DNSC 6290 Natural Language Processing

EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks

George Washington University

Shengqi Zhou

G44650601

Table of Contents

1. Overview

The paper (https://paperswithcode.com/paper/eda-easy-data-augmentation-techniques-for) is about using EDA (Easy Data Augmentation) to boost performance on text classification tasks. EDA consists of four operations: Synonym Replacement (SR): replace certain words with their synonyms, Random Insertion (RI): randomly insert synonyms of certain words into a sentence, Random Swap (RS): swap positions of two words randomly, Random Deletion (RD): randomly remove each word in a sentence with a certain probability. An example of those operations is shown in figure 1 in appendix. There are two main experiments in this study. First, test EDA on five different datasets with different size of training sets. Second, test four operations of EDA on five different text classification tasks with different amounts of words changed.

2. Modeling Methods

The modeling techniques that are used to test the performance of EDA is CNN (Convolutional Neural Networks) and RNN (Recurrent Neural Networks). The packages that are used for modeling are Keras and Sklearn. For CNN, as we can see from figure 2 in appendix, dense layer size of 20 with relu activation is used. 128 filters of size 5 1D convolutional layer is used. Categorical crossentropy loss function and adam optimizer is used, also monitor the accuracy during the training by passing accuracy to the metrics argument. And global 1D max pool layer is used. Also, from figure 4 in appendix, we see that when applying CNN, we used early stopping with a patience of 3 epochs. For RNN, as we can see from figure 3 in appendix. Categorical crossentropy loss function and adam optimizer is used, also monitor the accuracy during the training by passing accuracy to the metrics argument. Dense layer size of 20 with relu activation is used. Bidirectional layer with 64 LSTM cells and a drop out of 0.5 and another bidirectional layer with 32 LSTM cells and a drop out of 0.5. Also, from figure 5 in appendix, we see that when applying RNN, we used early stopping with a patience of 3 epochs.

3. Experiment Methods

First, five datasets that are used to perform experiments on are: (1) SST-2: Stanford Senti- ment Treebank, (2) CR: customer reviews, (3) SUBJ: subjectivity/objectivity dataset, (4) TREC: question type dataset, (5) PC: Pro-Con dataset. The hypothesis for this experiment is that EDA works better on smaller datasets. Then two experiments were run on CNN and RNN models. The results are shown on figure 6 in appendix. And we can see that the average improvement was around 0.9% for whole datasets, 3.9% when Ntrain = 500, 0.9% when Ntrain = 2000, 1% when Ntrain = 5000.

Second, comparison was made between models with and without EDA under different fractions of training dataset, from figure 7 we can see that the highest average accuracy without EDA was 88.3% and it was achieved by using 100% of the training data. Models trained by using EDA achieved better accuracy than this number, which is 88.6% with only 50% of the available training sets.

4. EDA Decomposed

In this section, the effects of each operation in EDA are explored. Because Synonym Replacement has already been studied and it is obvious to hypothesize that the majority of EDA's performance gain was from SR. then the four operations in EDA was applied over five text classification tasks, as shown in figure 8. The alpha parameter is the percent of words that are changed for augmentation. We realized that all four operations in EDA performed pretty well. For Synonym Replacement, the improvement of performance was better when the alpha value is small, and the performance became worse when the alpha value is high. For Random Insertion, the improvement performance was pretty stable across different alpha values. For Random Swap, the improvement performance was better when alpha is smaller than 0.2, and the performance started to become worse when alpha is larger than 0.3. for Random Deletion, the performance is high for low alpha value, but the performance became worse for high alpha value. And alpha = 0.1 seems to be a decent value for EDA performance.

5. Issues

5.1 Does EDA conserve true labels

In data augmentation procedures, the class labels of the datasets are kept while the data is altered. Is this section, RNN was trained on PC dataset for both with and without EDA scenarios. The result is shown on figure 9 in appendix. As we can see from the figure, we notice that for most part, sentences performed EDA were conserved the class labels of the sentences before EDA.

