Technology Review

Rohit Bansal (rbansal3)
December 12, 2018

Hash Vectorizer

Hash vectorizer as text analysis tool and its advantage over traditional TF-IDF vectorizer

Contents

Summary

Hash vectorizers are very effective tool for sentimental analysis in the scenarios where complete vocabulary of the document corpus is not available and the document is available for very limited period. In such a scenario, TF-IDF vectorizers are not very effective as these vectorizers require complete vocabulary in order to represent a document in terms of a TF-IDF vector. On the other hand, Hash vectorizers does not required complete vocabulary of document corpus as they represent each document on a fixed dimensional space using hash function for the terms present in the document. Due to this unique advantage of Hash vectorizers, they are very effective in performing sentiment analysis on social media content. These vectorizers are one of the most potent tools, that industry leaders use in order to tackle the menace called “fake news”!

As a member of team implementing SVM for text analysis, I have implemented both TF-IDF and Hash vectorizers in order to analyze the performance difference. In this paper, first, TF-IDF vectorizers and its limitations are discussed. Then hash vectorizers are introduced and their unique advantage is discussed. In the end, the result of sentimental analysis performed using hash vectorizer and TF-IDF vectorizer from our project is documented.

TF-IDF Vectorizers

Introduction

TF-IDF vectorizers are often used to create word vectors from set of documents. This vectorizer scales the term frequency (word count) by penalizing the terms that appear widely across the corpus. Thus, with TF-IDF vectorizer, each term appeared in a document is represented as a dimension impacting the document label. The TF-IDF vectorizers are one of the most popular and most commonly used technique in text analysis. For sentimental analysis, the vectors are often fed to a Naïve Bayes classification or Support Vector Machine classification in order to build a predictive model.

TF-IDF for a term ‘t’ in document ‘d’ in a collection of document ‘D’ is defined as:

$$TFIDF(t,d,D)=TF(t,d)\cdot IDF(t,D)$$

Each document d can be represented by a vector ‘d’ which will be a linear vector having TF-IDF values for each term in the document corpus. Thus

$$\mathrm{Vec(d)} = [\mathrm{TFIDF}(t_1,\mathrm{d, D}), \mathrm{TFIDF}(t_2,\mathrm{d, D}), \mathrm{TFIDF}(t_3,\mathrm{d, D}),\ldots\ldots\ldots \mathrm{TFIDF}(t_n,\mathrm{d, D})]$$

where n is the number of terms in the corpus.

Limitations

Though TF-IDF works effectively when complete document corpus vocabulary is available, it has some limitations:

- **Document Corpus Vocabulary**

In TF-IDF vectorizer, each term in corpus represents a dimension on which a document is represented. Thus TF-IDF vectorizers require complete corpus vocabulary in advance. This is not possible in a lot of practical cases as more and more data is being generated every day. Training model with limited vocabulary or vocabulary from different domain often leads to misleading result. For example, fake news being generated on social media. Since fake news can be about anything and any subject, the domain of the content is not defined. As a result, defining complete vocabulary for this case is not possible.

- **Data Retention Duration**

In TF-IDF vectorizer, each document must be represented in a multi-dimensional space where each dimension is represented by a term in corpus. If the analysis in being done, on the data which is available for limited time then the vocabulary changes significantly over the period of time due to purging of the old documents. In such a scenario, since new data is available for training while old data is getting purged, one must recalculate all the TF-IDF vectors based on new vocabulary. This makes TF-IDF vectors an expensive affair. For example, data generated by personal profiles on social media account can be retained only for limited period. Thus, to perform text analysis on data generated by social media account, TF-IDF vectors have to be re-calculated over the period of time making them infeasible to use for these scenarios.

- **Computation Time**

If the vocabulary size is very big, then TF-IDF vectorizer tend to become very expensive in terms of computation time. Often, dimension reduction is performed by applying different techniques to select most relevant terms.

The above constraints are very restrictive when it comes to sentimental analysis for text on social media content. With social media content, it is often the case that corpus vocabulary is changes very rapidly due to new content getting generated frequently while old content getting purged at the same time. For this reason, TF-IDF vectorizers are not very effective in these scenarios and an alternative is required.

Hash Vectorizers

Introduction

Hash vectorizers allow users to do continuous training as more training data becomes available, without specifying the vocabulary in advance. This feature along with appropriate SGD parameters enables training of some foreign languages across a time period longer than retention duration of the data.

Hash vectorizer uses a “hashing trick” to build a vector of pre-defined length. It applies a hash function to the terms (words) and uses the hash values as the dimension. As a result, a document is represented in a fixed length vector space (generally taken as 1048576). With hash vectorizer, a document ‘d’ will be represented as:

$$\mathrm{Vec(d)} = [h(t_1), h(t_2,), h(t_3),\ldots\ldots\ldots h(t_n)]$$

where n is the predefined number for a hash vector and h is the hash function

Advantages

- **Document Corpus Vocabulary**

Since each document is represented by a fixed length vector and the dimensions used to represent vector are the hash value of the terms present in the document, Hash vectors does not require the complete corpus vocabulary. This is a big advantage in scenarios where complete corpus vocabulary is not available

- **Data Retention Duration**

Since each document is represented by the hash function value of the terms in a fixed length vector space, Hash vector is immune to the purging of old documents. This makes it an effective tool for the scenarios where data can be retained only for short durations. Ex: social media content

- **Computation time**

Since it has fixed length (generally 1048576) thus it is faster to compute hash vector than to compute TF-IDF when the vocabulary size is huge (more than 1048576). The vocabulary size for social media content is generally far larger than 1048576 - making hash vectorizers very effective there.

