Data Augmentation using GPT-2

Masaoud Haidar

Abstract

Classifiers perform badly on imbalanced data. To solve this, we generate synthetic data from the minority class for a text classification task. In this paper, we compare between EDA, a word-based generator, and GPT-2, a context-based generator, when combined with TF-IDF and logistic regression, which is a word-based classifier, and BERT, a context based classifier. We find that GPT-2 performs well with both classifiers, while EDA only performs well with the word-based generator. In addition, we find that both generators result in data that is diverse but that doesn't necessarily belong to the minority class, which could hurt the training of the classifiers. Overall, the GPT-2 generator outperforms EDA on all tested settings, but there is still a large gap behind having real data.

1 Introduction

The problem of imbalanced datasets is common when doing classification. For most binary classification tasks, one of the two classes is very rate compared to the other. For example, if we're predicting a student's admission into a selective university from their essay, then very few of the students who apply will get accepted, and so, most of the data we have will be on students who were rejected. Training a model on the raw imbalanced data can create biased models. Most models end up predicting way more from the majority data than the minority data.

Some common solutions to imbalanced data tasks are majority under-sampling or minority over-sampling. However, when working with text classification tasks, we almost always don't have enough labeled data, so, majority under-sampling would waste some of this expensive labeled data. On the other hand, simple over-sampling from the minority class can result in over-fitting on these examples. A common solution to this is synthetic over-sampling, where we try to create new data points that aren't exact copies of the minority data points but are similar to them. Creating some synthetic data points and adding them to our training data should force our classifiers to predict more from the minority data without over-fitting to specific examples.

When using numerical data, generating synthetic data can be done using SMOTE or other common algorithms. But for text data, generating new data points is harder and is still a somewhat open question. One way to do this is using EDA: Easy Data Augmentation, a combination of random sampling and synonym replacement operations that can be used on an original data point to generate a new synthetic data point that is somewhat similar to the original one. This method has been shown to improve some classifiers, such as CNNs and RNNs, and is particularly useful when we have a small training set [Wei, 2019]. However, these operations can edit the order of the sentence too much, and thus, they can change the context of each word.

Another way to generate synthetic data is through deep learning models. Deep learning models can be trained to predict next words in a sentence, and thus, can be used to build a new sentence one word at a time. Recently, pre-trained models based on the transformer architecture have shown impressive results when used to generate text. Thus, these generators can be used to create synthetic data. [Papanikolaou, 2020] use GPT-2, a transformer based model, to generate synthetic data for imbalanced classification tasks. Their experiments combine this generator with a BERT classifier and show that this method can result in improving the F1 score of the classifier by up to 11 points.

[Kumar, 2020] also use transformer based models for data augmentation. They compare between GPT-2, BERT, and BART for data augmentation when combined with a BERT classifier. [Kumar, 2020] also study the generated data in terms of its diversity and how well it preserves the intended label.

We can see that these text generators have two main themes. The first is using random sampling to edit a data point, and the second is using deep learning models to generate new data points. In this project, we aim to compare between these two themes by picking an example from each one and using it to generate data and then running experiments on the generated data. While some of the previously mentioned papers do make this comparison, they make the comparison under a single type of classifiers. We think that using only one classifier, like BERT, which is a transformer based classifier, would be unfair to sampling based generators like EDA. Thus, we aim to test these generators on different types of classifiers.

2 Background

In this section, we will review the main models used in our project. For this project, we are mainly comparing between two data augmentation methods, GPT-2 and Easy Data Augmentation, when combined with two different classifiers, TF-IDF with logistic regression and BERT. We will provide the background for these models in this section and we will later discuss how we use them for our project.

2.1 Word-based Data Generator: Easy Data Augmentation

Easy Data Augmentation, EDA, is a combination of random sampling methods that can be used to create new data points from original ones [Wei, 2019]. It consists of four main edit operations:

1. Synonym Replacement: Randomly pick a word, and replace it by one of its synonyms.
2. Random Insertion: Randomly pick a word and find a synonym of it, and then randomly insert it somewhere in the text.
3. Random Swap: Randomly pick two data points and swap their positions.
4. Random Deletion: Randomly delete some words.

