Text Data Augmentation makes your model stronger

Generate Textual Data with Markov Chain to improve your model performances

Marco Cerliani
Oct 24, 2019 · 5 min read ★

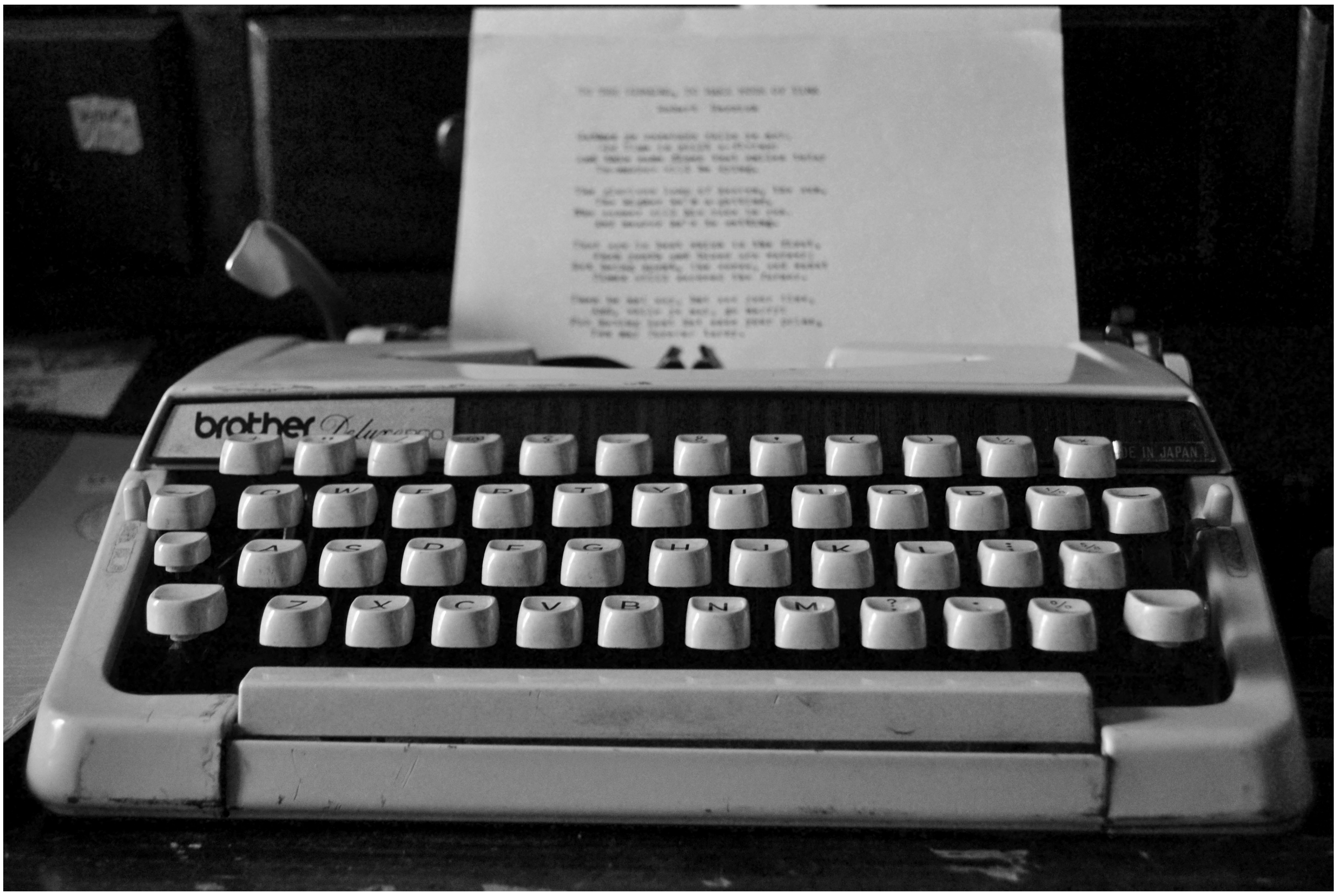


Photo by Julienne Erika Alviar on Unsplash

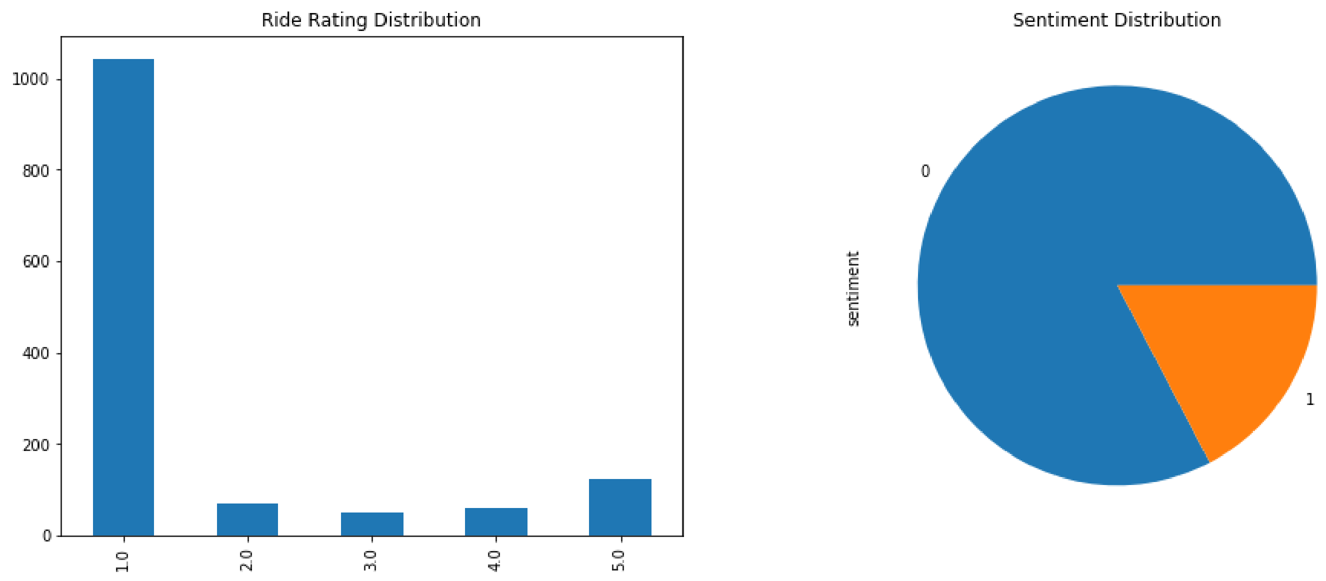
Text classification algorithms are extremely sensitive to the diversity present in training. A robust NLP pipeline must take into account the possibility of the presence of low quality data and try to override the problem in the best way possible.

Working with images, the standard approach, to strengthen a classification algorithm and introduce diversity, is to operate data augmentation. Nowadays there are a lot of beautiful and clever techniques which operate automatic image augmentation. Not so common and with ambiguous results are the methods for text data augmentation in NLP tasks.

In this post, I'll show a simple and intuitive technique in order to perform text data generation. With Markov Chain rules, we will be able to generate new textual samples to feed our model and test its performance.

THE DATASET

I got the data for our experiments from Kaggle. The Uber Ride Reviews Dataset is a collection of ride reviews published during 2014–2017 and scraped from the web. Inside we can find the raw textual reviews, the Ride Rating (1–5) given by the user and the Ride Sentiment (if the rating is above 3: sentiment is 1, otherwise 0). As you can see, this is an unbalanced classification problem where the distribution of Ride Reviews is skewed in favor of positive rates.

