Problem 1

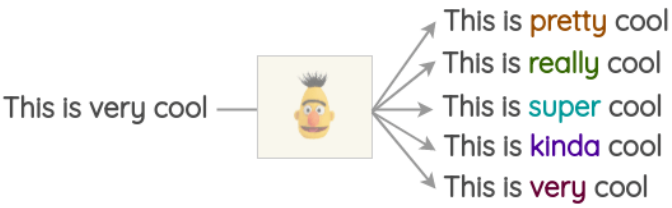
Reference: https://amitness.com/2020/05/data-augmentation-for-nlp/
https://neptune.ai/blog/data-augmentation-nlp

1.1 Lexical Substitution

In a line of text, it is acceptable to take a random word from the sentences and replace it with its synonym. And there are different ways to do so. First, we can use a thesaurus. For example, the WordNet dataset is huge online dictionary for us to loop up the synonyms and then perform the replacement. For example, as the picture below shows, the "awesome" in text "It is awesome" is substituted as "amazing", which is found in the Wordnet. Of course, word "awe-inspiring" and "awing" could also complete this job.

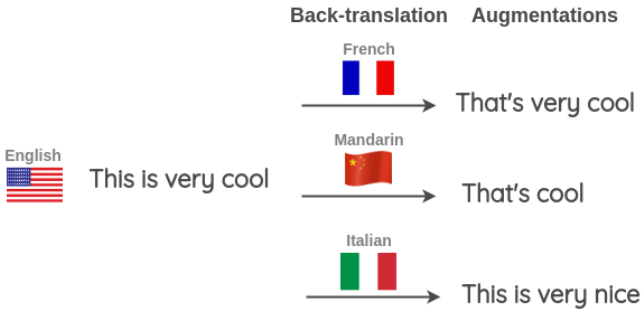
Instead to lookup a synonym, a near word embedding vector can also substitute it. There are many word-embedding approaches such as Word2Vec, GloVe and FastText. They map the word into vector space. Then, in the vector space, we can find its k-nearest neighbors' vector and use these neighbors' words to replace the target words.

And we can also use Masked Language Model such as BERT. Since it is trained on a large amount of text data using Masked Language Modeling" where the model has to predict masked words based on the context. In such case, we can Mask some words in our text data and use the pre-trained model to predict the masked sentences. For example, we could use a pre-trained BERT model, mask some parts of the text, and ask the BERT model to predict the token for the mask.



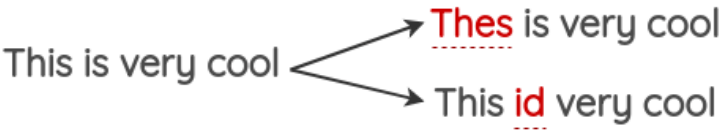
1.2 Back Translation

The machine translation can be used to augment text data while maintain the meaning of text data, either. It is useful and intuitive. The process is Take some sentence (e.g., in English) and translate to another Languages, for example, Chinese. It can be done using some pre-trained translation models even google translate. Then, for the translated sentences in Chinese, we translate them back into English. Check if the new sentence is different from our original sentence. If it is, then we use this new sentence as an augmented version of the original text. In addition, you can also use multi-languages and get multi-languages versioned sentences, such as French, Japanese, Russia. The following figure demonstrates the process that translate the English sentence "This is very cool" to French, Chinese, and Italian. And translate back again to English to get slightly different senteces.

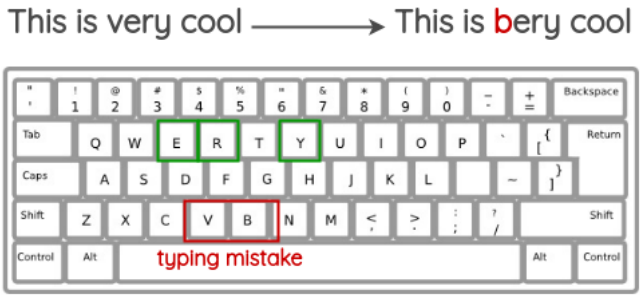


1.3 Random Noise Injection

The idea of these methods is to inject noise in the text so that the model trained is robust to perturbations. First, a method called spelling error injection in used. This method simulates the "typo" of human input. It adds spelling errors to random words in the text. These spelling errors can be added programmatically or using a mapping of common spelling errors. Figure below demonstrates the noise injection on "this" and "is"