To generate a new data point, we start with an original data point and perform some random edits to its words to get a new data point. Each of these operations has a parameter α that specifies the probability of applying this operation on a word. It is recommended to increase α if we have a large training dataset.

2.2 Context-based Data Generator: GPT-2

Generative Pre-Trained Transformer 2, GPT-2, is a generative model and is a direct up-scale of GPT. It takes as inputs some initial words in a sentence. Then, for every word in the language, it outputs the probability of that word coming next in the sequence. So, given an initial sequence of tokens $(u_1, u_2, ..., u_{i-1})$, GPT-2 computes, for every possible token U_j in our language:

$$P(u_i = U_j | u_1, u_2, ..., u_{i-1}) = G(u_1, u_2, ..., u_{i-1})_j$$

G here is the function of GPT-2. It consists of an Embedding layer, then a number of Decoder-only Transformer layers, each consisting of a self attention layer followed by a Feed Forward Neural Network, and the outputs of the final layer are projected into the embedding again to get a score for each work in the vocabulary. This is followed by a softmax of these scores to get the probability of each word. Please view the appendix for more details on these different layers and for more details about generating data using GPT-2.

GPT was originally designed for supervised classification tasks using a framework of "Generative Pre-Training and Discriminative Fine-Tuning" [Radford, 2018]. However, it can also be used to generate new text using auto-regression: Starting with some initial tokens, we can predict the possible next tokens and choose a token with a high probability. Then, we add this new token to our input and run the model on the new inputs (the original sequence plus the newly generated token) to predict the next token, and so on until we generate an end token. Note that if we use the raw pre-trained GPT-2, we will generate general data points. Instead, we can fine-tune GPT-2 on our specific dataset by training the model for a few new epochs on our dataset and then use it to generate data points that are similar to the ones in our dataset.

2.3 Word-based Classifier: TF-IDF with logistic regression

This model can be used for text classification tasks. First, we turn the text into numerical representation using the **Term Frequency - Inverse Document Frequency** representation, TF-IDF. Then, we train a numerical classifier like logistic regression to classify our (now numerical) data.

To create the TF-IDF representation of a dataset D: First, we create a one-hot encoding of the words. So, each individual word becomes a feature in our dataset. Then, we replace the ones in the encoding by the TF-IDF value of the word. For a word t in a document d, we calculate:

- The Term Frequency, which is the number of times the word t occurs in the document divided by the size of the document:
$$\text{TF}(t, d) = freq(t, d)$$
- The Inverse Document Frequency of the word with respect to all documents. This is calculated by the log of the number of the documents in the training set divided by the number of documents that have the word t, adjusted by 1 [Scikit-learn]:
$$\text{IDF}(t, D) = \log\left(\frac{|D| + 1}{|d_j \in D : t \in d_j| + 1}\right) + 1$$
- Then, we can calculate the TF-IDF, which is the multiplication of the two:
$$\text{TF-IDF}(D, d, t) = \text{TF}(d, t).\text{IDF}(t, D)$$

After we get the numerical representation, we can run a simple logistic regression for classification:

$$log\left(\frac{P(Y = 1)}{P(Y = 0)}\right) = \beta_0 + \beta_1 X_1 + ... + \beta_N X_N$$

Where $(X_1, ..., X_N)$ are the numerical features found from the text (the TF-IDF values for every word) and the β values are the parameters of the model, estimated from the training set.

2.4 Context-based Classifier: BERT

Bidirectional Encoder Representations from Transformers [Devlin, 2018] is a transformer based model. It is very similar to GPT except for two main differences:

- Each transformer layer is an encoder instead of a decoder. This means that when transforming a token, BERT can attend to all the tokens in the sentence (Where GPT only allows access to previous tokens).
- GPT's last layer calculates the score for each possible next word in the vocabulary. BERT's last layer is an encoding of the text into a vector of numbers, and we can add a fully connect layer on that encoding to perform any specific task, such as adding a layer of one node to perform a classification task.