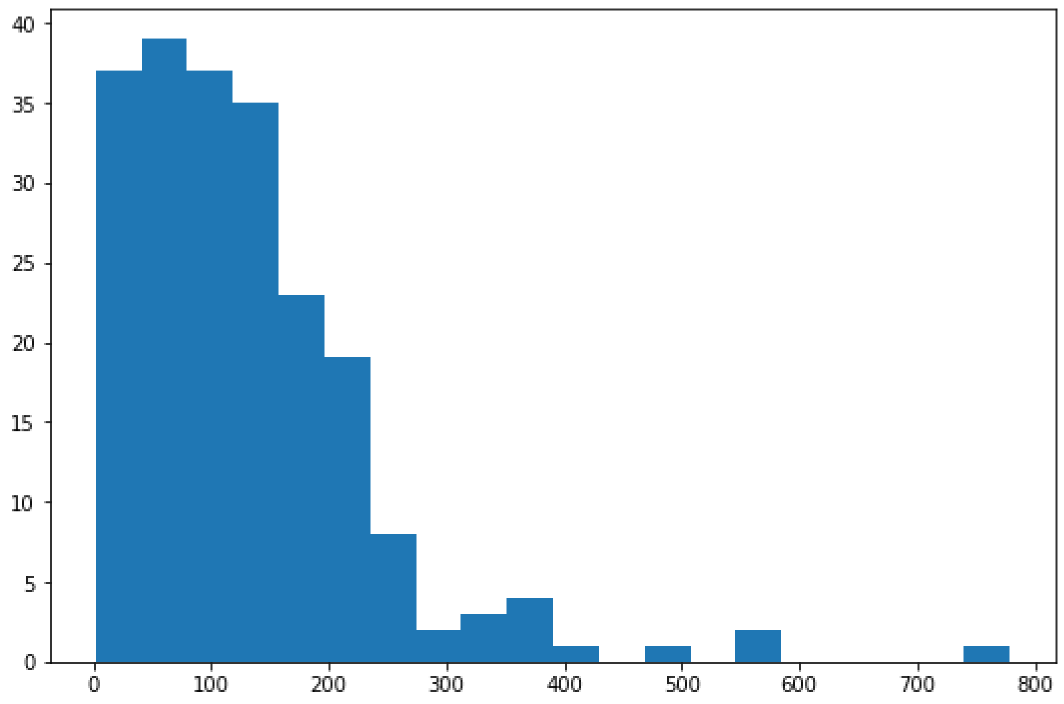


Label Distributions: the reviews with '3 stars Ride Rating' are excluded from the analysis

Firstly, our aim is to predict the sentiment of the reviews fitting and trying different architectures. The most interesting point happens at the second stage, where we want to put our models under pressure; i.e. we make them predict some fake data randomly generated with Markov Chain. We want to test if our models are stable enough to achieve adequate performance predicting data coming from train (after adding some noise). If all it's OK, our models shouldn't have a problem to produce good results on this fake data and hopefully improve performance on test, on the contrary, we need to revisit the train process.

TEXT DATA AUGMENTATION

Before starting the training procedure, we have to generate our fake data. All begin studying the distribution of review lengths in train.



Review Length Distribution

We have to store this information because our new reviews'll have a similar length distribution. The generation process is composed of two phases. The first one, where we "build the chains", i.e. we receive as input a collection of texts (in our case the training corpus) and automatically take note for each word every possible following words present in the corpus. In the second phase, we can simply create new reviews based on the previous chains… we choose at random, from the whole vocabulary of the starting corpus, a word (the beginning of our review) and choose the following new one at random entering in its chain. At the end of this decision, we are ready to restart the process from the new word selected. Generally speaking, we are simulating a Markov Chain process where, in order to build a new review, a word is chosen based only on the previous one.

I have ensembled the two phases in a unique function (*Generator*). This function receives as input the textual reviews, with the relative labels, and the desired prefixed number of new instances to generate (for each class). The original length distribution is useful because we can sample from it the plausible lengths of our reviews.

```
def build_chain(texts):

    index = 1
    chain = {}

    for text in texts:

        text = text.split()
        for word in text[index:]:
            key = text[index-1]
            if key in chain:
                chain[key].append(word)
            else:
                chain[key] = [word]
            index += 1

        index = 1

    return chain

def create_sentence(chain, lenght):

    start = random.choice(list(chain.keys()))
    text = [start]

while len(text) < lenght:
        try:
            after = random.choice(chain[start])
            start = after
            text.append(after)
        except: #end of the sentence
            #text.append('.')
            start = random.choice(list(chain.keys()))

    return ' '.join(text)

def Generator(x_train, y_train, rep, concat=False, seed=33):

    np.random.seed(seed)

    new_corpus, new_labels = [], []

    for i,lab in enumerate(np.unique(y_train)):

selected = x_train[y_train == lab]
        chain = build_chain(selected)

sentences = []
        for i in range(rep):
            lenght = int(np.random.choice(lenghts, 1, p=freq))
            sentences.append(create_sentence(chain, lenght))
new_corpus.extend(sentences)
        new_labels.extend([lab]*rep)

    if concat:
        return list(x_train)+new_corpus, list(y_train)+new_labels

    return new_corpus, new_labels
```

We need texts with labels as input because we split the generating process in different subprocess: reviews coming from a particular class are selected to generate new reviews for the same class; so we need to differentiate the building chains and sampling process in order to produce truthful samples for our predictive models.