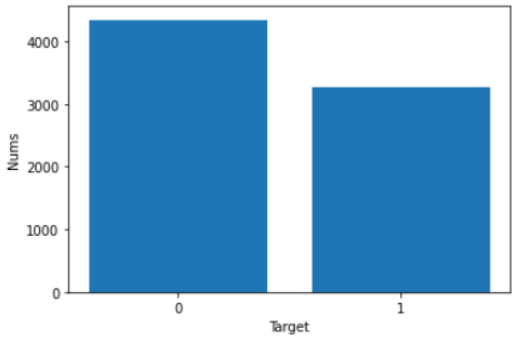
Another method is QWERTY keyboard error injection. This method tries to simulate common errors that happen when typing on a QWERTY layout keyboard due to keys that are very near to each other. The errors are injected based on keyboard distance. For example, the image show the area that human often make mistakes on.



The last, Unigram Noising was proposed. The idea is to replace a random word with a placeholder token. The paper uses "_" as the placeholder token. In the paper, they use it as a way to avoid overfitting on specific contexts as well as a smoothing mechanism for the language model. The technique helped improve perplexity and BLEU scores.

Problem 2

Task1



The figure is the histogram of the target class distribution of data. Obviously, Target class =1 is the imbalanced class

Task2

To avoid appending augmented text into validation data, sklearn function train_test_split is used to split the training set and validation set using proportion of 8:2. Then, two augmentation methods are used: Synonym based word augmentation and Random Sentence Augmentation.

As described in Problem 1, the basic idea of Synonym based lexical substitution is to substitute a word using its synonyms. Algorithm 1 describes the pseudo-code.

Algorithm 1 Synonym based Data Augmentation

```
Input : text sentence s.
Output: augmented text sentences s'

Define
Thesaurus = Wordnet
Stop word list from NLTK
Probability of a word chosen p = 0.3
New sentence s' = ""
For each word w in s:
    If w not in stop list:
        If a random number >= p:
            Synonyms = Wordnet[w] # a list of synonyms
            w' = a synonym that randomly taken from Synonyms with uniform prob.
        Else:
            w' = w
        Append w' to s'
```

The other data augmentation method used is Random Sentence Augmentation. Algorithm 2 demonstrates the pseudocode of it. Note that p1 = p2 = p3 = 0.1, k = 3.

Algorithm 2 Random Sentence Augmentation.

Input : text sentence s, swap prob p1, delete prob p2, substitute prob p3,k
Output: augmented text sentences s'

Define
Thesaurus = Wordnet
Stop word list from NLTK

Generate a random number p range (0,1) using uniform distribution

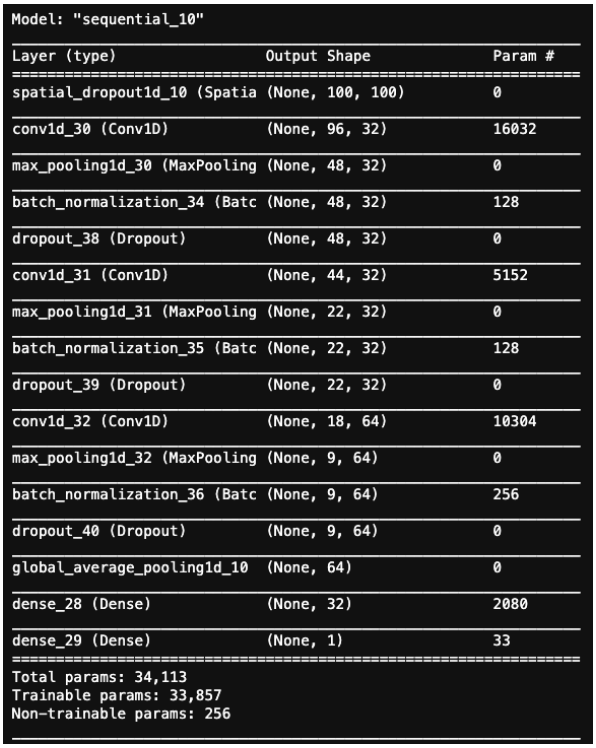
If p locates in 0~p1:
Randomly choose k pair of "neighbor words", which are next to each other
Swap words in each pair of neighbor words
Return new text s'
If p locates p1 ~ p2:
Randomly choose k words and delete them.
Return new text s'
If p locates p2~p3:
Randomly choose k words
Substitute those words which are not in stop list using synonyms
Return new text s'
Return s

