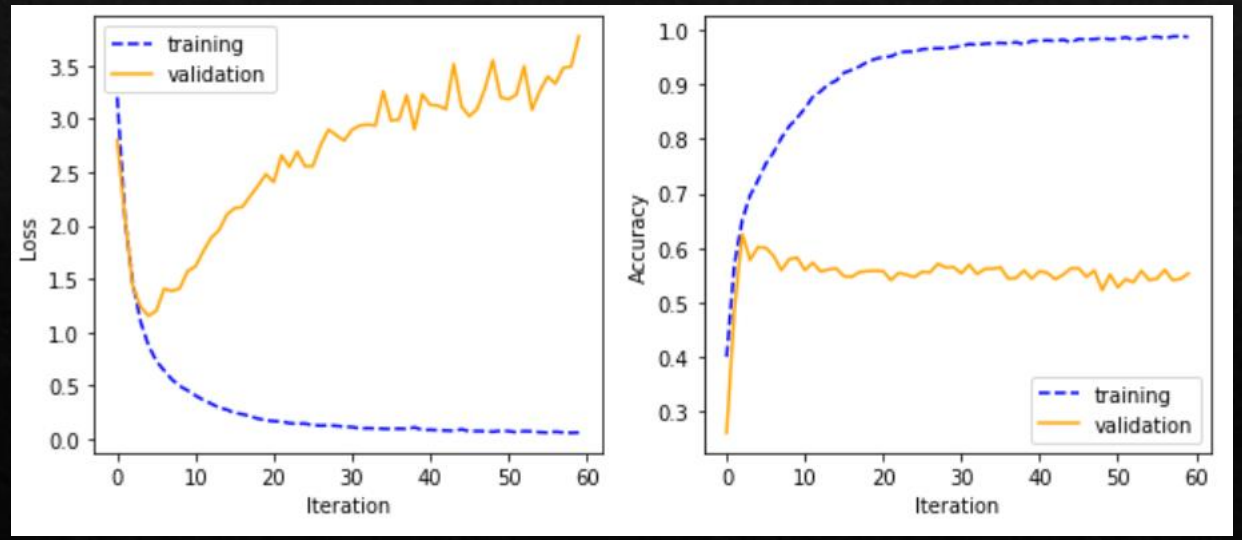
```
Train loss:  0.7214
Train accuracy:  0.7144
Train precision:  0.7862
Train recall:  0.5979
Train F1:  0.6792

Validation loss:  0.9758
Validation accuracy:  0.6150
Validation precision:  0.7050
Validation recall:  0.5175
Validation F1:  0.5969
```



RNN-CNN – Changing Optimizer

- Learning rate = 0.3, **Optimizer = "Adam",** Activation="relu", Epoch=60
  - Result: "Adam" could be a better choice of optimizer (but since it self-tune parameters during training, we will continue hyperparameter tuning with 'SGD')

```
Train loss:  1.2744
Train accuracy:  0.7209
Train precision:  0.8729
Train recall:  0.4691
Train F1:  0.6102

Validation loss:  1.4135
Validation accuracy:  0.6290
Validation precision:  0.8026
Validation recall:  0.4085
Validation F1:  0.5414
```

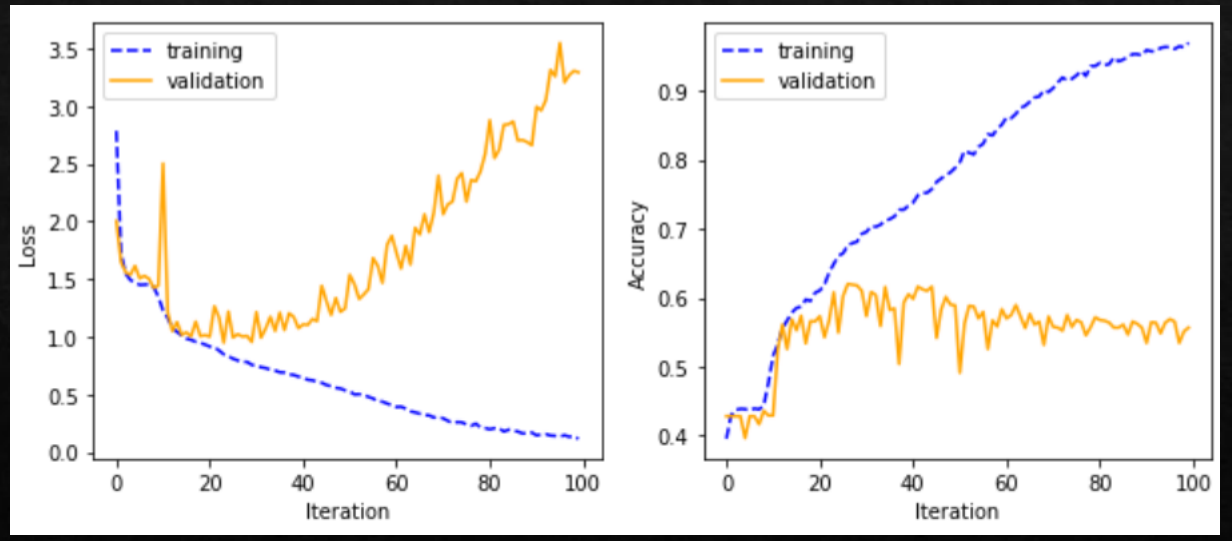


Activation Function

RNN-CNN with anti-overfitting

- As shown before, Learning rate = 0.5, Optimizer = "SGD", **Activation = "relu"**, Epoch = 100

