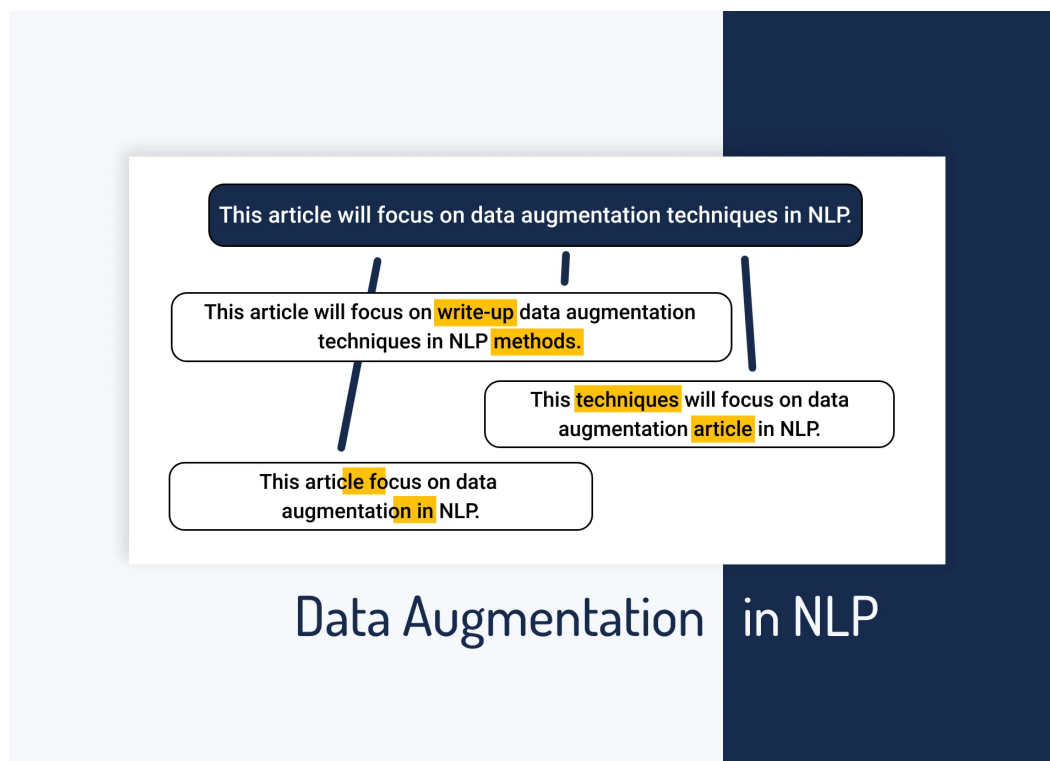


Data Augmentation in NLP: Best Practices From a Kaggle Master



There are many tasks in NLP from text classification to question answering but whatever you do the amount of data you have to train your model impacts the model performance heavily.

What can you do to make your dataset larger?

Simple option -> Get more data :).

But acquiring and labeling additional observations can be an expensive and time-consuming process.

What you can do instead?

Apply data augmentation to your text data.

Data augmentation techniques are used to **generate additional, synthetic data using the data you have**. Augmentation methods are super popular in computer

vision applications but they are just as powerful for NLP.

In this article, we'll go through all the major data augmentation methods for NLP that you can use to increase the size of your textual dataset and improve your model performance.

