Home

A Complete Guide on Feature Extraction Techniques

Shankar297 — Published On May 31, 2022 and Last Modified On June 3rd, 2022

Intermediate Python Technique

This article was published as a part of the Data Science Blogathon.

Introduction on Feature Extraction

In Natural Language Processing, Feature Extraction is one of the most important steps to be followed for a better understanding of the context of what we are dealing with. After the initial text is cleaned, we need to transform it into its features to be used for modeling. Document data is not computable so it must be transformed into numerical data such as a vector space model. This transformation task is generally called feature extraction of document data. Feature Extraction is also called Text Representation, Text Extraction, or Text Vectorization.

In this article, we will explore different types of Feature Extraction Techniques like Bag of words, Tf-Idf, n-gram, word2vec, etc. Without wasting our time let's start our article.

First, let us understand the answer to some questions:

1. What is Feature Extraction from the text?
2. Why do we need it?
3. Why is it difficult?
4. What are the techniques?

1. What is Feature Extraction from the text?

If we have textual data, that data we can not feed to any machine learning algorithm because the Machine Learning algorithm doesn't understand text data.
It understands only numerical data. The process of converting text data into numbers is called Feature Extraction from the text.
It is also called text vectorization.

2. Why do we Need it?

So we know that machines can only understand numbers and to make machines able to identify language we need to convert it into numeric form.

3. Why is it Difficult?

If we ask any NLP practitioner or data scientist then the answer will be yes, somewhat it is difficult.
Now let us compare text feature extraction with feature extraction in other types of data.
So in an image dataset, image feature extraction is easy because images are already present in form of numbers(Pixels).
If we talk about audio data, suppose emotion prediction from speech recognition so, in this, we have data in form of waveform signals where features can be extracted over some time Interval.
But when we have a sentence and we want to predict its sentiment, How will you represent it in numbers? In this article, we are going to study these techniques.

4. What are the Techniques?

1. One Hot Encoding
2. Bag of Word (BOW)
3. n-grams
4. Tf-Idf
5. Custom features
6. Word2Vec(Word Embedding)

Common Terms Used

Corpus(c) — The total number of words present in the whole dataset is known as Corpus.

Vocabulary (V) — Total number of unique words available in the corpus.

Document (D) — There are multiple records in a dataset so a single record or review is referred to as a document.

Word (w) — Words that are used in a document are known as Word.

Techniques for Feature Extraction

1. One Hot Encoding

One hot encoding means converting the words of your document into a V-dimension vector.

This technique is very intuitive means it is simple and you can code it yourself. This is only the advantage of One-Hot Encoding.

Let's say we have documents "We are learning Natural Language Processing", "We are learning Data Science", and "Natural Language Processing comes under Data Science".

corpus – We are learning Natural Language Processing, We are learning Data Science, Natural Language Processing comes under Data Science

Vocabulary(Unique words) – We are learning Natural Language Processing Data Science comes under – V – 10

Document1 — We are learning Natural Language Processing

	0	1	2	3	4	5	6	7	8	9
We	1	0	0	0	0	0	0	0	0	0
are	0	1	0	0	0	0	0	0	0	0
learning	0	0	1	0	0	0	0	0	0	0
Natural	0	0	0	1	0	0	0	0	0	0
Language	0	0	0	0	1	0	0	0	0	0
Processing	0	0	0	0	0	1	0	0	0	0

Advantage

1. It is Intuitive
2. Easy to implement

One hot encoding is not used in the industry because it has flaws.

Disadvantage

1. It creates Sparsity.
2. Size of each document after one hot encoding may be different. The machine learning model doesn't work.
3. Out of Vocabulary (OOV) problem. At the time of prediction new word come which is not available in the vocabulary.
4. No capturing of semantic meaning.

2. Bag of Words

It is one of the most used text vectorization techniques. A bag-of-words is a representation of text that describes the occurrence of words within a document.

Specially used in the Text Classification task.

We can directly use CountVectorizer class by Scikit-learn.

code example

```
df
```

	text	output
0	We are learning Natural Language Processing.	1
1	We are learning Data Science.	1
2	Natural Language Processing comes under Data S...	0

```
cv = CountVectorizer()
```

```
bow = cv.fit_transform(df['text'])
```

```
#Vocabulary
print(sorted(cv.vocabulary_))
```

```
['are', 'comes', 'data', 'language', 'learning', 'natural', 'processing', 'science', 'under', 'we']
```

```
print(bow[0].toarray())
```

```
[[1 0 0 1 1 1 1 0 0 1]]
```

```
bow.toarray()
```

```
array([[1, 0, 0, 1, 1, 1, 1, 0, 0, 1],
       [1, 0, 1, 0, 1, 0, 0, 1, 0, 1],
       [0, 1, 1, 1, 0, 1, 1, 1, 1, 0]], dtype=int64)
```

Advantage

1. Simple and intuitive.
2. Size of each document after BOW same.
3. Out of Vocabulary (OOV) problem does not occur, which means the model does not give an error.