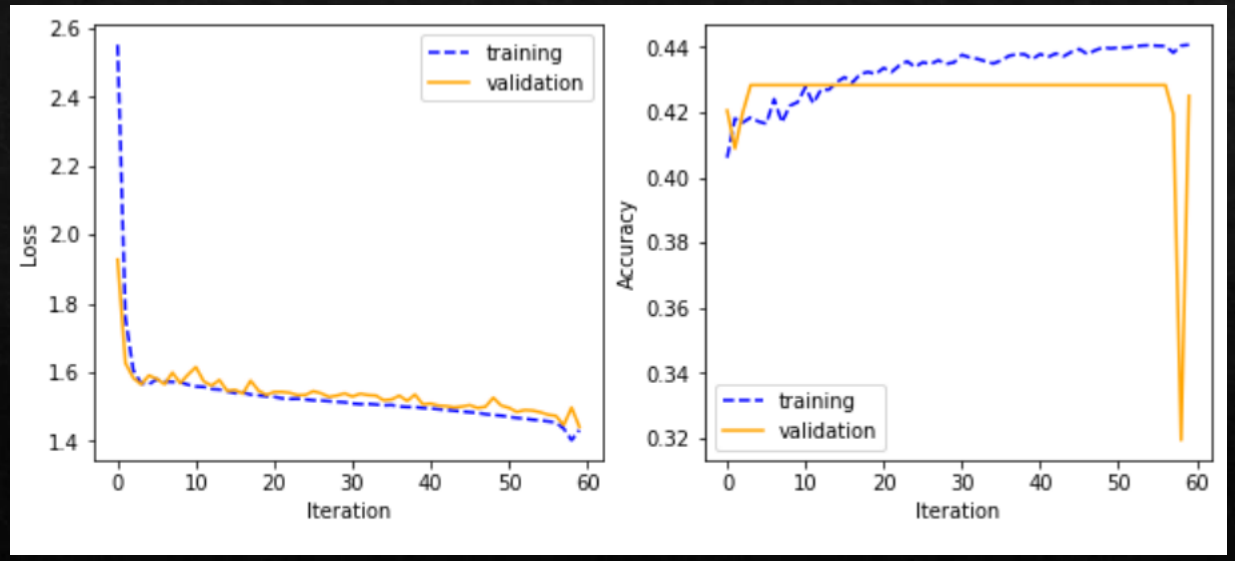
```
Train loss:  0.7918
Train accuracy:  0.6646
Train precision:  0.7189
Train recall:  0.6082
Train F1:  0.6589

Validation loss:  0.9788
Validation accuracy:  0.6205
Validation precision:  0.6729
Validation recall:  0.5575
Validation F1:  0.6098
```



RNN-CNN – Changing Activation

- Learning rate = 0.5, Optimizer = "SGD", **Activation = "tanh",** Epoch=60

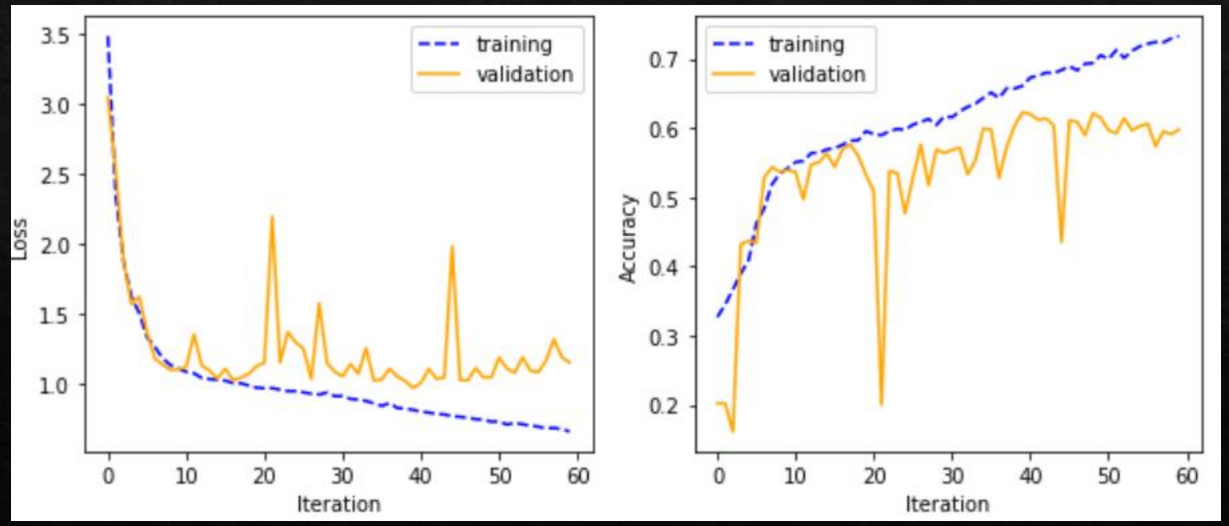
```
Train loss:  1.5439
Train accuracy:  0.4395
Train precision:  0.6822
Train recall:  0.0081
Train F1:  0.0160

Validation loss:  1.5243
Validation accuracy:  0.4520
Validation precision:  0.7826
Validation recall:  0.0090
Validation F1:  0.0178
```



RNN-CNN – Changing Activation

- Learning rate = 0.5, Optimizer = "SGD", **Activation = "sigmoid",** Epoch=60

```
Train loss:  0.7390
Train accuracy:  0.7035
Train precision:  0.7891
Train recall:  0.5750
Train F1:  0.6652

Validation loss:  0.9752
Validation accuracy:  0.6135
Validation precision:  0.7259
Validation recall:  0.5110
Validation F1:  0.5998
```



RNN-CNN – Changing Activation

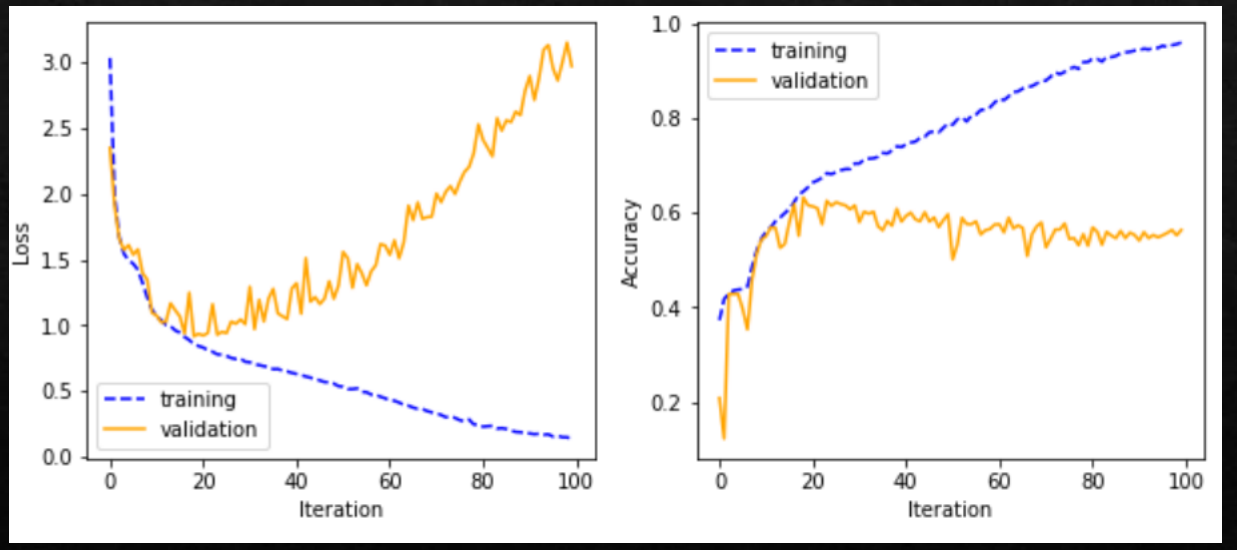
- Conclusion: Activation = “relu” gives the best result