5.2 How much augmentation

In this section, how much number of generated augmented sentences per original sentences, which is $n_{aug}$was Determined. In figure 10 in the appendix, the average performances of all datasets for various n_{aug} is shown. The conclusion for this part is that: For one, generating a large number of augmented sentences yielded large performance boosts for smaller training sets. For another, generating more than four augmented sentences per original sentences was unhelpful. According to the performance, the recommended number of n_{aug} is shown in figure 11 in the appendix.

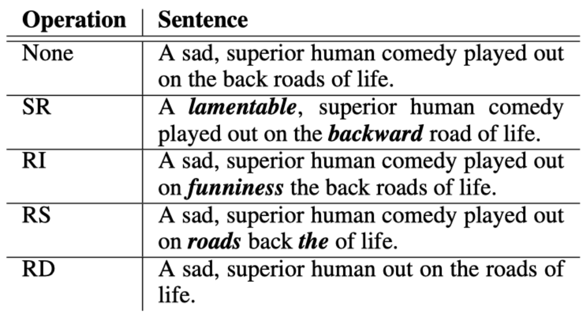
6. Comparison with related work

There are other methods about data augmentation, and they are pretty creative but complex. Because most studies use data augmentation as a complementary result for translation or in a task specific context, it is difficult to compare studies directly. But there are two similar one. Hu (2017) mentioned a model that generates fake data, which demonstrates 3% gain in accuracy on two datasets. Kobayashi (2018) showed by replacing words with other words that were predicted from the sentence using a bidirectional language model, which yielded 0.5% gain on all five datasets. However, building and training a biLSTM model requires a lot of hard work. EDA is much easier to use because it does not require training language models and does not need other external datasets. In figure 12 in appendix, we can see that EDA is much easier to use when compared with other EDA methods. This is also an indication that EDA is easy to implement than other methods.

7. Major contribution

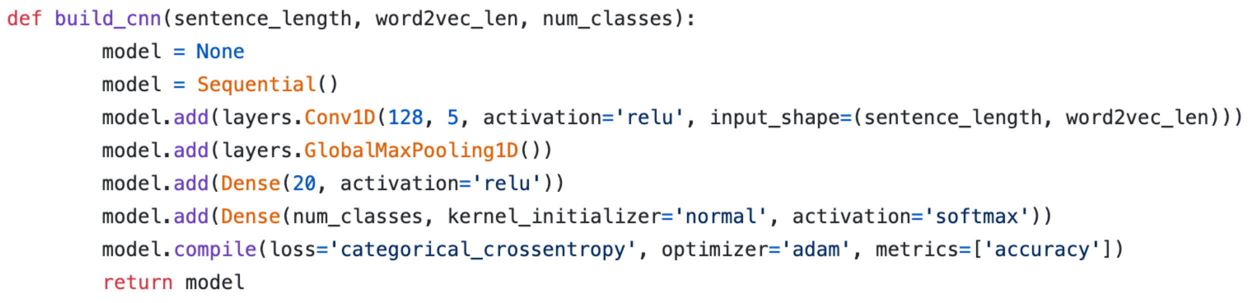
The final contributions are that EDA has positive improvement on boosting performance and reducing overfitting for training on smaller datasets. Also, Performing EDA by using half of the whole training dataset, the research got the same accuracy with all available data. The most importance part is that EDA is easy to implement than other complex methods. A chart of result is shown in figure 6 in appendix.