Data augmentation for vision vs NLP

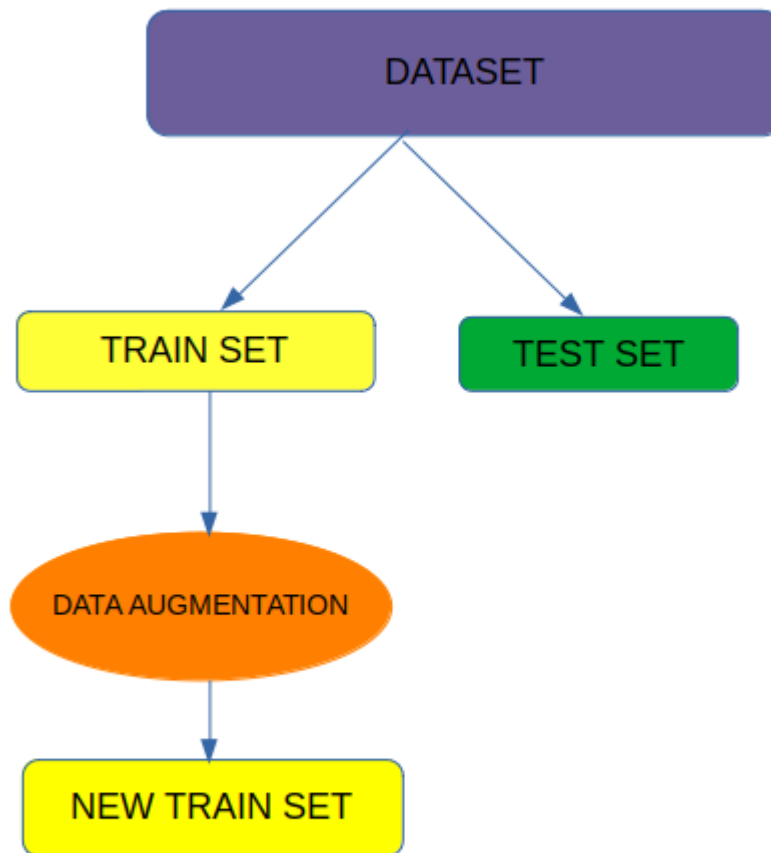
In computer vision applications data augmentations are done almost everywhere to get larger training data and make the model generalize better.

The main methods used involve:

- cropping,
- flipping,
- zooming,
- rotation,
- noise injection,
- and many others.

In **computer vision**, these transformations are **done on the go** using data generators. As a batch of data is fed to your neural network it is randomly transformed (augmented). You don't need to prepare anything before training.

This isn't the case with **NLP**, where data augmentation should be done carefully due to the grammatical structure of the text. The methods discussed here are used **before training**. A new augmented dataset is generated beforehand and later fed into data loaders to train the model.



Data Augmentation methods

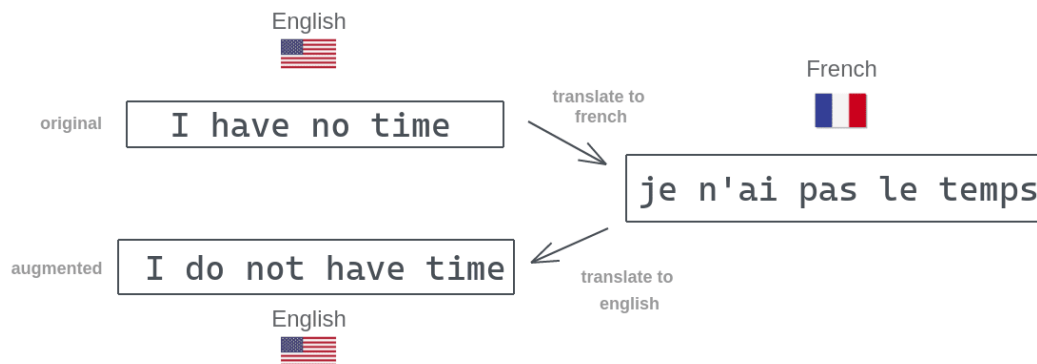
In this article, I will mainly focus on NLP data augmentation methods provided in the following projects:

So, let's dive into each of them.

Back translation

In this method, we **translate the text data to some language and then translate it back to the original language**. This can help to generate textual data with different words while preserving the context of the text data.

Language translations APIs like google translate, Bing, Yandex are used to perform the translation. For example, given the sentence:



You can see that the sentences are not the same but their content remains the same after the back-translation. If you want to try this method for a dataset you can use [this notebook](#) as reference.

Easy Data Augmentation

Easy data augmentation uses traditional and very simple data augmentation methods. EDA consists of **four simple operations that do a surprisingly good job** of preventing overfitting and helping train more robust models.

- **Synonym Replacement**

Randomly choose n words from the sentence that are not stop words. Replace each of these words with one of its synonyms chosen at random.