Batch Size

RNN-CNN with anti-overfitting

- As shown before, Learning rate = 0.3, Optimizer = "SGD", Activation = "relu", Epoch =100, **Batch size = 100**

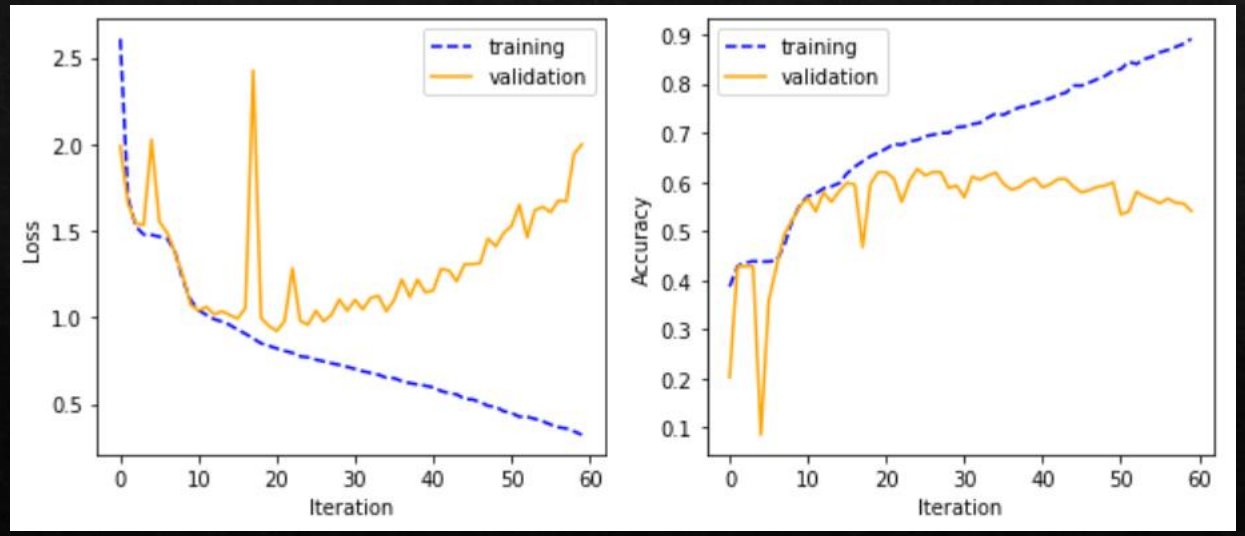
```
Train loss:  0.7214
Train accuracy:  0.7144
Train precision:  0.7862
Train recall:  0.5979
Train F1:  0.6792

Validation loss:  0.9758
Validation accuracy:  0.6150
Validation precision:  0.7050
Validation recall:  0.5175
Validation F1:  0.5969
```



RNN-CNN – Changing Batch Size

- Learning rate = 0.3, Optimizer = "SGD", Activation="relu", **Batch Size = 50**
  - Result: same performance, but the training process is less stable

```
Train loss:  0.7214
Train accuracy:  0.7144
Train precision:  0.7862
Train recall:  0.5979
Train F1:  0.6792

Validation loss:  0.9758
Validation accuracy:  0.6150
Validation precision:  0.7050
Validation recall:  0.5175
Validation F1:  0.5969
```



RNN-CNN – Changing Batch Size

- Conclusion: No difference

Conclusion

- By varying the learning rate, optimizer, activation function, and batch size, we discovered that:
  - The performance for all new trials are poorer than the 1st trial without any parameter tuning and anti-overfitting techniques
  - Learning rate=0.5 perform better than learning rate=0.3, but poorer than learning rate=0.1, not sure what this mean, more investigation will be needed
  - Optimizer="Adam" yields better performance with same model configuration, but since it would have done self-tunes during training, and will get away from the default hyperparameters, we will still use "SGD" as the optimizer
  - Activation="relu" would be a better choice, so from now on we will use "relu" to do the experiment
  - Changing batch size does not change the performance, so from now on we will use batch size = 100 since the training process would be more stable

2nd Batch of Experiments

Background

- We fixed optimizer="SGD", activation="relu", batch size=100

Learning Rate

RNN-CNN – Changing learning rate

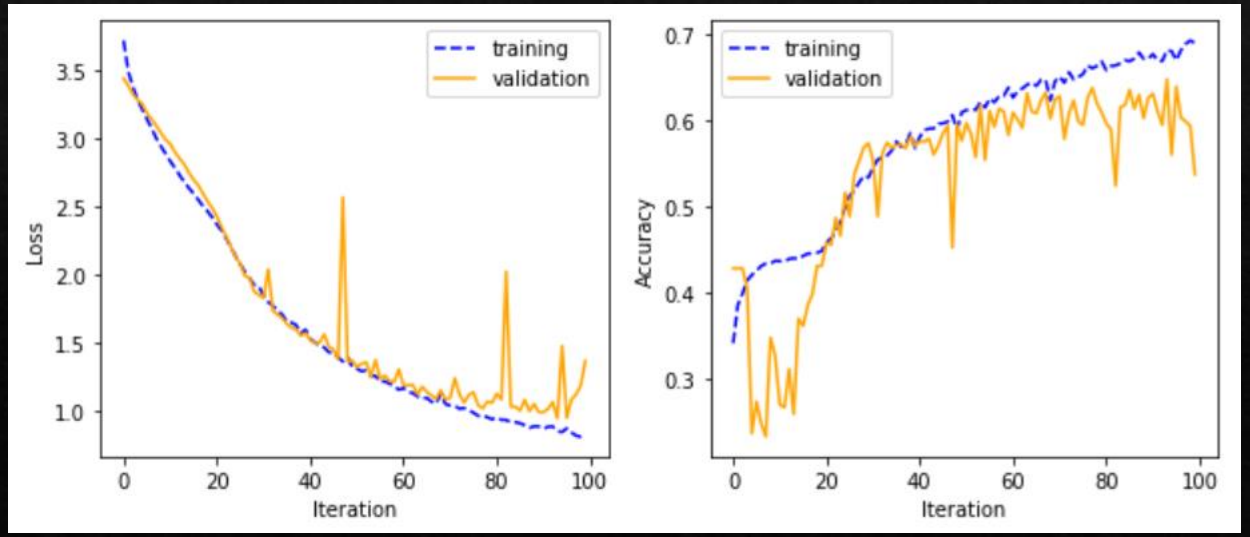
- Learning rate = 0.1
  - Starting from the 30th epoch, the validation accuracy did not improved much.
  - The best validation accuracy is 0.6239, with a validation loss of 0.9317
  - We stopped the training process at the 70th epoch to save computation resources

```
0.5606 - val_loss: 0.9846 - val_accuracy: 0.6050 - val_precision: 0.7414 - val_recall: 0.4683
Epoch 41/100
162/162 [==============================] - 412s 3s/step - loss: 0.7944 - accuracy: 0.6710 - precision: 0.7481 - recall:
0.5680 - val_loss: 0.9455 - val_accuracy: 0.6183 - val_precision: 0.7067 - val_recall: 0.5300
Epoch 42/100
162/162 [==============================] - 405s 2s/step - loss: 0.7793 - accuracy: 0.6738 - precision: 0.7450 - recall:
0.5748 - val_loss: 0.9317 - val_accuracy: 0.6239 - val_precision: 0.7017 - val_recall: 0.5422
Epoch 43/100
162/162 [==============================] - 406s 3s/step - loss: 0.7660 - accuracy: 0.6796 - precision: 0.7466 - recall:
0.5902 - val_loss: 1.0103 - val_accuracy: 0.6050 - val_precision: 0.6770 - val_recall: 0.5439
Epoch 44/100
162/162 [==============================] - 403s 2s/step - loss: 0.7606 - accuracy: 0.6842 - precision: 0.7502 - recall:
0.5969 - val_loss: 0.9514 - val_accuracy: 0.6167 - val_precision: 0.6904 - val_recall: 0.5500
Epoch 45/100
162/162 [==============================] - 405s 2s/step - loss: 0.7654 - accuracy: 0.6847 - precision: 0.7477 - recall:
0.5924 - val_loss: 0.9909 - val_accuracy: 0.5906 - val_precision: 0.6532 - val_recall: 0.5200
Epoch 46/100
```