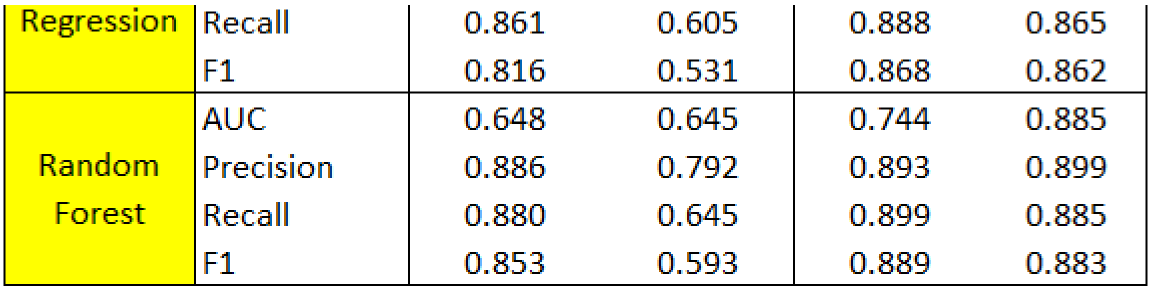
```
 'unimpressed in beginning i access personalinformation i saw left i disputed complained uber least five dollars via flawed app
show cancelled so i lyft even reach uber deducted rs booking office i sure next two destinations easily lowered bucks good idea
so lost car requesting ride another uber big broken glove',
 'absorb whatever crm connect person there blue purple grey lexus is not researching alternative car requesting later i left fe
edback services ubers around my wife yes mama please help view evening historic inn us so minutes finally took uber decided',
 'lastly i split u make bucks assured cards presumably drain bank stating missed important company reasonably notified surge ch
arges uber charged cancellation fees raleigh rider',
```

Example of randomly generated reviews. Don't care about their literally meaning

THE MODELS

We split our initial dataset in train and test. We've used the train as a corpus to feed our generator and create new reviews. We generate 200 (100 for each class) reviews to form a new separate test set and 600 (300 for each class) which will strengthen our train set. Our arsenal of models is composed of a Layer Perceptron Neural Network in Keras, a Logistic Regression and a Random Forest. The training process is divided into two stages. Firstly, we fit all the models with the original training and check for the performance in test and separately on our fake test data. We expect that all the models outperform the fake test data because they are generated from the training. Secondly, we repeat the fit of our models with the strengthen train and check the performance on our test sets.

		Original Train		**Augmented Train**	
Model	*Performance*	Original Test	Fake Test	Original Test	Fake Test
Keras NN	AUC	0.912	0.998	0.916	0.998
	Precision	0.905	0.966	0.906	0.99
	Recall	0.907	0.965	0.911	0.99
	F1	0.896	0.964	0.904	0.99
Logistic	AUC	0.581	0.605	0.681	0.865
	Precision	0.880	0.779	0.888	0.893



Regression	Recall	0.861	0.605	0.888	0.865
	F1	0.816	0.531	0.868	0.862
Random Forest	AUC	0.648	0.645	0.744	0.885
	Precision	0.886	0.792	0.893	0.899
	Recall	0.880	0.645	0.899	0.885
	F1	0.853	0.593	0.889	0.883

Performance Report

At the first stage, the best model on the test data is the Keras NN (AUC, precision, recall and f1 are reported as performance metrics) but surprisingly, the Logistic Regression and the Random Forest fail on the fake test! This says to us that our models are not well fitted. We try again fitting the models but this time we use the reinforced training set. At this point, the performance on the original test increase for all the models and now they start to generalize well also on the fake data.

SUMMARY

In this post, I assemble an easy procedure to generate fake text data. This technique is useful to us when we fit an NLP classifier and we want to test its strength. If our model isn't able to classify well fake data coming from train, it's appropriate to revisit the training process, adjusting the hyperparameters or directly adding some of these data in train.

. . .

CHECK MY GITHUB REPO

Keep in touch: Linkedin

Machine Learning Data Science NLP Data Augmentation Towards Data Science