使用NLP从文章中自动提取关键字

大雄没有叮当猫 数据挖掘与机器学习笔记 8月29日

1.背景

在许多研究和新闻文章中，关键字是其中的一个重要组成部分，因为关键字提供了文章内容的简洁表示。关键字在从信息检索系统，数据库中查找文章以及优化搜索引擎中起着至关重要的作用。此外，关键字还有助于将文章分类为相关主题或者学科。

提取关键字的常规方法包括根据文章内容和作者的判断手动分配关键字，但是这需要花费大量的时间和精力，并且在选择适当的关键字方面也可能不准确。随着NLP的发展，关键字提取也变得更加高效、准确。

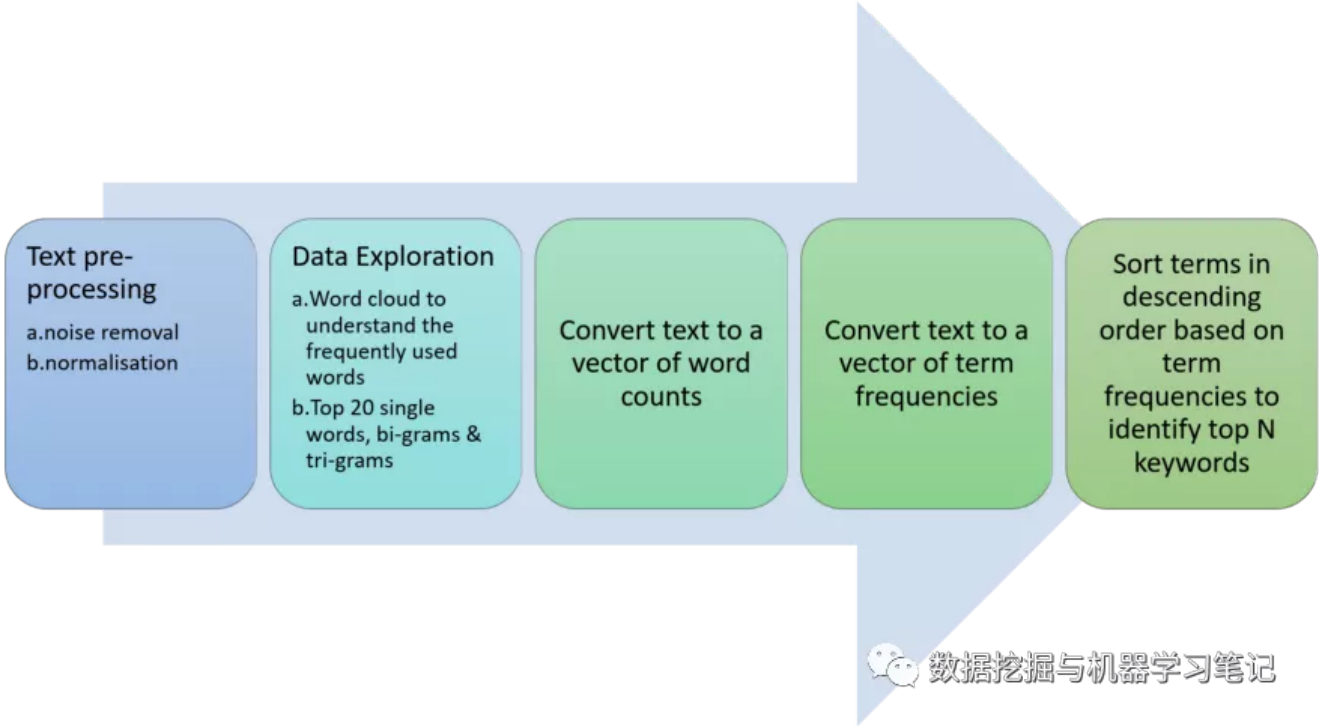
在下面的文章中，我们将展示使用NLP来进行关键字提取。

2.数据集

在本文中，我们将从包含约3800个摘要的数据集中提取关键字。原始数据集来源于kaggle-NIPS Paper,神经信息处理系统(NIPS)是世界上顶级的机器学习会议之一。该数据集包括从1987年第一次会议到2016年会议的NIPS论文的标题和摘要。

原始数据集还包含文章文本，但是，由于重点是理解关键字提取的概念，并且使用全部文章内容可能会占用大量计算资源，因此仅使用摘要用于NLP建模。

3.方法



3.1 文本预处理和数据探索

(1)导入数据集

本文使用的数据集是Kaggle上NIPS论文数据集中papers.csv文件数据集。需要对原始数据集进行一些处理。

```
#加载原始数据集
filePath="/content/drive/My Drive/data/关键字提取/papers.csv"
dataset=pd.read_csv(filePath)
dataset.head()
```