3 Experimental Setup

In this project, we aim to compare between two Data Augmentation methods, GPT-2 and EDA, when combined with two different classifiers, BERT and TF-IDF with logistic regression. We aim to understand how the different augmentation methods perform with different classifiers.

Notice the following: the EDA generator doesn't give much importance to the order of the words but only cares about what words or synonyms to include in each data point. Similarly, the TF-IDF representation discards any information about the order of the words and only keeps track of the frequence of the words. On the other hand, both GPT2 and Bert use the transformer structure that gives large importance to the order of the words and their interaction with each other (through dot products). This is why we chose to test these generator with these classifiers.

To test this, we used a sentiment analysis dataset where we look at Amazon reviews. Our full dataset includes about 150K data points, with each data point having a review text and a rating of 1-5 stars. Our task is to classify bad reviews. We define those as reviews with 3 stars or less. With this definition, about 20% of our data points are bad reviews. We label those bad reviews with "positive" to match the literature on imbalanced data (so that the minority class is the positive class).

Then, we split our dataset into an 80% train set and 20% test set using a stratified split. Then, from the training set, we create the following datasets:

(A) **Main Training Set**: This has the original 80:20 negative to positive rate.

(B) **Artificially Imbalanced Set**: From (A), we under-sample the minority data points to achieve a 95:5 rate.

(C) **GPT2-Augmented Set:** We fine-tune a GPT-2 model on the positive data points in (B), then use that to generate new examples that we add to (B). We generate enough examples such that we get an 80:20 rate.

For this project, we used "distilGPT2", a compressed version of GPT2.

(D) **Smaller GPT2-Augmented Set:** Exact same as (C), but we only generate enough samples to get a 90:10 rate.

(E) **EDA-Augmented Set:** We use EDA on the positive data points in (B), then use that to generate new examples that we add to (B). Again, we generate enough examples such that we get an 80:20 rate.

For this project, we use $\alpha = 0.1$ as our probability of applying each of the operations on a word. This value is recommended for datasets of the same size as ours [cite EDA].

The idea of our method is that the classifiers trained on (B) are baselines, and the classifiers trained on (A) are golden models. Our goal is to use the dataset from (B) to generate new data points and then train classifiers on those augmented datasets, and then compare them to the ones trained on (A) and (B). This will give us an idea of how much our data augmentation methods can close the gap between imbalanced and balanced datasets.

For the test data, the main test data has an 80:20 rate, and similarly to dataset (B), we under-sample from that to create a second imbalanced test data. The balanced test data gives us a way to compare the classifiers trained on (A), (C) and (E), while the imbalanced test data gives us a more real-world example, as the test data for imbalanced data tasks would also be imbalanced.

Once we have our datasets, we record the following:

1. **Word-based Classifier Performance:** Train a TF-IDF with a logistic regression on each of these training datasets. Record the accuracy and F1 scores on the test data.
2. **Context-based Model Performance:** Train a BERT Classifier on each of these training datasets. Again, record the accuracy and F1 scores on the test data.

   For this project, we use tiny-bert, a small version of BERT that only uses two layers of encoder-transformers and a hidden size of 128. This is the smallest available version of pre-trained BERT. We picked this smallest model so that it is somewhat comparable to the word-based model.
3. **Distribution of the Data:** We train a transformer model on the original data. Then, we run the model on the original data and the EDA and GPT-2 generated data and we extract the outputs of the last hidden layer of the model. Then, we use TSNE to compress these outputs into two dimensions and plot them. The goal of this process is to visualize how similar the generated points are to the original data points. Ideally, we want our generated data points to be close but not exactly equal to the original points, and we want them to span the same areas that the original positive data points span, but not have too much overlap with the negative data. These features would represent an ideal generator that generates diverse data points that still look similar to the original positive ones.