RNN-CNN – Changing learning rate

- Learning rate = 0.01
  - Result: Performance is the best among all trials in this batch, and it outperform the 1st RNN-CNN model we tried too (which had validation accuracy = 0.6325)

```
Train loss:  0.8125
Train accuracy:  0.7015
Train precision:  0.7748
Train recall:  0.5938
Train F1:  0.6723

Validation loss:  0.9537
Validation accuracy:  0.6385
Validation precision:  0.7283
Validation recall:  0.5455
Validation F1:  0.6238
```



RNN-CNN – Changing learning rate

- Learning rate = 0.001
  - Result: training takes too long, after 60 epochs, the training and validation accuracy are stuck at around 0.4 (therefore we manually stopped the training – due to time and computational resource limitation)

```
Epoch 61/100
162/162 [==============================] - 428s 3s/step - loss: 3.0508 - accuracy: 0.4337 - precision: 0.6140 - recall:
0.2054 - val_loss: 3.1272 - val_accuracy: 0.4228 - val_precision: 0.8889 - val_recall: 0.0044
Epoch 62/100
162/162 [==============================] - 438s 3s/step - loss: 3.0442 - accuracy: 0.4360 - precision: 0.6140 - recall:
0.2049 - val_loss: 3.1222 - val_accuracy: 0.4217 - val_precision: 0.9000 - val_recall: 0.0050
Epoch 63/100
162/162 [==============================] - 453s 3s/step - loss: 3.0374 - accuracy: 0.4379 - precision: 0.6151 - recall:
0.2087 - val_loss: 3.1166 - val_accuracy: 0.4217 - val_precision: 0.8182 - val_recall: 0.0050
Epoch 64/100
162/162 [==============================] - 345s 2s/step - loss: 3.0332 - accuracy: 0.4345 - precision: 0.6126 - recall:
0.2062 - val_loss: 3.1064 - val_accuracy: 0.4256 - val_precision: 0.8571 - val_recall: 0.0067
Epoch 65/100
162/162 [==============================] - 406s 3s/step - loss: 3.0247 - accuracy: 0.4352 - precision: 0.6104 - recall:
0.2078 - val_loss: 3.1001 - val_accuracy: 0.4211 - val_precision: 0.8571 - val_recall: 0.0067
Epoch 66/100
```

RNN-CNN – Changing learning rate

- Learning rate = 0.0001
  - Result: training takes too long, after 50 epochs, the training and validation accuracy are still around 0.2 -0.3 (therefore we manually stopped the training – due to time and computational resource limitation)

```
0.2098 - val_loss: 3.5299 - val_accuracy: 0.2728 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 54/100
162/162 [==============================] - 476s 3s/step - loss: 3.6334 - accuracy: 0.3634 - precision: 0.4270 - recall:
0.2094 - val_loss: 3.5286 - val_accuracy: 0.2772 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 55/100
162/162 [==============================] - 479s 3s/step - loss: 3.6367 - accuracy: 0.3598 - precision: 0.4151 - recall:
0.2050 - val_loss: 3.5274 - val_accuracy: 0.2767 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 56/100
162/162 [==============================] - 478s 3s/step - loss: 3.6393 - accuracy: 0.3626 - precision: 0.4225 - recall:
0.2085 - val_loss: 3.5283 - val_accuracy: 0.2739 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 57/100
162/162 [==============================] - 464s 3s/step - loss: 3.6289 - accuracy: 0.3640 - precision: 0.4228 - recall:
0.2109 - val_loss: 3.5265 - val_accuracy: 0.2756 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 58/100
162/162 [==============================] - 511s 3s/step - loss: 3.6229 - accuracy: 0.3693 - precision: 0.4308 - recall:
0.2097 - val_loss: 3.5260 - val_accuracy: 0.2756 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 59/100
 12/162 [=>............................] - ETA: 7:15 - loss: 3.6711 - accuracy: 0.3500 - precision: 0.4011 - recall: 0.19
08
```

Conclusion

- Learning rate 0.001 and 0.0001 would be too small, it takes too much time to converge and would stuck at the same place for a long time because of the small steps it takes to learn, therefore, we abandoned them during the training
- Learning rate 0.1 still have a poor performance during training, by comparing this with our 1st trial on RNN+CNN which did not have batch normalization, l2 regularization, and dropout, its performance is poorer
  - We consider this as a sign of over-regularization, and will try reducing them in the next batch of experiments
- Learning rate 0.01 outperform all previous experiment (validation accuracy =0.6325), that's why from now on, we will use a learning rate of 0.01

