# Text encoding for use in machine learning

*Posted by Rasmus Nordbjærg at https://github.com/rasmus-bn/Investigation-Reporting-Blog on December 2019*

***Artificial Intelligence algorithms doesn't understand raw text data. However the world is drowning in text data and Artificial Intelligence is our best tool for processing and analyzing large data quantities. Fortunately text data can be encoded into a format that are understandable by Artificial Intelligence. Encoding text data allows you to ride the wave of Artificial Intelligence in the expanding ocean of collected text data.***

## Introduction

In a world with a continously rising amout of data[1] it is essential to be able to automate the processing of the data. Machine learning or artificial intelligence is a great tool for analyzing large amounts of data. However the algorithms are based on math and can not be directly fed with images or text data. The algorithms only understands numbers so in order to use machine learning to process texts or images the data needs to be preprocessed.

Machine learning models work by calculating a result *y* from the input variable *X*. However a model can have multiple input and output variables. Input variables are called **features** and the collection of input variables for a machine learning model is called a **feature vector**[2].

An example case of a machine learning model could be this: *The model should predict how much more time a pizza should stay in the oven before it is done. We know how long the pizza has already been in the oven and we know the oven temperature so these would be the features (input variables)*:

- $X_1$ = time passed
- $X_2$ = oven temperature

*We want to predict how much time is left before we can the pizza out:*

- y = time left

The important thing to take away is that the model only understand *X*s and the *y*s and these variables can only be given numerical values. There exists multiple methods for encoding text data into numerical features and I want to compare some of these. In the end I want to answer the below question:

*How can I encode text data so that a machine learning algorithm can understand the data?*

Due to the scope of the blog I will only explain different ways of using Bag of Words. I will therefore not dive into word stemming, removal of stopwords or other ways of preparing the text data. I will also leave out exlpaining n-grams for the sake of simplicity and easy readability.

# Bag of Words

In my machine learning exam project[3] I had to predict whether a review was positive or negative based on the written text of the reviewer. This sort of analysis is called sentiment anelysis and I will base my explanations on that kind of use case. So the input is a text that needs to be encoded into features and the ouput is a binary value: positive or negative.

Bag of Words means having one feature for every unique word in the dataset[4]. If a dataset only contains 3 unique words the input for the machine learning model would consist of three features. In order to represent a text the features would be assigned numerical values. A scoring method is used to calculate what the numerical value should be. I will got through the three scoring methods that we considered in my machine learning project.

Bag of Words only tracks what words are mentioned in the text and not in which order the words occured. This means that the original text can not be recovered from the encoded data. However in the use case of sentiment analysis the purpose is to predict whether the inputted text is negative or positive so we don't need the ability to recover the text from the encoded representation.

## Binary scoring method

With binary scoring[5] each method can have either 0 or 1. If a feature is given the value of 1 it means that the corresponding word exists the text and 0 means that it doesn't. Below is an image demonstrating this in python code. The string array *corpus* is the data set from which the features are defined.

```
In [2]: from sklearn.feature_extraction.text import CountVectorizer
corpus = [
    "Look. A rose",
    "That is a rose not a tulip",
    "A horse is a horse, and a rose is a rose",
    "A horse is not a rose",
    "A rose is not a horse"
]

binary_BoW = CountVectorizer(stop_words=None, token_pattern=u"(?u)\\b\\w+\\b", binary=True)
binary_BoW.fit(corpus)

encoded_text = binary_BoW.transform(["A rose is a rose is a rose, and not a tulip",])

print(format_elements(binary_BoW.get_feature_names()))
print(format_elements(encoded_text.toarray()[0]))
```

```
a       and     horse   is      look    not     rose    that    tulip
1       1       0       1       0       1       1       0       1
```

*https://raw.githubusercontent.com/rasmus-bn/Investigation-Reporting-Blog/master/images/Binary%20scoring.png - A screenshot from 'Encoding examples.ipynb' in the blog repo*

The limit of this method is that it does not take into account the number of occurences of the words. These 2 sentences would equal eachother if encoded with the binary scoring method: "It really was a good movie", "It was a really, really good movie".

```
In [3]: encoded_text = binary_BoW.transform(["A rose is a rose",])
        print(format_elements(binary_BoW.get_feature_names()))
        print(format_elements(encoded_text.toarray()[0]) + '\n')

        encoded_text = binary_BoW.transform(["A rose is a rose is a rose",])
        print(format_elements(binary_BoW.get_feature_names()))
        print(format_elements(encoded_text.toarray()[0]))
```

```
a       and     horse   is      look    not     rose    that    tulip
1       0       0       1       0       0       1       0       0

a       and     horse   is      look    not     rose    that    tulip
1       0       0       1       0       0       1       0       0
```

*https://raw.githubusercontent.com/rasmus-bn/Investigation-Reporting-Blog/master/images/Binary%20duplicates.png - A screenshot from 'Encoding examples.ipynb' in the blog repo*

