DATA X

NLP Module: Feature Engineering & Text Representation

Berkeley SCET

NLP Process

Text Processing

Clean up the text to make it easier to use and more consistent to increase prediction accuracy later on

Feature Engineering & Text Representation

Learn how to extract information from text and represent it numerically

Learning Models

Use learning models to identify parts of speech, entities, sentiment, and other aspects of the text.

Feature Engineering & Text Representation

One-Hot-Encoding

Transforms categorical feature into many binary features

Bag of Words Model Using Countvectorizer

Generalization of one-hot-encoding for a string of text

N-Gram Encoding

Captures word order in a vector model

TFIDF

Converts a collection of raw documents to a matrix of TFIDF features

What is Feature Engineering & What is Text Representation?

Feature engineering is the process of transforming the raw data to improve the accuracy of models by creating new features from existing data

Text Representation is numerically representing text to make it mathematically computable

One Hot Encoding

One Hot Encoding

Encodes categorical data as real numbers such that the magnitude of each dimension is meaningful

For each distinct possible value, a new feature is created

Exp:

	name	kind	age
0	Goldy	Fish	0.5
1	Scooby	Dog	7.0
2	Brian	Dog	3.0
3	Francine	Cat	10.0
4	Goldy	Dog	1.0

	age	name_Brian	name_Francine	name_Goldy	name_Scooby	kind_Cat	kind_Dog	kind_Fish
0	0.5	0	0	1	0	0	0	1
1	7.0	0	0	0	1	0	1	0
2	3.0	1	0	0	0	0	1	0
3	10.0	0	1	0	0	1	0	0
4	1.0	0	0	1	0	0	1	0

One Hot Encoding in Scikit-Learn

```
from sklearn.preprocessing import OneHotEncoder

oh_enc = OneHotEncoder()

oh_enc.fit(df[['name', 'kind']])

oh_enc.transform(df[['name', 'kind']]).todense()
```

One Hot Encoding in Pandas

```
pd.get_dummies(df[['name', 'kind']])
```

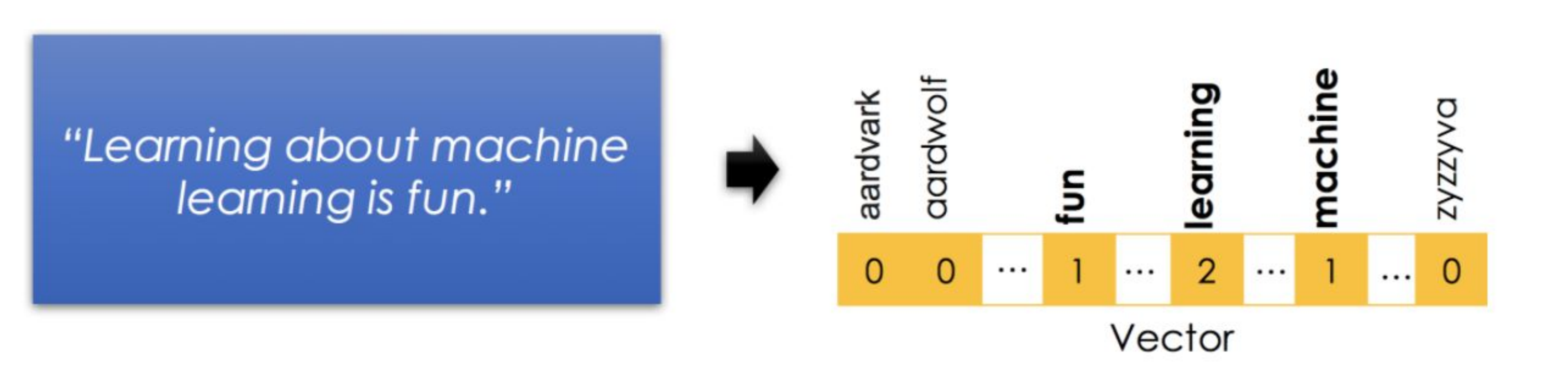
Bag of Words Model

Bag of Words Model

Extracts features from text

Stop words not included, word order is lost, sparse encoding

Exp:



Bag of Words Model using Scikit-Learn

sample_text = ['This is the first document.', 'This document is the second document.', ‘This is the third document.’]

from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(stop_words="english")

vectorizer.fit(sample_text)

#to see what words were kept

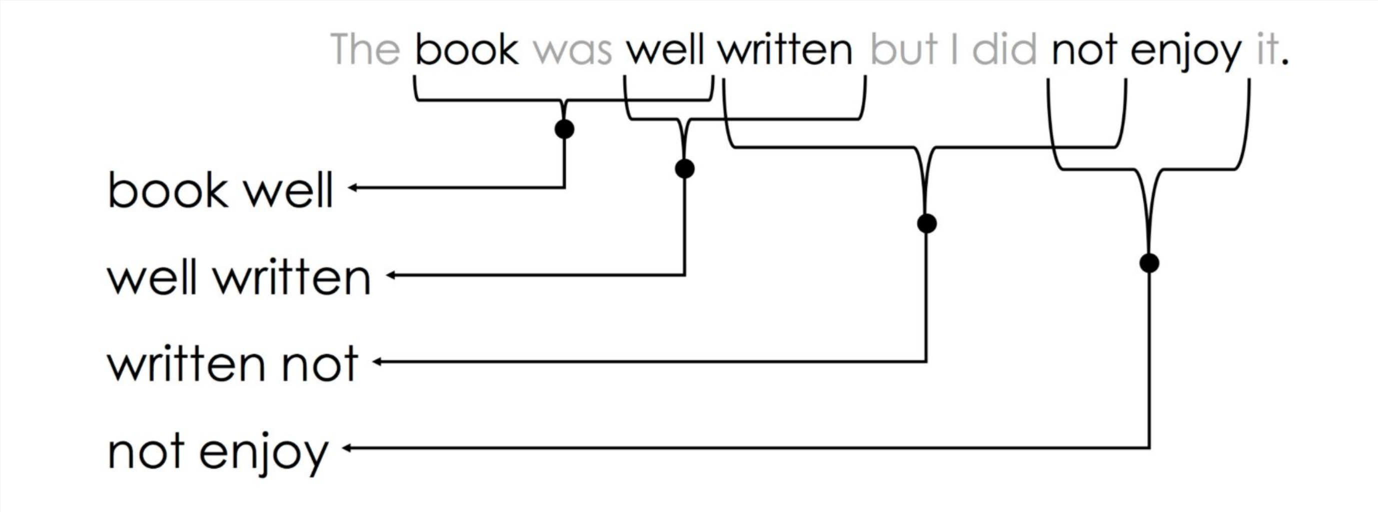
print("Words:", list(enumerate(vectorizer.get_feature_names())))

N-gram Encoding

N-gram Encoding

Extracts features from text while capturing local word order by defining counts over sliding windows

Exp n =2 :



N-gram Encoding using Scikit-Learn

sample_text = ['This is the first document.', 'This document is the second document.', ‘This is the third document.’]

from sklearn.feature_extraction.text import CountVectorizer

bigram = CountVectorizer(ngram_range=(1, 2))

bigram.fit(sample_text)

#to see what words were kept

print("Words:", list(zip(range(0,len(bigram.get_feature_names())), bigram.get_feature_names())))

TFIDF Vectorizer

TFIDF Vectorizer

Converts a collection of raw documents to a matrix of TFIDF features

What are TFIDF Features?

TFIDF stands for term frequency inverse document frequency and it represents text data by indicating the importance of the word relative to the other words in the text

2 Parts:

TF: (# of times term t appears in a document)/ (total # of terms in the document)

IDF: (log10 (total # of documents)/(# of documents with term t in it)

TFIDF Vectorizer con.

TFIDF = TF*IDF

TF represents how frequently the word shows up in the document

IDF represents how important the word is to the document (rare words)

TFIDF Vectorizer Encoding using Scikit-Learn

```
from sklearn.feature_extraction.text import TfidfVectorizer

sample_text =  ['This is the first document.', 'This document is the second document.', ‘This is the third document.’ ]

vectorizer = TfidfVectorizer()

X = vectorizer.fit_transform(sample_text)

print(vectorizer.get_feature_names())
```