Optimizer

RNN-CNN –Changing Optimizer

- Optimizer = 'RMSprop", learning rate = 0.1
  - Result: loss decline quickly in the first few epochs and accuracy does not change much for the remaining epochs (therefore we manually stopped the training – due to time and computational resource limitation)

```
0.9745 - val_loss: 3.4754 - val_accuracy: 0.5506 - val_precision: 0.5554 - val_recall: 0.5483
Epoch 44/100
162/162 [==============================] - 457s 3s/step - loss: 0.0966 - accuracy: 0.9739 - precision: 0.9741 - recall:
0.9738 - val_loss: 3.4573 - val_accuracy: 0.5828 - val_precision: 0.5885 - val_recall: 0.5817
Epoch 45/100
162/162 [==============================] - 454s 3s/step - loss: 0.0968 - accuracy: 0.9743 - precision: 0.9745 - recall:
0.9740 - val_loss: 3.3523 - val_accuracy: 0.5606 - val_precision: 0.5626 - val_recall: 0.5567
Epoch 46/100
162/162 [==============================] - 467s 3s/step - loss: 0.0854 - accuracy: 0.9788 - precision: 0.9791 - recall:
0.9786 - val_loss: 3.8022 - val_accuracy: 0.5350 - val_precision: 0.5373 - val_recall: 0.5328
Epoch 47/100
162/162 [==============================] - 448s 3s/step - loss: 0.0893 - accuracy: 0.9760 - precision: 0.9763 - recall:
0.9760 - val_loss: 3.5658 - val_accuracy: 0.5550 - val_precision: 0.5562 - val_recall: 0.5522
Epoch 48/100
162/162 [==============================] - 486s 3s/step - loss: 0.0906 - accuracy: 0.9773 - precision: 0.9775 - recall:
0.9771 - val_loss: 3.5846 - val_accuracy: 0.5561 - val_precision: 0.5577 - val_recall: 0.5528
Epoch 49/100
  7/162 [>.............................] - ETA: 7:19 - loss: 0.0734 - accuracy: 0.9843 - precision: 0.9843 - recall: 0.98
43
```

3rd Batch of Experiments

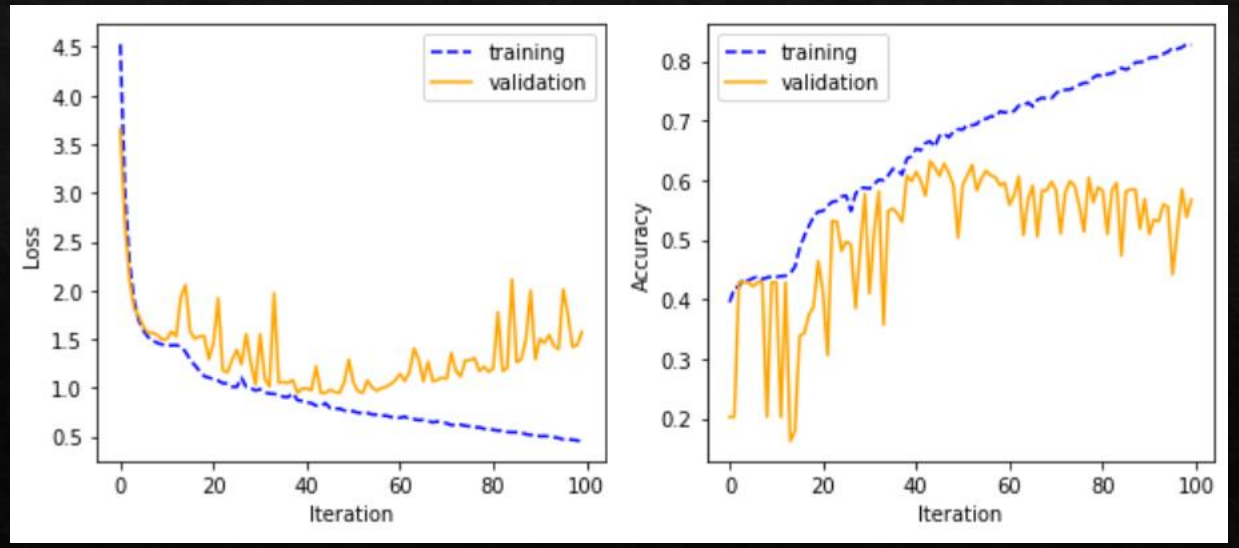
Background

- From the above experiments, we can see that all new trials of training (except the one with learning rate = 0.01) after adding the batch normalization, l2 regularization, and dropout have their performance dropped.
- Therefore, we think that maybe there is an over-regularization problem.
- For this batch of experiments, we will examine the effect of a reduction in l2 regularization, reduction of dropout rate, and not implementing batch normalization on the model.
- Since the learning rate = 0.01 performed the best in our 2nd batch of experiment, we have chose it for this batch of experiments

RNN-CNN – reduce l2 regularization

- Learning rate = 0.01, **l2_reg = 0.001** (instead of 0.005 originally), Epoch = 100, optimizer = "SGD", activation="relu"
  - Result: Performance declined

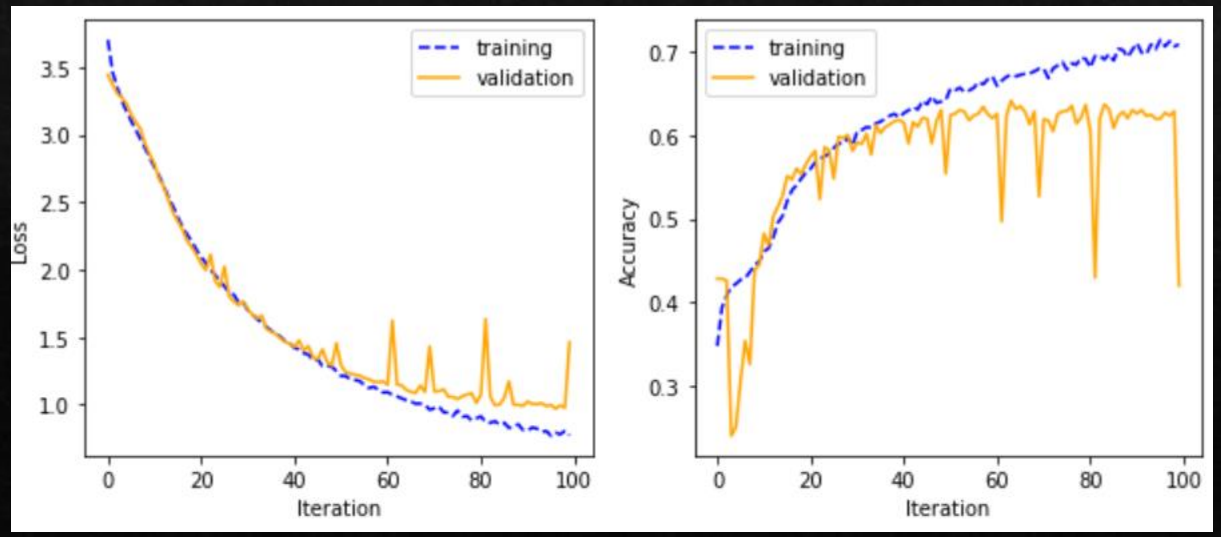
```
Train loss:  0.7705
Train accuracy:  0.6910
Train precision:  0.7678
Train recall:  0.5892
Train F1:  0.6667

Validation loss:  0.9236
Validation accuracy:  0.6275
Validation precision:  0.7281
Validation recall:  0.5410
Validation F1:  0.6208
```