## Count based scoring

Instead of just telling whether the word is used in the text we could use count based scoring. This means that the numerical value given to a feature should be the number of occurences for the word occurs in the text[5]. This is demonstrated in below image.

```
In [3]: from sklearn.feature_extraction.text import CountVectorizer
        corpus = [
            "Look. A rose",
            "That is a rose not a tulip",
            "A horse is a horse, and a rose is a rose",
            "A horse is not a rose",
            "A rose is not a horse"
        ]

        count_based_BoW = CountVectorizer(stop_words=None, token_pattern=u"(?u)\\b\\w+\\b")
        count_based_BoW.fit(corpus)

        encoded_text = count_based_BoW.transform(["A rose is a rose is a rose, and not a tulip",])

        print(format_elements(count_based_BoW.get_feature_names()))
        print(format_elements(encoded_text.toarray()[0]))
```

```
a       and     horse   is      look    not     rose    that    tulip
4       1       0       2       0       1       3       0       1
```

*https://raw.githubusercontent.com/rasmus-bn/Investigation-Reporting-Blog/master/images/Count%20based%20scoring.png - A screenshot from 'Encoding examples.ipynb' in the blog repo*

Using this method means that the encoding contains information of what words was in the text and how many times they were used.

## TF-IDF

The count based scoring seemed like it had all the functionality needed to perform sentiment analysis. However the scoring method called TF-IDF adds just some extra nice functionality. TF-IDF stands for Term Frequency - Inverse Document Frequency[6]. Instead of being based on occurence count the TF-IDF is based on the occurence frequency in the current text and the inverse frequency of the word in the entire dataset. This means that words that are rare in the dataset will be weighted higher than words that are very common[5]. Below is an example of the TF-IDF does in use:

```
In [4]: from sklearn.feature_extraction.text import TfidfVectorizer
corpus = [
    "Look. A rose",
    "That is a rose not a tulip",
    "A horse is a horse, and a rose is a rose",
    "A horse is not a rose",
    "A rose is not a horse"
]

tfidf_BoW = TfidfVectorizer(stop_words=None, token_pattern=u"(?u)\\b\\w+\\b")
tfidf_BoW.fit(corpus)

encoded_text = tfidf_BoW.transform(["A rose is a rose is a rose, and not a tulip",])

print(format_elements(tfidf_BoW.get_feature_names()))
print(format_elements(encoded_text.toarray()[0]))
```

```
a       and     horse   is      look    not     rose    that    tulip
0.62    0.33    0.0     0.37    0.0     0.22    0.47    0.0     0.33
```

*https://raw.githubusercontent.com/rasmus-bn/Investigation-Reporting-Blog/master/images/TF-IDF%20scoring.png - A screenshot from 'Encoding examples.ipynb' in the blog repo*

As shown in the example the more common words used like 'rose' and 'is' be weighted as less important than the more rare words like 'tulip'. 'tulip' is mentioned once in the text and has a score of 0.33 which is very close to the score of 'is' at 0.37 which is mentioned 3 times.

This is useful in the case of sentiment analysis. If many positive and negative negative would mention a specific word the that word will have less significant meaning when trying to predict positive or negative. The TfidfVectorizer takes care of down-prioritising those kind of words. The machine learning library for python called scikit-learn reccommends[7] TfdifVectorizer[8] over CountVecotizer[9] when working with text data. This was also the method we went with.

## Conclusion

Using Bag of Words way of encoding with TF-IDF scoring method is a great way to encode text data to feed into a machine learning model. It provides a score for each word, based on how many times the word occurs in a text and how rare that word is. This is great because the it helps sorts out the more significant words from the less.

If you need the ability to extract the original text from the encoded data the Bag of Words is not the tool for you. However in the use case of sentiment analysis where the predicted result is whether or not the text was positive or negative Bag of Word with TF-IDF is great way to encode the data.

## Follow-up reflection

Encoding the text data is only a small step of building the classification model. We had multiple data preprocessing steps that we went through such as word stemming, stopwords removal and other text cleanup. We also engineered some extra features such as word count, stopword count & frequency, text length in characters.

We used TF-IDF in our machine learning exam project and we managed to get a prediction accuracy of arround 80%. However this result was not credited alone to TF-IDF as many other factors were in place.

# References


1. Wikipedia on Global Internet usage (last read on 10/12-19): https://en.wikipedia.org/wiki/Global_Internet_usage
2. Wikipedia on Features in machine learning (last read on 10/12-19): https://en.wikipedia.org/wiki/Feature_(machine_learning)
3. My machine learning exam project (state of master branch 10/12-19): https://github.com/rasmus-bn/MLExamProject
4. Wikipedia on Bag of Words (last read on 10/12-19): https://en.wikipedia.org/wiki/Bag-of-words_model
5. Bag of words - Jason Brownlee 09/10-19 (last read on 10/12-19) https://machinelearningmastery.com/gentle-introduction-bag-words-model/
6. Wikipedia's explanation of Term Frequency - Inverse Document Frequency (last read on 10/12-19): https://en.wikipedia.org/wiki/Tf%E2%80%93idf
7. scikit-learn's guide on working with text data recommending TFIDFVectorizer over CountVectorizer (last read on 10/12-19): https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html
8. scikit-learn's TfidfVectorizer (last read on 10/12-19): https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
9. scikit-learn's CountVectorizer (last read on 10/12-19): https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html