   For this project, we use a base BERT classifier to extract the hidden layers. This is the regular version of BERT that has 12 encoder-transformers and a hidden size of 768. Instead of doing this on all data, which would make it hard to create visualizations, we sample 500 original positive points, 500 original negative points, 500 EDA-augmented points, and 500 GPT2-augmented points and plot only those.
4. **Preserving Labels:** We train a transformer model on the original data (same model as in 3). Then, we record the accuracy of the model on the generated data. The generated data is all positive, so, the accuracy simply captures what rate of the generated data get classified as positive. Some of the data augmentation methods can change the data too much that it doesn't look positive anymore, and so, this method tries to capture that.

4 Results

In this section, we will present the main results we found.

4.1 Word-based Classifier Performance

We trained a classifier of a TF-IDF representation followed by a logistic regression and recorded the accuracy and F1 scores. The results are shown in Table 1. We can see that the full GPT-2 augmentation outperforms the partial GPT-

2 augmentation and the EDA augmentation in both accuracy and F1 scores. This GPT-2 augmentation improves the F1 score by 17.5% but is still 14.1% behind the golden dataset. The EDA augmentation only improves the F1 score by 2.55%.

Dataset	Accuracy	F1 Score
A. Original 80:20	0.8799	0.662
B. Imbalanced 95:5	0.821	0.246
C. GPT2 80:20	0.8366	0.4209
D. GPT2 90:10	0.8326	0.3585
E. EDA 80:20	0.8189	0.2715

Table 1: Results of the Word-Based Model, TF-IDF with logistic regression, when trained on the different datasets. The test dataset used has an 80:20 rate. Check the appendix for the full results.

4.2 Context-based Model Performance

We trained a classifier of a tiny-bert and recorded the accuracy and F1 scores. The results are shown in Table 2. We can see that the full GPT-2 augmentation again outperforms the partial GPT-2 augmentation and the EDA augmentation in both accuracy and F1 scores. However, this GPT-2 augmentation only improves the F1 score by 0.9%. The EDA augmentation results in an extremely bad model that has an F1 score of almost 0.

Dataset	Accuracy	F1 Score
A. Original 80:20	0.8886	0.7002
B. Imbalanced 95:5	0.8443	0.4198
C. GPT2 80:20	0.8403	0.4289
D. GPT2 90:10	0.8327	0.3521
E. EDA 80:20	0.7931	0.0038

Table 2: Results of the Context-Based Model tiny-bert, when trained on the different datasets. The test dataset used has an 80:20 rate. Check the appendix for the full results.

4.3 Distribution of the Data

We train a Base BERT on the original training dataset and record the outputs of the last layer of the model. Then, we make TSNE plot of 500 samples from each group. The resulting TSNE plot is shown in Figure 1 and 2.

We can see that the original positive and negative data points have good separation, except at the top center of the figure where they have a slight overlap. We can also see that the GPT-2 and EDA augmented data points span most of the same areas as the original positive data points, but they have more overlap with the negative data points in the middle of the figure. This tells us that the generated data is diverse, but some of it might not look as positive as the original data points.