RNN-CNN – reduce dropout rate

- Learning rate = 0.01, l2_reg = 0.005, Epoch = 100, optimizer = "SGD", activation="relu", **dropout_rate = 0.04** (instead of 0.05 originally)
  - Result: Performance enhanced, validation accuracy = 0.6515

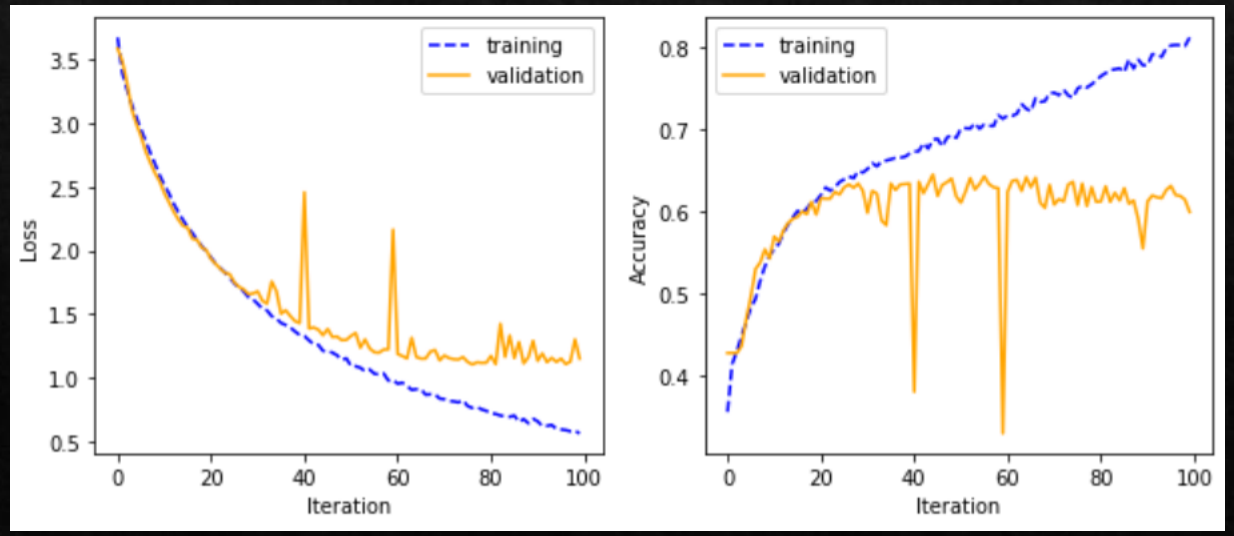
```
Train loss:  0.9978
Train accuracy:  0.6839
Train precision:  0.7554
Train recall:  0.5911
Train F1:  0.6632

Validation loss:  1.1293
Validation accuracy:  0.6515
Validation precision:  0.7168
Validation recall:  0.5580
Validation F1:  0.6275
```



RNN-CNN – reduce dropout rate

- Learning rate = 0.01, l2_reg = 0.005, Epoch = 100, optimizer = "SGD", activation="relu", **dropout_rate = 0.03** (instead of 0.05 originally)
  - Result: Performance enhanced, but poorer than having dropout_rate = 0.04

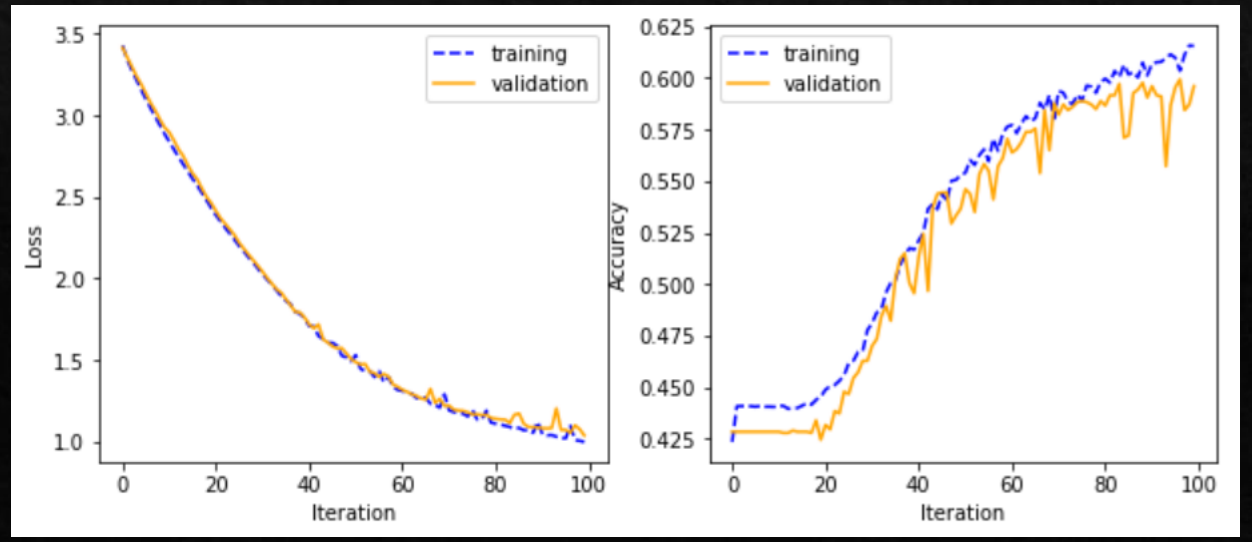
```
Train loss:  1.1420
Train accuracy:  0.7226
Train precision:  0.8001
Train recall:  0.6089
Train F1:  0.6916

Validation loss:  1.3255
Validation accuracy:  0.6440
Validation precision:  0.7347
Validation recall:  0.5360
Validation F1:  0.6198
```