For example, given the sentence:

*This **article** will focus on summarizing data augmentation **techniques** in NLP.*

The method randomly selects n words (say two), the words *article* and *techniques*, and replaces them with *write-up* and *methods* respectively.

*This **write-up** will focus on summarizing data augmentation **methods** in NLP.*

- **Random Insertion**

Find a random synonym of a random word in the sentence that is not a stop word. Insert that synonym into a random position in the sentence. Do this n times.

For example, given the sentence:

*This **article** will focus on summarizing data augmentation **techniques** in NLP.*

The method randomly selects n words (say two), the words *article* and *techniques* find the synonyms as *write-up* and *methods* respectively. Then these synonyms are inserted at a random position in the sentence.

*This article will focus on **write-up** summarizing data augmentation techniques in NLP **methods**.*

- **Random Swap**

Randomly choose two words in the sentence and swap their positions. Do this n times.

For example, given the sentence

*This **article** will focus on summarizing data augmentation **techniques** in NLP.*

The method randomly selects n words (say two), the words *article* and *techniques* and swaps them to create a new sentence.

*This **techniques** will focus on summarizing data augmentation **article** in NLP.*

- **Random Deletion**

Randomly remove each word in the sentence with probability p .

For example, given the sentence

*This **article** will focus on summarizing data augmentation **techniques** in NLP.*

The method selects n words (say two), the words *will* and *techniques*, and removes them from the sentence.

*This **article** focus on summarizing data augmentation in NLP.*

You can go to this [repository](#) if you want to apply these techniques to your projects.

NLP Albumentation

Previously, we talked about differences between computer vision data augmentation and NLP data augmentation. But in this section, we will see how we can apply some of the ideas used in CV data augmentation in NLP.

For that, we will use the [Albumentations](#) package.

Let's take a look at a couple of the techniques here.

- **Shuffle Sentences Transform**

In this transformation, if the given text sample contains multiple sentences these sentences are shuffled to create a new sample.

For example:

**text = '<Sentence1>. <Sentence2>. <Sentence4>. <Sentence4>. <Sentence5>.
<Sentence5>.'**

Is transformed to:

**text = '<Sentence2>. <Sentence3>. <Sentence1>. <Sentence5>. <Sentence5>.
<Sentence4>.'**

- **Exclude duplicate transform**

In this transformation, if the given text sample contains multiple sentences with duplicate sentences, these duplicate sentences are removed to create a new sample.

For example given the sample,

**text = '<Sentence1>. <Sentence2>. <Sentence4>. <Sentence4>. <Sentence5>.
<Sentence5>.'**

We transform it to:

'<Sentence1>. <Sentence2>.<Sentence4>. <Sentence5>.'

There are many other transformations which you can try with this library. You can check this wonderful [notebook](#) to see the complete implementation.

NLPAug Library

Until now we have discussed many methods by which data augmentation can be used in NLP.

But effectively implementing these methods from scratch is a lot of work.

In this section, I will introduce you to a **python package that lets you do all these data augmentation easily** and you can tune the level of augmentation you

need using various arguments.

[NLPAug](#) helps you with augmenting NLP for your machine learning projects. Let's see how we can use this library to perform data augmentation.

NLPAug offers three types of augmentation:

- Character level augmentation
- Word level augmentation
- Sentence level augmentation

In each of these levels, NLPAug provides all the methods discussed in the previous sections such as:

- random deletion,
- random insertion,
- shuffling,
- synonym replacement,
- etc.

From my experience, the **most commonly used and effective technique is synonym replacement via word embeddings**.

We replace n number words with its synonyms (word embeddings that are close to those words) to obtain a sentence with the same meaning but with different words.

While performing synonym replacement we can choose which pre-trained embedding we should use to find the synonyms for a given word.

With NLPAug we can choose non-contextual embeddings like:

- Glove,
- word2vec,
- etc,

or contextual embeddings like:

- Bert,
- Roberta,
- etc.

For example:

```
aug = naw.ContextualWordEmbsAug(  
    model_path='bert-base-uncased', action="insert")  
augmented_text = aug.augment(text)
```

Original:

The quick brown fox jumps over the lazy dog

Augmented Text:

*even the quick brown fox **usually** jumps over the lazy dog*

Things to keep in mind while doing NLP

Data Augmentation

As I said in the introduction, there are certain things that we need to be careful of while doing augmentation in NLP.