8. Appendix



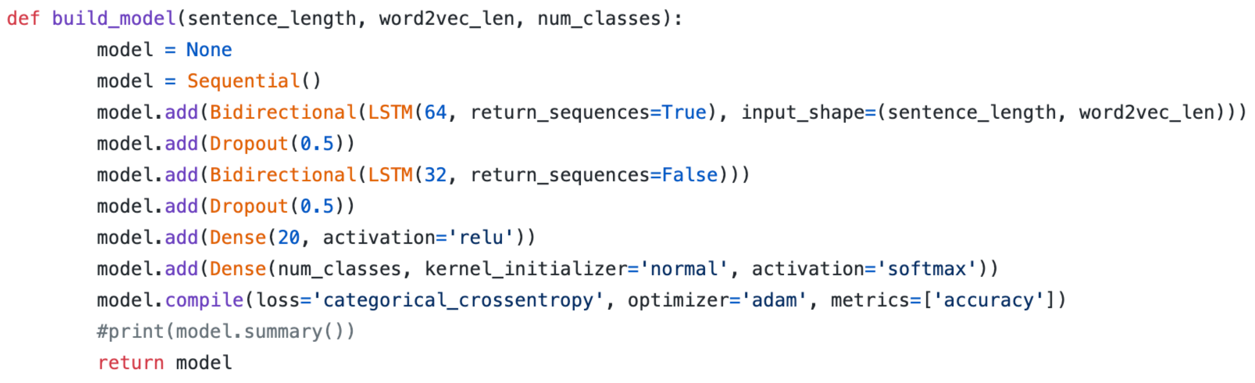
Operation	Sentence
None	A sad, superior human comedy played out on the back roads of life.
SR	A ***lamentable***, superior human comedy played out on the ***backward*** road of life.
RI	A sad, superior human comedy played out on ***funniness*** the back roads of life.
RS	A sad, superior human comedy played out on ***roads*** back ***the*** of life.
RD	A sad, superior human out on the roads of life.

Figure1: Examples of sentences generated by performing four operations of EDA



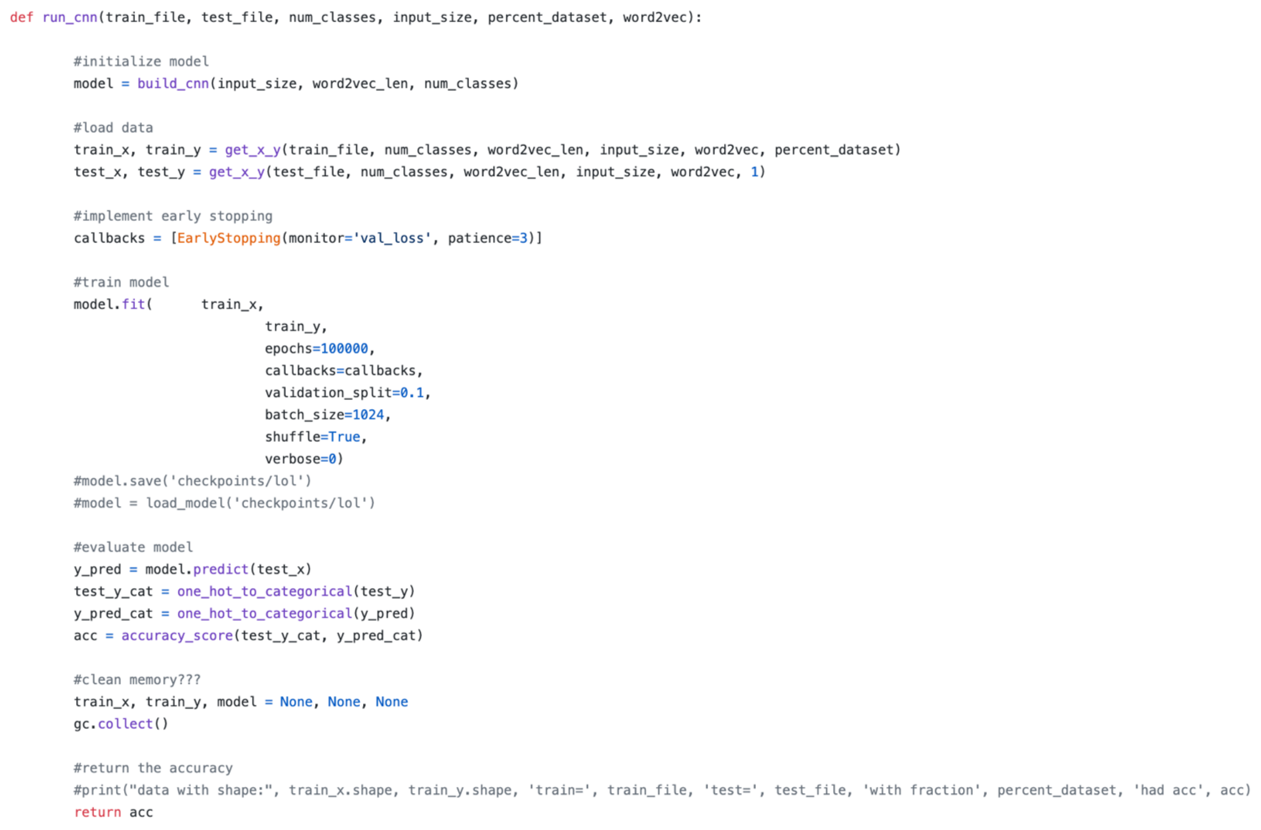
```
def build_cnn(sentence_length, word2vec_len, num_classes):
        model = None
        model = Sequential()
        model.add(layers.Conv1D(128, 5, activation='relu', input_shape=(sentence_length, word2vec_len)))
        model.add(layers.GlobalMaxPooling1D())
        model.add(Dense(20, activation='relu'))
        model.add(Dense(num_classes, kernel_initializer='normal', activation='softmax'))
        model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
        return model
```