RNN-CNN – no batch normalization

- Learning rate = 0.01, l2_reg = 0.005, Epoch = 100, optimizer = "SGD", activation="relu", dropout_rate = 0.05, **batch_normalization = False**
  - Result: from the graph, we can see it is underfitting, the accuracy is still having an increasing trend (however, due to time limitation, we will not continue training it)

```
Train loss:  1.0057
Train accuracy:  0.6210
Train precision:  0.7703
Train recall:  0.4422
Train F1:  0.5618

Validation loss:  1.0545
Validation accuracy:  0.5995
Validation precision:  0.7458
Validation recall:  0.4240
Validation F1:  0.5406
```



Conclusion

- By decreasing the dropout rate from 0.05 to 0.04, we've got an improvement in performance, this shows that over-regularization did incurred before
- We will stick with dropout rate = 0.04 from now on

Last Trial

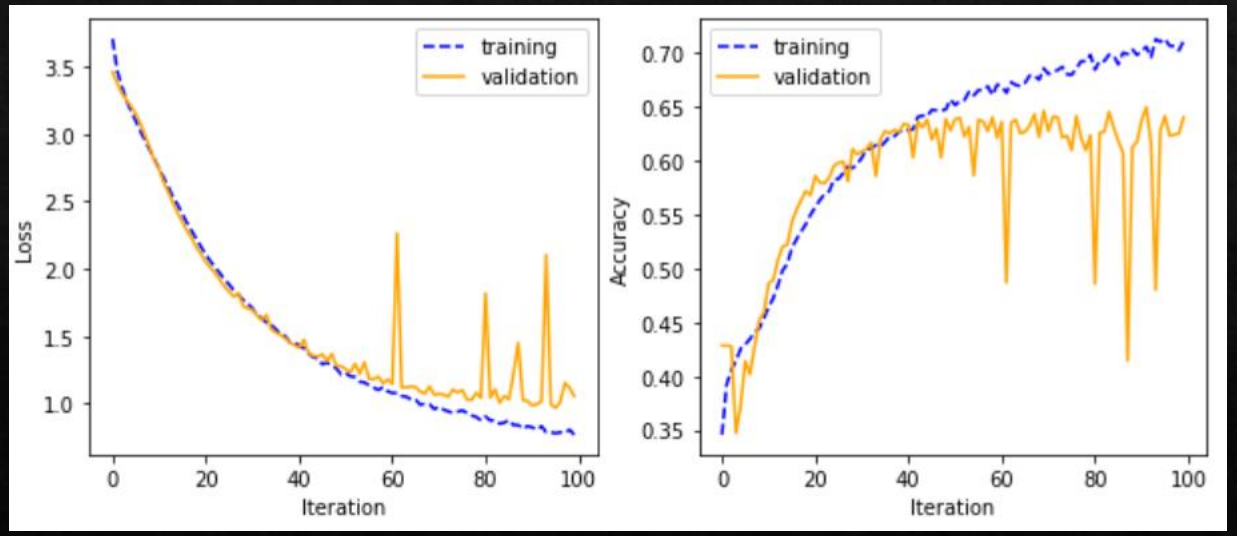
RNN+CNN – add ‘cool’, ‘funny’, ‘useful’

- From the previous batches of experiments, we observed that models having the following setup is having the best performance:
  - Learning rate = 0.01, l2_reg = 0.005, Epoch = 100, optimizer = “SGD”, activation=“relu”, **dropout_rate = 0.04** (instead of 0.05 originally)
- Now, we want to see if adding ‘cool’, ‘funny’, ‘useful’ as features would also improve the performance of RNN+CNN architecture just like 1-layer perceptron and CNN did

RNN+CNN – add ‘cool’, ‘funny’, ‘useful’

- Result: Performance dropped

```
Train loss:  0.7307
Train accuracy:  0.7366
Train precision:  0.7918
Train recall:  0.6622
Train F1:  0.7212

Validation loss:  0.9998
Validation accuracy:  0.6450
Validation precision:  0.7005
Validation recall:  0.5790
Validation F1:  0.6340
```

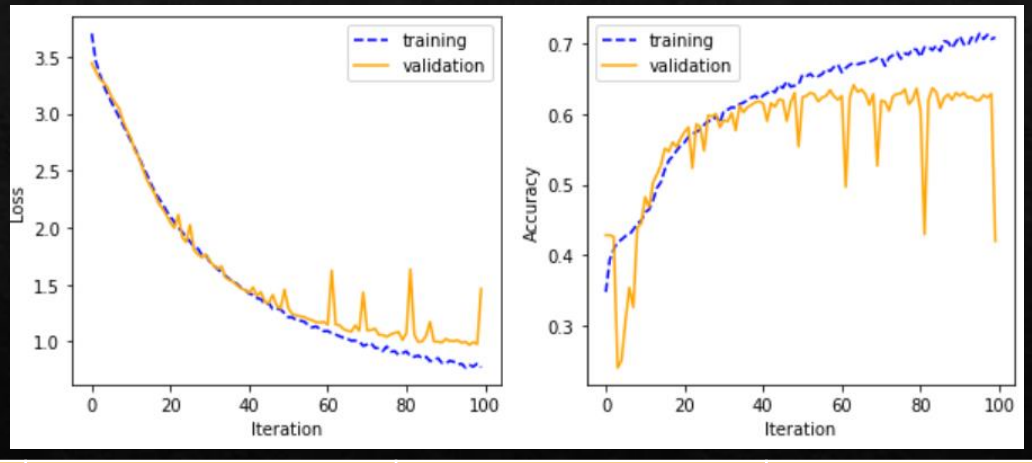


Best Model

Best Model

- From the 3rd batch of experiment, we got:
  - Learning rate = 0.01, l2_reg = 0.005, Epoch = 100, optimizer = "SGD", activation="relu", **dropout_rate = 0.04** (instead of 0.05 originally)

```
Train loss:  0.9978
Train accuracy:  0.6839
Train precision:  0.7554
Train recall:  0.5911
Train F1:  0.6632

Validation loss:  1.1293
Validation accuracy:  0.6515
Validation precision:  0.7168
Validation recall:  0.5580
Validation F1:  0.6275
```



Model\Validation Performance	Macro-F1	Precision	Recall	Accuracy
RNN+CNN (dropout rate 0.4)	0.6275	0.7168	0.5580	0.6515
Strong Baseline	**0.5370**	**0.5509**	**0.5324**	**0.6500**

Conclusion: Model Comparison

Model Comparison

- **The metrics tools we used before are based on tf.keras.metrics**, here, we will use evaluate.py to evaluate the performance of our best model in each section
- Best Model: CNN (added 'funny', 'cool', 'useful') – all metrics passed the strong baseline