	id	year	title	event_type	pdf_name	abstract	paper_text
0	1	1987	Self-Organization of Associative Database and ...	NaN	1-self-organization-of-associative-database-an...	Abstract Missing	767\n\nSELF-ORGANIZATION OF ASSOCIATIVE DATABA...
1	10	1987	A Mean Field Theory of Layer IV of Visual Cort...	NaN	10-a-mean-field-theory-of-layer-iv-of-visual-c...	Abstract Missing	683\n\nA MEAN FIELD THEORY OF LAYER IV OF VISU...
2	100	1988	Storing Covariance by the Associative Long-Ter...	NaN	100-storing-covariance-by-the-associative-long...	Abstract Missing	394\n\nSTORING COVARIANCE BY THE ASSOCIATIVE\n...
3	1000	1994	Bayesian Query Construction for Neural Network...	NaN	1000-bayesian-query-construction-for-neural-ne...	Abstract Missing	Bayesian Query Construction for Neural\nNetwor...
4	1001	1994	Neural Network Ensembles, Cross Validation, an...	NaN	1001-neural-network-ensembles-cross-validation...	Abstract Missing	Neural Network Ensembles, Cross\nValidation, a...

数据挖掘与机器学习笔记

数据集包含七列，而"abstract"那列表示论文摘要，可以看到其中存在大量的“Abstract Missing”，即没有摘要的论文，我们需要将其去掉。这里，我只简单地保留了字符数超过50的摘要，最后剩下3900多个摘要。

```
dataset=dataset[dataset['abstract'].str.len()>50]
```

```
len(dataset) #3924
```

(2)初步文本探索

在进行文本预处理之前，建议快速的统计摘要中单词数量，看看最常见和最不常见的单词都有哪些。

```
#统计每个摘要中有多少个单词
dataset["word_count"]=dataset["abstract"].apply(lambda x:len(str(x).split(" ")))
dataset[["abstract","word_count"]].head()
```

	abstract	word_count
941	Non-negative matrix factorization (NMF) has pr...	107
1067	Spike-triggered averaging techniques are effec...	81
2384	It is known that determinining whether a DEC-P...	67
2385	We present the first truly polynomial algorith...	143
2388	Semi-supervised inductive learning concepts h...	119

数据挖掘与机器学习笔记

```
#对每个摘要中的单词数量做一个描述性统计
dataset.word_count.describe()
```

```
count    3924.000000
mean      148.390928
std        45.605755
min        19.000000
25%       116.000000
50%       143.000000
75%       177.000000
max       317.000000
Name: word_count, dtype: float64
```

数据挖掘与机器学习笔记

摘要的平均单词数量为148个，单词数量在19到317之间。单词数量很重要，可以让我们知道正在处理的数据集的大小以及摘要单词数量的变化。

另外，也可以看看摘要当中一些常见的单词和稀有单词。比较常见的单词可以添加到停用词表当中。

```
#最常见的单词
freq=pd.Series("".join(dataset["abstract"]).split()).value_counts()[:20]
freq
```

the	29793
of	20918
a	16339
and	13626
to	12869
in	8980
that	7838
is	7666
for	7169
on	5579
we	5167
We	4995
with	4512
this	

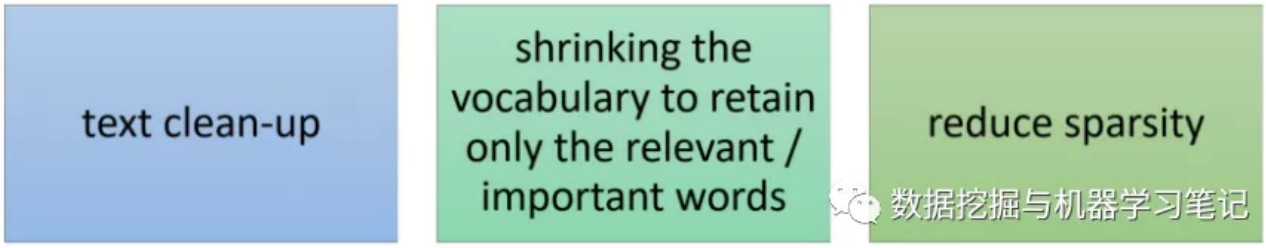
数据挖掘与机器学习笔记

```
#最不常见的单词
freq1=pd.Series(" ".join(dataset["abstract"]).split()).value_counts()[-20:]
freq1
```

co-occurrence.	1
al.~(2012).	1
requested,	1
Data"	1
Vectors	1
attenuated	1
encloses	1
maxing	1
downside	1
specification.	1
Attentive	1
Classically,	1
(CVaR).	1
tween	1
\emph{similarity}.	1
Wabbit	