Disadvantage

1. BOW also creates Sparsity.
2. OOV, Ignoring the new word.
3. Not consider sentence ordering issues.

3. Bag of n-grams

A bag-of-n -grams model represents a text document as an unordered collection of its n-grams.

bi-gram — using two words of the document

tri-gram — using three words of the document

n-gram — using n number of words of the document

code example

```
df
```

	text	output
0	We are learning Natural Language Processing.	1
1	We are learning Data Science.	1
2	Natural Language Processing comes under Data S...	0

```
cv = CountVectorizer(ngram_range=(1,2))
```

```
bow = cv.fit_transform(df['text'])
```

```
#Vocabulary
print(cv.vocabulary_)
```

```
{'we': 18, 'are': 0, 'learning': 8, 'natural': 11, 'language': 6, 'processing': 13, 'we are': 19, 'ar
e learning': 1, 'learning natural': 10, 'natural language': 12, 'language processing': 7, 'data': 4,
'science': 15, 'learning data': 9, 'data science': 5, 'comes': 2, 'under': 16, 'processing comes': 1
4, 'comes under': 3, 'under data': 17}
```

```
print(bow[0].toarray())
```

```
[[1 1 0 0 0 0 1 1 1 0 1 1 1 1 0 0 0 0 1 1]]
```

```
bow.toarray()
```

```
array([[1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1],
       [1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1],
       [0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0]],
      dtype=int64)
```

Advantage

1. Simple and easy to implement.

2. Able to capture the semantic meaning of the sentence.

Disadvantage

1. As we move from unigram to N-Gram then the dimension of vector formation increases and slows down the algorithm.

2. OOV, Ignoring the new word.

4. Tf-Idf — Term Frequency and Inverse Document Frequency

TF-IDF (term frequency-inverse document frequency) is a statistical measure that evaluates how relevant a word is to a document in a collection of documents.

- Term Frequency (TF):
  The number of times a word appears in a document is divided by the total number of words in that document. 0 < Tf< 1

$$tf_{ij} = \frac{\text{Number of times term i appears in the document}}{\text{Total Number of terms in the document j}}$$

- Inverse Document Frequency (IDF):
  The logarithm of the number of documents in the corpus is divided by the number of documents where the specific term appears. In Scikit-learn use log(N/ni) + 1 formula.

$$idf_i = \log\left(\frac{\text{Total number of documents}}{\text{Number of documents with term i in it}}\right)$$

code example

```
df
```

	text	output
0	We are learning Natural Language Processing.	1
1	We are learning Data Science.	1
2	Natural Language Processing comes under Data S...	0

```
tfidf = TfidfVectorizer()
```

```
tfidf_vectors = tfidf.fit_transform(df['text'])
```

```
#Vocabulary
print(sorted(tfidf.vocabulary_))
```

```
['are', 'comes', 'data', 'language', 'learning', 'natural', 'processing', 'science', 'under', 'we']
```

```
print(tfidf_vectors[0].toarray())
```

```
[[0.40824829 0.         0.         0.40824829 0.40824829 0.40824829
  0.40824829 0.         0.         0.40824829]]
```

```
tfidf_vectors.toarray()
```

```
array([[0.40824829, 0.        , 0.        , 0.40824829, 0.40824829,
        0.40824829, 0.40824829, 0.        , 0.        , 0.40824829],
       [0.4472136 , 0.        , 0.4472136 , 0.        , 0.4472136 ,
        0.        , 0.        , 0.4472136 , 0.        , 0.4472136 ],
       [0.        , 0.45212331, 0.34385143, 0.34385143, 0.        ,
        0.34385143, 0.34385143, 0.34385143, 0.45212331, 0.        ]])
```

Q. Why do we take a log to calculate IDF?

If we have a very rare word, the IDF value without a log is very high. And then we have to calculate Tf * If at that time Idf value will dominate the Tf value because the Tf value lies from 0 to 1. That means we normalize the IDF value using a log.