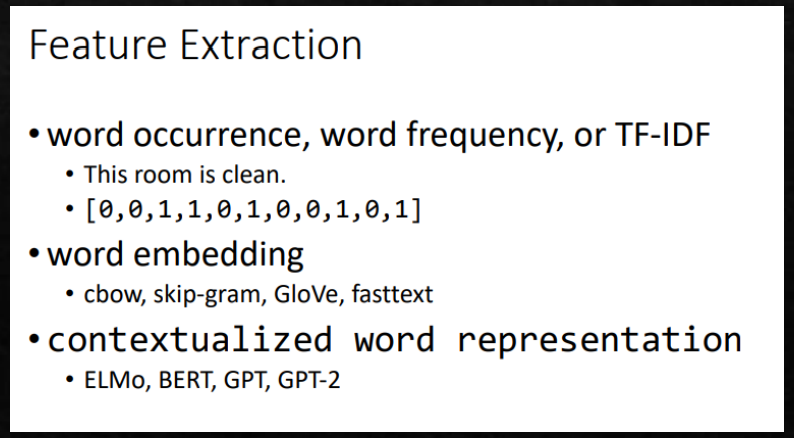
Model\Validation Performance	Macro-F1	Precision	Recall	Accuracy
1-Layer-Perceptron (added 'funny', 'cool', 'useful')	0.5422	0.5789	0.5305	0.6590
CNN (added 'funny', 'cool', 'useful')	0.5609	0.5688	0.5682	0.6620
RNN (Bi-directional LSTM)	0.4785	0.5101	0.4818	0.5995
RNN + CNN (dropout rate 0.4)	0.5105	0.5605	0.5114	0.6515

Strong Baseline	**0.5370**	**0.5509**	**0.5324**	**0.6500**

Appendix
(not useful)

Appendix Notes

- Feature Extraction/Selection
  - Evaluate relationship between each input variable and the target variable & Select those input variables that have the strongest relationship with the target variable
  - Text data: word occurrence, word frequency, TF-IDF, word embedding, N-gram, stop word, contextualized word representation

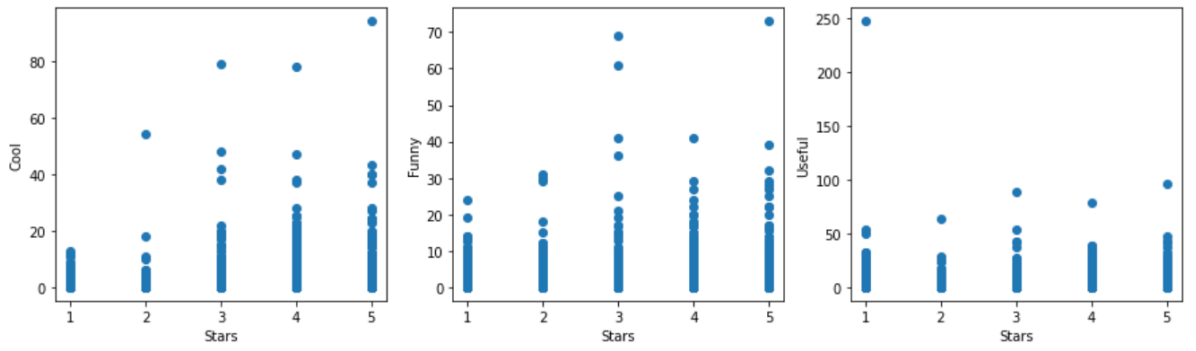


https://machinelearningmastery.com/feature-selection-with-real-and-categorical-data/

Appendix Notes

- Pandas
  - pd.read_csv(filename) returns a dataframe
  - df.loc[rows, columns]
    - E.g., df.loc[:,['text','stars']] selects all rows of column 'text' & 'stars'
- Matplotlib.pyplot
  - plt.subplot(row,column,curr_column)

```
plt.figure(figsize=(15,4))
plt.subplot(1,3,1)
plt.scatter(train_df['stars'], train_df['cool'])
plt.xlabel("Stars")
plt.ylabel("Cool")
plt.subplot(1,3,2)
plt.scatter(train_df['stars'], train_df['funny'])
plt.xlabel("Stars")
plt.ylabel("Funny")
plt.subplot(1,3,3)
plt.scatter(train_df['stars'], train_df['useful'])
plt.xlabel("Stars")
plt.ylabel("Useful")
plt.show()
```



Appendix Notes

- One-hot:
  - Demands large storage space for the word vectors and reduces model efficiency.
  - Embedding layer
    - Enables us to convert each word into a fixed length vector of defined size.
    - The resultant vector is a dense one with having real values instead of just 0's and 1's.
    - The fixed length of word vectors helps us to represent words in a better way along with reduced dimensions.

https://medium.com/analytics-vidhya/understanding-embedding-layer-in-keras-bbe3ff1327ce

Appendix Notes

- Convolutional Neural Network (CNN) have proven to be successful at document classification problems
- A conservative CNN configuration:
  - 32 filters (parallel fields for processing words), Kernel size of 8, RELU activation function
  - Pooling layer that reduces the output of the convolutional layer by half.
  - The 2D output from the CNN part of the model flatten to one long 2D vector to represent the 'features' extracted by the CNN.
  - Standard Multilayer Perceptron layers to interpret the CNN features.
  - Output layer uses a sigmoid activation function to output a value between 0 and 1 for the negative and positive sentiment in the review. Adam optimizer is used.
- A downside of learning a word embedding as part of the network is that it can be very slow, especially for very large text datasets.
- The word2vec algorithm is an approach to learning a word embedding from a text corpus in a standalone way. The benefit of the method is that it can produce high-quality word embeddings very efficiently, in terms of space and time complexity.

https://machinelearningmastery.com/develop-word-embedding-model-predicting-movie-review-sentiment/

Appendix Notes

- Best practice for document classification deep learning
  - https://machinelearningmastery.com/best-practices-document-classification-deep-learning/
- RNN for text classification
  - https://towardsai.net/p/deep-learning/text-classification-with-rnn?amp=1

Appendix Notes

- In the paper on text classification by Yijun Xiao and Kyunghyun Cho, the authors even suggest that maybe all pooling/subsampling layers can be replaced by recurrent layers.
  - http://arxiv.org/abs/1602.00367v1

https://yerevann.github.io/2016/06/26/combining-cnn-and-rnn-for-spoken-language-identification/#combinations-of-cnn-and-rnn

Questions in Mind

- How to treat the predicted variable? (Categorical, Ordinal, Numerical?)
- How to do resampling (imbalanced data)?
- How to include 'useful' feature in training?
- Are the stop-words suitable for this problem?
- How are the punctuations related to the target variables?

Useful Source (NLP Pipeline with Code): https://towardsdatascience.com/text-analysis-feature-engineering-with-nlp-502d6ea9225d