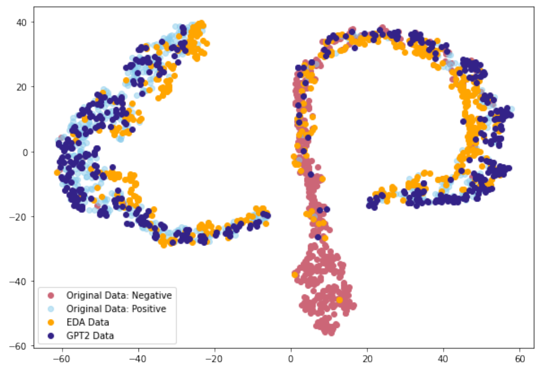
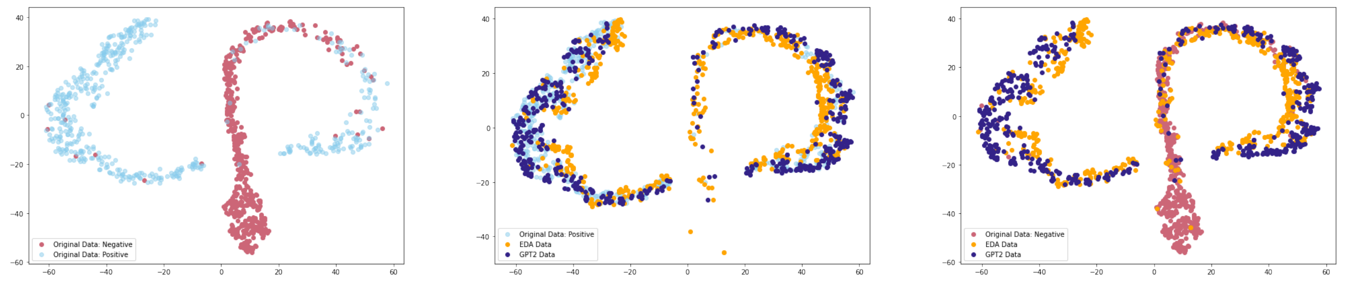


Figure 1: TSNE plots of 500 samples from each of: The original positive data (light blue), the original negative data (red), the GPT2 augmented data (dark blue), and the EDA augmented data (yellow).



(a) Samples from the original set (positive and negative data points)

(b) Samples of the original positive in addition to the GPT and EDA data

(c) Samples of the original negative in addition to the GPT and EDA data

Figure 2: TSNE plots of 500 samples from each type, divided on multiple figures.

4.4 Preserving Labels

If the data augmentation process changes too much in a data point, the new generated data point might not look positive anymore, which could negatively impact the training. Instead of having a human manually look at examples and asses them, we instead train a model to do so.

So, we train a Base BERT on the original training dataset that has the original 80:20 rate and record its accuracy on positive data points. We record this accuracy on the original positive data points and on the EDA and GPT-2 generated datasets (which are also fully positive data points).

The Base Bert Classifier achieves 90.9% accuracy on its own positive training data, but only achieves 76.32% accuracy on the GPT-2 data and only 64.16% accuracy on the EDA data. This means that some of the generated data points don't look as positive as the original data points.

5 Conclusion

We have seen that a context-based classifier performs extremely badly with EDA, but a word-based classifier performs fine with EDA. We've also seen that GPT-2 improvese the word-based classifier by up to 17.5%, but only improves the context-based classifier by 0.9%, which could be because this classifier was already not as affect by the imbalanced data problem. Overall, we have seen that GPT-2 outperforms EDA on all tested settings.

We found that the generated data from both EDA and GPT-2 are diverse. However, these methods sometimes don't preserve the positive labels, meaning that their points are too different from the original data that they look like negative data points. This could explain why both GPT-2 and EDA don't come close to performing as well as having original balanced data.

In addition to the expirements on GPT-2 and EDA, our project offers a new framework for testing generators and models designed on imbalanced by creating artificially imbalanced datasets and comparing results to the original balanced dataset. This framework helped us understand how well our models are doing in addition to understanding how large is the gap left between the new synthetic data and the real data.

6 References

1. J. Devlin, M. Chang, K. Lee, and K. Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". 2018.
2. V. Kumar, A. Choudhary, and E. Cho. "Data Augmentation using Pre-trained Transformer Models". 2020.
3. Y. Papanikolaou and A. Pierleoni. "DARE: Data Augmented Relation Extraction with GPT-2". 2020.
4. P. Platen. "How to generate text: using different decoding methods for language generation with Transformers". Blog on Hugging Face. 2020.
5. A. Radford, J. Wu, R. Child, D Luan, D. Amodei, and I. Sutskever. "Language Models are Unsupervised Multitask Learners". 2019.
6. A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. "Improving Language Understanding by Generative Pre-Training". 2018.
7. Scikit-learn documentation. Available at: https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction
8. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, and I. Polosukhin. "Attention Is All You Need". In Advances in Neural Information Processing Systems. pages 5998–6008. 2017.
9. J. Wei and K. Zou. "EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks". Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). pages 6383–6389. Association for Computational Linguistics. 2019.