Advantage

1. This technique is widely used in Information retrieval like a search engine.

Disadvantage

1. Sparsity
2. If we have a large dataset then dimensionality increases, slowing down algos.
3. OOV, Ignoring the new word.
4. Semantic meaning does not capture.

5. Custom Features

Creating new custom features using domain knowledge.

examples

1. Number of a word in the document.
2. Number of negative words in the document.
3. Ratio of +ve review to -ve review.
4. Word count.
5. Character count.

6. Word2vec

Word Embeddings

Wikipedia says “In natural language processing, word embedding is a term used for the representation of words for text analysis, typically in the form of a real-valued vector that encodes the meaning of the word such that the words that are closer in the vector space are expected to be similar in meaning.”

let’s consider one example boy-man vs boy-table, Can you tell which of the pair has more similar words to each other?

For a human, it’s easy to understand the associations between words in a language. We know that boy and man have more similar meanings than boy and table but what if we want machines to understand this kind of relation automatically in our languages as well? That is what word embeddings come into the picture.

Types of Word Embedding

1. Frequency-based – Count frequency of word
    1. BOW
    2. Tf-idf
    3. Glove(based on Matric Factorization)
2. Prediction based
    1. Word2Vec

What is Word2Vec?

Word2Vec is somewhat different than other techniques which we discussed earlier because it is a Deep learning-based technique.

Word2Vec is a word embedding technique, that converts a given word into a vector as a collection of numbers.

As we have other techniques then why do we need word2vec?

1. Word2vec capture semantic meaning like happiness and joy have the same meaning.
2. Word2vec create low dimension vector(each word is a collection of a range of 200 to 300.
3. Word2vec creates a Dense vector(non-zeros)

How to Use It?

We have two approaches to use Word2Vec

1. Use a pre-trained model

2. Self-Trained model

```
[44] sent = [['We', 'are', 'learning', 'Natural', 'Language', 'Processing.'],
     ['We', 'are', 'learning', 'Data', 'Science.'],
     ['Natural', 'Language', 'Processing', 'comes', 'under', 'Data', 'Science']]
```

```
[45] model = Word2Vec(sent, min_count=1,size=100)
```

```
[46] print(model)

     Word2Vec(vocab=12, size=100, alpha=0.025)
```

```
[47] words = list(model.wv.vocab)
     print(words)

     ['We', 'are', 'learning', 'Natural', 'Language', 'Processing.', 'Data', 'Science.', 'Processing', 'comes', 'under', 'Science']
```

```
[51] print(model['learning'].shape)
     print(model['learning'])

     (100,)
     [-3.6961546e-03 -2.6269942e-03  8.7758398e-04 -3.8207523e-03
      -3.0689037e-03 -4.6555917e-03 -2.9484809e-03 -1.1377539e-03
       3.0795666e-03 -3.3104317e-03  2.8665327e-03  5.2075350e-04
       1.0757345e-04  1.1974319e-03  4.8525701e-03  1.7803089e-03
       2.0558801e-03 -1.2477032e-03 -4.1798567e-03 -1.4200310e-03
      -1.8920181e-03  5.9364579e-04 -1.5910589e-03  2.9696017e-03
       2.9846141e-04  3.6758489e-03  3.6999460e-03  1.3158120e-03
       3.6706710e-03  3.9097141e-03  8.4801391e-04  7.0084195e-04
       1.9501924e-05  3.3038950e-03  1.0337532e-03 -2.6573497e-03
       6.6759216e-04 -4.2170109e-03  5.1860814e-04 -3.9811381e-03
      -2.9249403e-03  5.8123260e-04  4.4140718e-03  2.0246889e-04
```

Conclusion

In this article, we learned about different types of feature extraction techniques. The key takeaways from the article are,

- We learned different types of feature extraction techniques such as one-hot encoding, bag of words, TF-IDF, word2vec, etc.
- One Hot Encoding is a simple technique giving each unique word zero or one.
- A bag-of-words is a representation of text that describes the occurrence of words within a document.
- TF-IDF is a statistical measure that evaluates how relevant a word is to a document in a collection of documents.
- and at the last we have seenWord2Vec. Word2Vec is a word embedding technique, that converts a given word into a vector as a collection of numbers.
- Each technique has its pros and cons.

So, this was all about feature extraction techniques. Hope you liked the article.

The media shown in this article is not owned by Analytics Vidhya and is used at the Author's discretion.

blogathon Encoding feature extraction Feature Extraction Features Word2Vec