The **main issue faced when training on augmented data** is that algorithms, when done incorrectly, is that you heavily overfit the augmented training data.

Some things to keep in mind:

- Do not validate using the augmented data.
- If you're doing K-fold cross-validation, always keep the original sample and augmented sample in the same fold to avoid overfitting.
- Always try different augmentation approaches and check which works better.
- A mix of different augmentation methods is also appreciated but don't overdo it.
- Experiment to determine the optimal number of samples to be augmented to get the best results.
- Keep in mind that data augmentation in NLP does not always help to improve model performance.

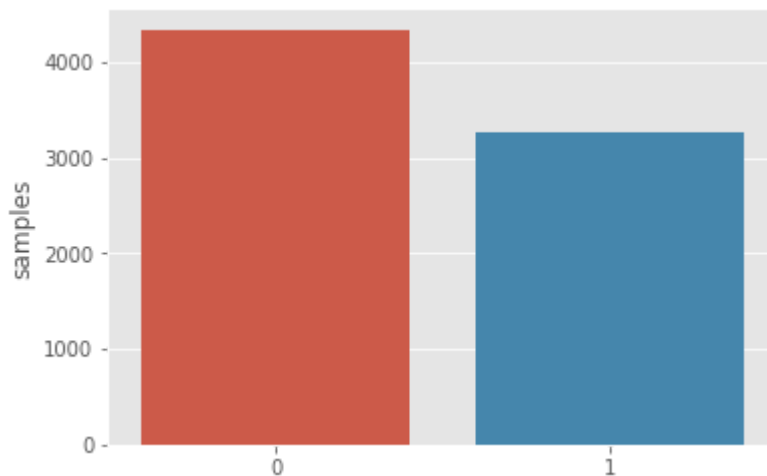
Data Augmentation workflow

In this section, we will try data augmentation on [Real or Not? NLP with Disaster Tweets](#) competition hosted on Kaggle.

In [one of my previous posts](#), I used the data from this competition to try different non-contextual embedding methods. Here, I will use the very same classification pipeline I used there but I will add data augmentation to see if it improves the model performance.

First, let's load the training dataset and check the target class distribution.

```
...  
x=tweet.target.value_counts()  
sns.barplot(x.index,x)  
plt.gca().set_ylabel('samples')
```



We can see that there is a small class imbalance here.

Let's generate some positive samples using the synonym replacement method.

Before data augmentation, we split the data into the train and validation set so that no samples in the validation set have been used for data augmentation.

```
train,valid=train_test_split(tweet,test_size=0.15)
```

Now, we can do data augmentation of the training dataset. I have chosen to generate 300 samples from the positive class.

```
def augment_text(df,samples=300,pr=0.2):  
    aug_w2v.aug_p=pr  
    new_text=[]  
  
    ##selecting the minority class samples  
    df_n=df[df.target==1].reset_index(drop=True)
```

```

## data augmentation loop
for i in tqdm(np.random.randint(0, len(df_n), samples)):

    text = df_n.iloc[i]['text']
    augmented_text = aug_w2v.augment(text)
    new_text.append(augmented_text)

## dataframe
new=pd.DataFrame({'text':new_text, 'target':1})
df=shuffle(df.append(new).reset_index(drop=True))
return df

```

```
train = augment_text(train)
```

We can now use this augmented text data to train the model.

If you are interested in learning how to build the entire pipeline from data preparation for NLP, training a classifier, and running inference you can [check my other article](#).

So did data augmentation with synonym replacement work?

Without Data Augmentation With Data Augmentation

ROC AUC score 0.775

0.785

With data augmentation, we got a good boost in the model performance (AUC).

Playing with different techniques and tuning hyperparameters of the data augmentation methods can improve results even further but I will leave it for now.

If you'd like to do that I prepared a [notebook where you can play with things](#).

Final thoughts

In this article, we discussed and implemented different data augmentation methods for textual data.

To my knowledge, these are the best publically available techniques and packages to do the task.

Hopefully, you will find them useful in your projects.