Appendix

1 More Background on GPT-2

Here, we review some key concepts used in GPT-2.

1.1 The Self-Attention Mechanism

The Self-Attention mechanism was introduced to transform text features [Vaswani, 2017]. The mechanism takes as an input a sequence of tokens represented by vectors $(a_1, a_2, ..., a_n)$ and using matrix multiplication with three different trained matrices, it computes, for each vector a_i, three resulting vectors: a query vector q_i, a key vector k_i, and a value vector v_i. Then, to transform the token i, the mechanism computes $q_i K^T$ Which are the scores for all the other tokens when transforming the token i. Then the mechanism uses these scores to compute a weighted average of all the value vectors of these tokens. If we group all the vectors for all the tokens into the matrices (Q, K, V), then we can summarise the attention mechanism by:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{|k_i|}}\right)V$$

Where the scaling factor $\frac{1}{\sqrt{|k_i|}}$ is used to avoid making the dot-product too large for long vectors.

1.2 The Decoder-Only Transformer

In a Decoder Transformer, when transforming a term to the next layer, we only compute the weighted average from the previous terms, and we don't allow it to access values from future words. This is done in the $softmax$ step, where we set the scores for future words to be 0 (or $-\infty$ if using a logit $softmax$). This is done to observe the auto-regressive property of the model.

1.3 Auto-Regression Generation: Top-K Sampling

To generate text using GPT-2, we start with some initial tokens. Then, we run the tokens through the model to calculate the probabilities for the next word. Then, we choose the next word and add it to our input. We repeat this step until we generate an ending token, which means that this document is done.

Once we calculate the probabilities, we don't simply pick the word with the most probability. Rather, a better method for generation is to pick the top K words, and then randomly sample one of these words. This turns out to perform better than other methods, like greedy sampling or beam search [Platen, 2020], and it allows us to generate a lot of different examples instead of a deterministic generator.

2 Full Results

Here are the full results of all the models, on all training sets, on all test sets:

Model	Training Dataset	Accuracy on Train	F1 on Train	Accuracy on Full Test 80:20	F1 on Full Test 95:5	Accuracy on Imbalanced Test 80:20	F1 on Imbalanced Test 95:5
TF-IDF + log	A. Original 80:20	0.8929	0.7016	0.8799	0.662	0.9409	0.4891
TF-IDF + log	B. Imbalanced 95:5	0.9575	0.3038	0.821	0.246	0.9545	0.2273
TF-IDF + log	C. GPT2 80:20	0.9446	0.8502	0.8366	0.4209	0.9452	0.3515
TF-IDF + log	D. GPT2 90:10	0.9489	0.6842	0.8326	0.3585	0.952	0.3203
TF-IDF + log	E. EDA 80:20	0.9682	0.9203	0.8189	0.2715	0.9475	0.2264
Tiny Bert	A. Original 80:20	0.9	NA	0.8886	0.7002	0.9399	0.5162
Tiny Bert	B. Imbalanced 95:5	0.9644	NA	0.8443	0.4198	0.9573	0.3955
Tiny Bert	C. GPT2 80:20	0.9587	NA	0.8403	0.4289	0.949	0.3684
Tiny Bert	D. GPT2 90:10	NA	NA	0.8327	0.3521	0.9538	0.332
Tiny Bert	E. EDA 80:20	0.999	NA	0.7931	0.0038	0.949	0.0046
Base Bert	A. Original 80:20	0.981	NA	0.9193	0.7865	0.9592	0.6433

Table 1: Full results of all models

Note that there are some "NA"s in the table. This is because for Bert models, the batches in the training dataset are shuffled every time we iterate over them. This improves batch training but makes it hard to measure performance after the training is done. We aren't interested in results on training data anyway so we didn't try to overcome this problem.