Figure2: code for building CNN



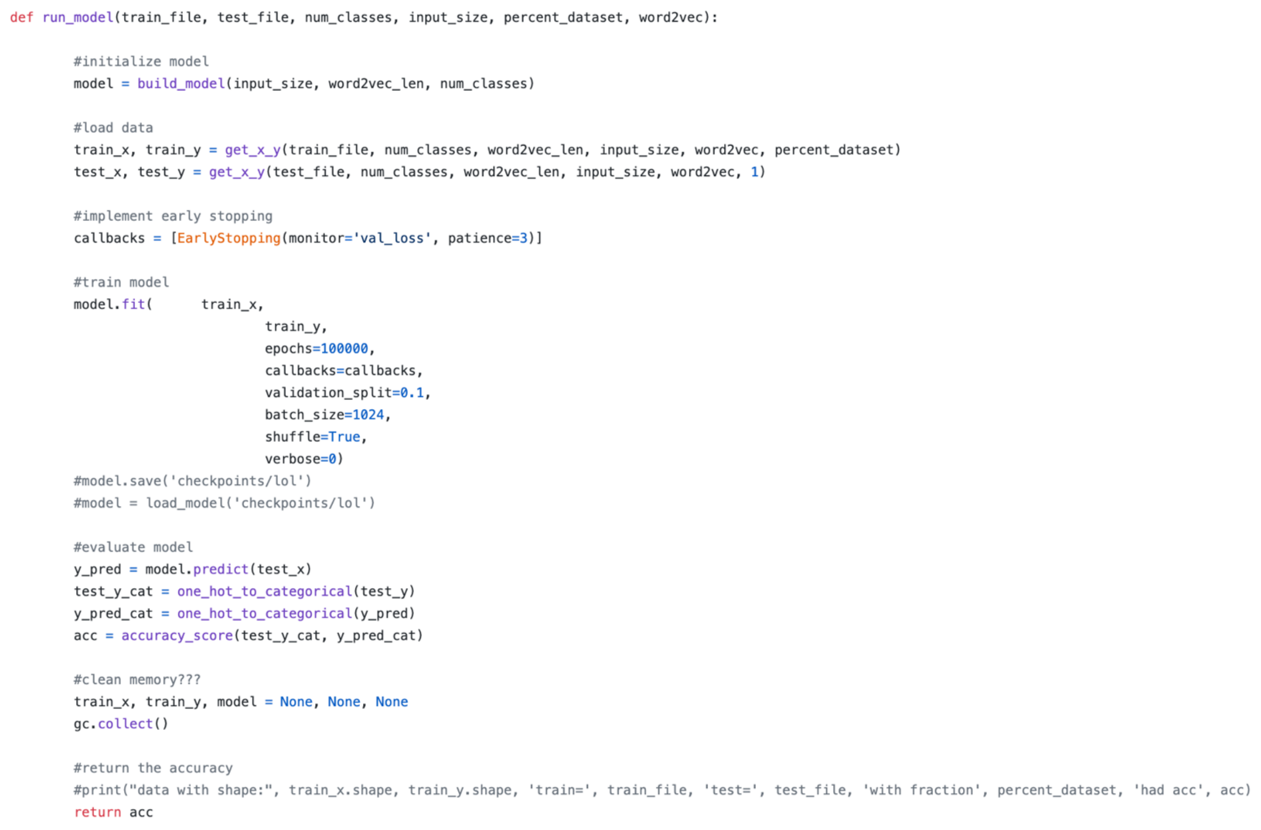
```
def build_model(sentence_length, word2vec_len, num_classes):
        model = None
        model = Sequential()
        model.add(Bidirectional(LSTM(64, return_sequences=True), input_shape=(sentence_length, word2vec_len)))
        model.add(Dropout(0.5))
        model.add(Bidirectional(LSTM(32, return_sequences=False)))
        model.add(Dropout(0.5))
        model.add(Dense(20, activation='relu'))
        model.add(Dense(num_classes, kernel_initializer='normal', activation='softmax'))
        model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
        #print(model.summary())
        return model
```