The above advantages make Hash vectorizers a very effective tool for sentimental analysis on text data generated on social media. It can handle large and ever-changing data more effectively than compared to TF-IDF vectorizers.

Hash vs TF-IDF Vectorizer with SVM Classifier

Supported Vector Machine (SVM) has been an extremely popular supervised learning algorithm to recognizing patterns for classification and regression analysis. It is often used for creating text classifier like sentimental classification, for its effectiveness in high dimensional spaces especially in cases where the number of dimensions is greater than that of samples. There are few to no SVM text classifier tool kit that provides user the flexibility to explore different natural language processing (NLP) techniques for pre-processing input text to construct a dictionary with text features/hyperparameters.

As part of my project for this course, I developed a framework that enables users to use SVM in integration with different NLP techniques. Using this framework, I compared hash vectorizers performance against TF-IDF vectorizer using scikit learn library.

I analyzed it with two test scenarios:

Test Scenario 1: Sentimental Analysis on Seen Data

In this scenario, I used tweets data generated for airlines for sentimental analysis. In the given scenario, the complete document corpus vocabulary was available. Thus, in this scenario TF-IDF vectorizer should perform better than hash vectorizer. However, the gap in performance should be minimal.

Here are the findings from the test runs:

- **Cross validation score for SVM classifier with Hash Vectorizer**

I used 5 splits on the training data and computed the cross-validation score for Hash vectorizer. Here are the scores

```
Cross Validated Scores:
        [0.74338684 0.7445173  0.74813475 0.74496948 0.73163012]
Mean score:     74.25%
Score Variance: 0.00%

Test Accuracy Score:    74.25%
Train Accuracy Score:   90.28%
Test Precision Score:   4.59%
Train Precision Score:  2.41%
Test Recall Score:      4.60%
Train Recall Score:     3.06%

C:\Users\rbansal\Desktop\Learning\Git\Python\Remote\CS410Project>
```

- **Cross validation score for SVM classifier with TF-IDF Vectorizer**

I used 5 splits on the training data and computed the cross-validation score for TF-IDF vectorizer. Here are the scores:

```
Cross Validated Scores:
       [0.74542166 0.74790866 0.7494913  0.7470043  0.73479539]
Mean score:     74.49%
Score Variance: 0.00%

Test Accuracy Score:    74.49%
Train Accuracy Score:   93.43%
Test Precision Score:   4.67%
Train Precision Score:  2.48%
Test Recall Score:      4.60%
Train Recall Score:     3.19%
```

By analyzing above results, it can be identified that TF-IDF has mean accuracy score of 74.49% while hash vectorize have mean score of 74.25%. This indicates that in the scenario where total corpus vocabulary is available TF-IDF performs slightly better but there is no significant difference.

Test Scenario 2: Sentimental Analysis on Unseen Data

In this scenario, I used tweets data generated for airlines for training the model. I used the trained model for sentimental analysis on the tweets data generated for a stock market scrip (NVIDIA). In the given scenario, stock market data is unseen data thus the complete document corpus vocabulary was not available with the trained model. Thus, in this scenario hash vectorizer should perform better than TF-IDF vectorizer. Here are the findings from the test runs:

- **Sentimental classification by SVM classifier with Hash Vectorizer**

I used tweets data for NVIDIA stock and tried to get the sentiments about the stock from tweets using Hash vectorizer. Here is the output from the utility we built:

```
C:\Users\rbansal\Desktop\Learning\Git\Python\Remote\CS410Project>python3 svm.py
Total Messages Processed:       5641
Positive Message Percentage:    2.1981918099627724 %
Negative Message Percentage:    94.06133664243929 %
Neutral Message Percentage:     3.7404715475979438 %
```

- **Sentimental classification by SVM classifier with TF-IDF Vectorizer**

I used tweets data for NVIDIA stock and tried to get the sentiments about the stock from tweets using TF-IDF vectorizer. Here is the output from the utility we built:

```
C:\Users\rbansal\Desktop\Learning\Git\Python\Remote\CS410Project>python3 svm.py
Total Messages Processed:       5641
Positive Message Percentage:    2.995922708739585 %
Negative Message Percentage:    88.56585711753235 %
Neutral Message Percentage:     8.438220173728062 %
```

Note, NVIDIA is a stock which lost almost 50% valuation in last three months.

As a result, most of the tweets for NVIDIA were supposed to be negative. We identified that round about 95% tweets were in fact negative. In the above test case, hash vectorizer was able to tag 94% of the data as negative while TF-IDF was able to tag only 88% of the data as negative. So, it is evident that Hash Vectorizer performed much better than TF-IDF in this scenario.

By analyzing the above two scenarios and the associated results, we can conclude that Hash vectorizer performs much better in case of unseen data i.e. when the complete vocabulary is not available. On the other hand, when complete corpus vocabulary is available, then there is no significant drop in the performance of the Hash vectorizer model when compared to TF-IDF vectorizer model. This makes Hash vectorize a very useful tool when it comes to sentimental analysis on text data available on social media.

Access

I have made this paper available for download from my github page. This paper and my further work in order to identify and tag fake news can be found here. Also, the SVM classifier utility that I developed in order to analyze all these vectorizers can be found here.