500 / 800 Augmented Examples chosen from training data with label = 1 are test in the model prediction part.

Task3

I use the Glove_100d.txt as the pretrained embedding layer to map the word to vector. Before mapping, I transform all letters to lowercase, remove all punctuations, and fill the sequence to length = 100 using zero padding.

Task4



```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
spatial_dropout1d_10 (Spatia (None, 100, 100)          0
_________________________________________________________________
conv1d_30 (Conv1D)           (None, 96, 32)            16032
_________________________________________________________________
max_pooling1d_30 (MaxPooling (None, 48, 32)            0
_________________________________________________________________
batch_normalization_34 (Batc (None, 48, 32)            128
_________________________________________________________________
dropout_38 (Dropout)         (None, 48, 32)            0
_________________________________________________________________
conv1d_31 (Conv1D)           (None, 44, 32)            5152
_________________________________________________________________
max_pooling1d_31 (MaxPooling (None, 22, 32)            0
_________________________________________________________________
batch_normalization_35 (Batc (None, 22, 32)            128
_________________________________________________________________
dropout_39 (Dropout)         (None, 22, 32)            0
_________________________________________________________________
conv1d_32 (Conv1D)           (None, 18, 64)            10304
_________________________________________________________________
max_pooling1d_32 (MaxPooling (None, 9, 64)             0
_________________________________________________________________
batch_normalization_36 (Batc (None, 9, 64)             256
_________________________________________________________________
dropout_40 (Dropout)         (None, 9, 64)             0
_________________________________________________________________
global_average_pooling1d_10  (None, 64)                0
_________________________________________________________________
dense_28 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_29 (Dense)             (None, 1)                 33
=================================================================
Total params: 34,113
Trainable params: 33,857
Non-trainable params: 256
_________________________________________________________________
```

The figure summarizes the structure of my model. Since embedding part is done using Glove, there is no need to create an embedding layer. Following the embedding part, a spatial dropout with probability = 0.1 is used. Following is two convolutional 1D block, each has 32 convolutional1D layer with kernel size = 5*5, a MaxPooling1D layer, a batch normalization layer, and a dropout layer with probability = 0.25. A global average Pooling 2D is used to flatten the feature map, followed by 32 dense layer and 1 dense layer as output unit.

sklearn function train_test_split is used to split the training set and validation set using proportion of 8:2. The experiment settings are described as follows:

Epochs = 50

Batch size = 128

Optimizer = Adam

Learning Rate : initial = 0.003, drop to its 95% every 2 epoch.

Early Stop: The training will stop if the validation loss stops decreasing for 8 epochs

Task5

Performance on Data without Augmentation:

	precision	recall	f1-score	support
0	0.78	0.91	0.84	874
1	0.84	0.65	0.73	649
accuracy			0.80	1523
macro avg	0.81	0.78	0.79	1523
weighted avg	0.81	0.80	0.79	1523

AUC Score = 0.7805

Performance on Data with 800 Augmented texts is the best:

	precision	recall	f1-score	support
0	0.79	0.91	0.85	874
1	0.84	0.68	0.75	649
accuracy			0.81	1523
macro avg	0.82	0.79	0.80	1523
weighted avg	0.81	0.81	0.81	1523

AUC Score = 0.7926

The reason is intuitive. The data augmentation maintains the relative balance between each label. The model will not skew to any label. Furthermore, the data augmentation increases the data diversity on the training set, which allows model to learn better.