Figure3: code for building RNN



```python
def run_cnn(train_file, test_file, num_classes, input_size, percent_dataset, word2vec):

        #initialize model
        model = build_cnn(input_size, word2vec_len, num_classes)

        #load data
        train_x, train_y = get_x_y(train_file, num_classes, word2vec_len, input_size, word2vec, percent_dataset)
        test_x, test_y = get_x_y(test_file, num_classes, word2vec_len, input_size, word2vec, 1)

        #implement early stopping
        callbacks = [EarlyStopping(monitor='val_loss', patience=3)]

        #train model
        model.fit(      train_x,
                                train_y,
                                epochs=100000,
                                callbacks=callbacks,
                                validation_split=0.1,
                                batch_size=1024,
                                shuffle=True,
                                verbose=0)
        #model.save('checkpoints/lol')
        #model = load_model('checkpoints/lol')

        #evaluate model
        y_pred = model.predict(test_x)
        test_y_cat = one_hot_to_categorical(test_y)
        y_pred_cat = one_hot_to_categorical(y_pred)
        acc = accuracy_score(test_y_cat, y_pred_cat)

        #clean memory???
        train_x, train_y, model = None, None, None
        gc.collect()

        #return the accuracy
        #print("data with shape:", train_x.shape, train_y.shape, 'train=', train_file, 'test=', test_file, 'with fraction', percent_dataset, 'had acc', acc)
        return acc
```

Figure4: code for applying CNN for data augmentation



```python
def run_model(train_file, test_file, num_classes, input_size, percent_dataset, word2vec):

        #initialize model
        model = build_model(input_size, word2vec_len, num_classes)

        #load data
        train_x, train_y = get_x_y(train_file, num_classes, word2vec_len, input_size, word2vec, percent_dataset)
        test_x, test_y = get_x_y(test_file, num_classes, word2vec_len, input_size, word2vec, 1)

        #implement early stopping
        callbacks = [EarlyStopping(monitor='val_loss', patience=3)]

        #train model
        model.fit(      train_x,
                                train_y,
                                epochs=100000,
                                callbacks=callbacks,
                                validation_split=0.1,
                                batch_size=1024,
                                shuffle=True,
                                verbose=0)
        #model.save('checkpoints/lol')
        #model = load_model('checkpoints/lol')

        #evaluate model
        y_pred = model.predict(test_x)
        test_y_cat = one_hot_to_categorical(test_y)
        y_pred_cat = one_hot_to_categorical(y_pred)
        acc = accuracy_score(test_y_cat, y_pred_cat)

        #clean memory???
        train_x, train_y, model = None, None, None
        gc.collect()

        #return the accuracy
        #print("data with shape:", train_x.shape, train_y.shape, 'train=', train_file, 'test=', test_file, 'with fraction', percent_dataset, 'had acc', acc)
        return acc
```

Figure5: code for applying RNN for data augmentation

	Training Set Size			
Model	500	2,000	5,000	full set
RNN	75.3	83.7	86.1	87.4
+EDA	79.1	84.4	87.3	88.3
CNN	78.6	85.6	87.7	88.3
+EDA	80.7	86.4	88.3	88.8
Average	76.9	84.6	86.9	87.8
+EDA	79.9	85.4	87.8	**88.6**

Figure6: Average performances (%) of five tasks for RNN and CNN models with and without EDA on different training sizes

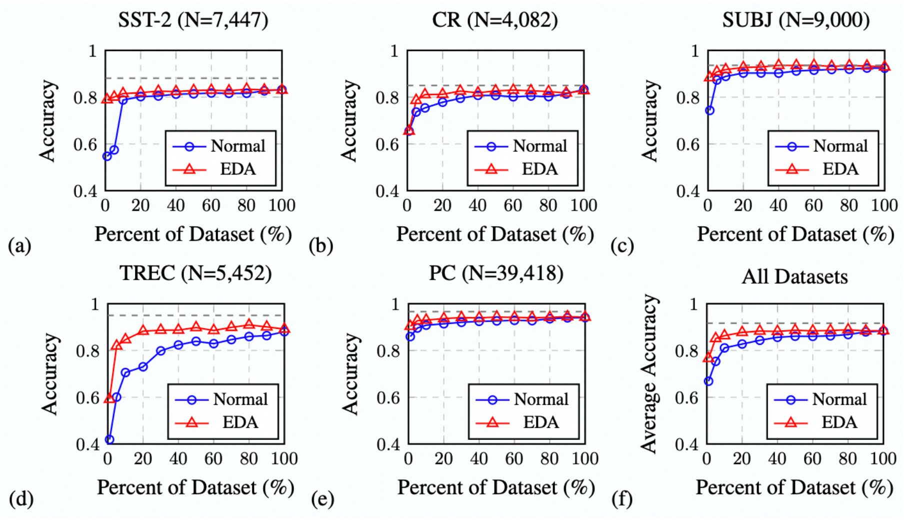


Figure7: Performance of text classification tasks with and without EDA under different training sets size

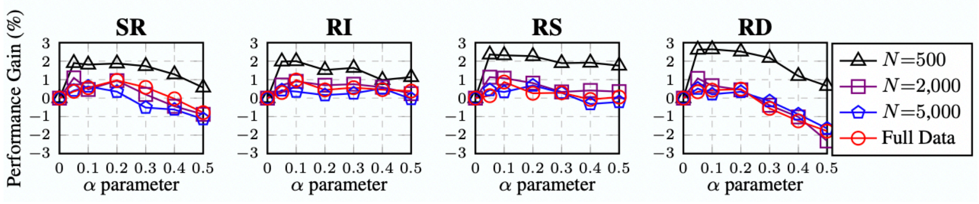


Figure8: Performance gain of performing EDA over five text classification tasks under different training set

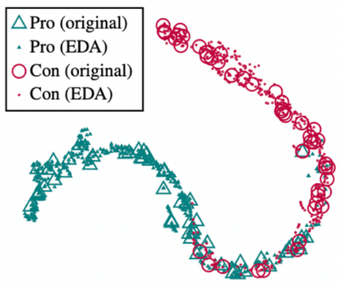


Figure9: Latent space visualization of original and augmented sentences in the Pro-Con dataset.

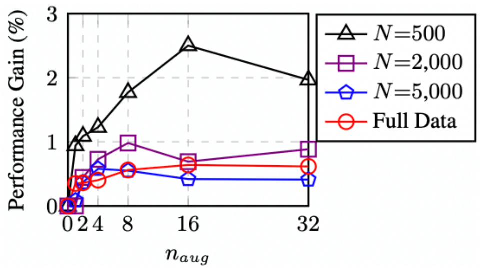


Figure10: Performance gain of EDA across five text classification tasks for various training set sizes.

N_{train}	α	n_{aug}
500	0.05	16
2,000	0.05	8
5,000	0.1	4
More	0.1	4

Figure11: Recommended usage parameters

Technique (#datasets)	LM	Ex Dat
Trans. data aug.[1] (1)	yes	yes
Back-translation[2] (1)	yes	yes
VAE + discrim.[3] (2)	yes	yes
Noising[4] (1)	yes	no
Back-translation[5] (2)	yes	no
LM + SR[6] (2)	yes	no
Contextual aug.[7] (5)	yes	no
SR - kNN[8] (1)	no	no
EDA (5)	**no**	**no**

Figure12: related work in data augmentation