数据挖掘与机器学习笔记

(3)文本预处理



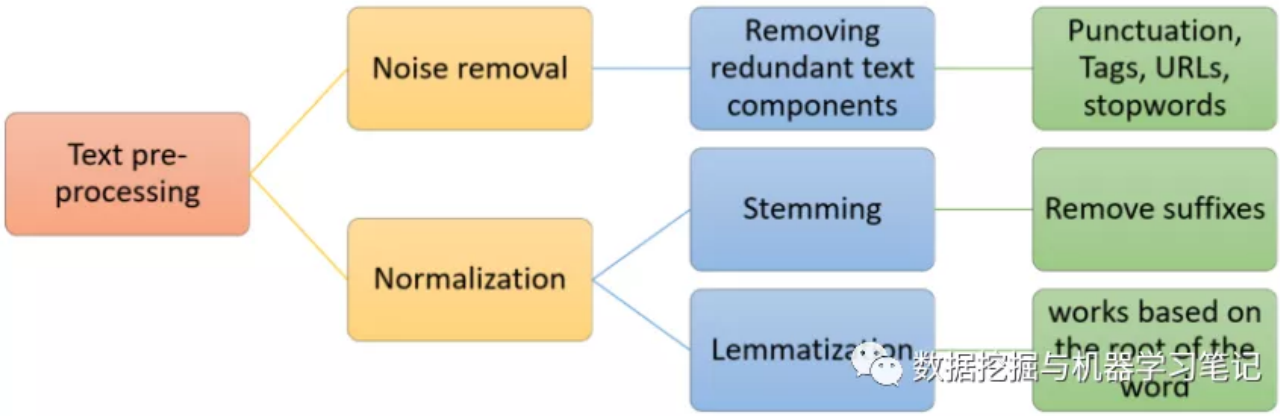

```
#文本预处理
from nltk.stem.porter import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

lem=WordNetLemmatizer()
stem=PorterStemmer()
word="inversely"
print("stemming:",stem.stem(word))
print("lemmatization:",lem.lemmatize(word,"v"))
```

- 稀疏性

在文本挖掘中，基于词频创建的巨大的稀疏矩阵，其中存在许多值为0.这个问题称为数据稀疏性，使用各种技术可以将其最小化。

文本预处理可以分为两类:噪声消除和标准化。核心文本分析中多余的数据组成部分可以被视为噪声。



而处理同一单词不同表现形式的多次出现则称为标准化。主要有两种形式：词干(Stemming)和词形还原（Lemmatisation）。举个例子：learn,learned,learning,learner等都是单词learn的不同版本。标准化会将这些单词转为单一标准化的版本"learn"。

- 词干通过删除后缀来规范化文本

- 词性还原是一种更高级的技术，它基于单词的词根进行工作

以下示例说明了词干和词形还原的工作方式：

```
#文本预处理
from nltk.stem.porter import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

lem=WordNetLemmatizer()
stem=PorterStemmer()
word="inversely"
print("stemming:",stem.stem(word))
print("lemmatization:",lem.lemmatize(word,"v"))
```

为了对数据集进行文本预处理，首先导入所需的库

```
import nltk
import re
from nltk.corpus import stopwords
from nltk.tokenize import RegexpTokenizer
nltk.download("stopwords")
# nltk.download('wordnet')
```

去除停用词：停用词包括句子中大量的介词、代词、连词等。在分析文本之前，需要先删除这些单词，以便经常使用的单词主要是与上下文相关的单词，而不是文本中最常见的单词。

python nltk库中有默认的停用词列表，我们也可以自己添加自定义的停用词。

```
#停用词
stop_words=set(stopwords.words("english"))

new_words=["using","show","result","large","also","iv","one","two","new","previously","shown"]
stop_words=stop_words.union(new_words)
```

还需要对文本做一些处理，比如去除标点符号、小写化等

```
corpus=[]
for text in dataset["abstract"]:
  #去除标点符号
  text=re.sub("[^a-zA-Z]"," ",text)
```

```
    #转换成小写
    text=text.lower()
    #去除标签
    text=re.sub("&lt;/?.*?&gt;"," &lt;&gt; ",text)
    #去除特殊符号和数字
    text=re.sub("(\\d|\\W)+"," ",text)
    #字符串转为List
    text=text.split()
    ##Stemming
    ps=PorterStemmer()
    #Lemmatisation
    lem=WordNetLemmatizer()
    text=[lem.lemmatize(word) for word in text if not word in stop_words]
    text=" ".join(text)
    corpus.append(text)
```

(4) 数据探索

现在，对经过预处理的语料进行可视化，以便了解最常用的单词。

```
from os import path
from PIL import Image
from wordcloud import WordCloud,STOPWORDS,ImageColorGenerator
import matplotlib.pyplot as plt
%matplotlib inline
wordcloud=WordCloud(background_color="white",stopwords=stop_words,max_words=100,max_font_size=50,

print(wordcloud)
fig=plt.figure(1)
plt.imshow(wordcloud)
plt.axis("off")
plt.show()
```

image-20200829135426820

3.2 文本向量化

语料中的文本需要转换为机器学习算法可以解释的格式。包括两个部分：

Tokenisation and Vectorisation.

Tokenisation 是将文本转为单词列表的过程。然后通过矢量化将单词列表转为整数矩阵。*Vectorisation* 也称为特征提取。我们使用词袋模型，该模型仅考虑单词的频率，而忽略单词的时序。

（1）使用词频创建向量

```
from sklearn.feature_extraction.text import CountVectorizer
import re

cv=CountVectorizer(max_df=0.8,stop_words=stop_words,max_features=10000,ngram_range=(1,3))
X=cv.fit_transform(corpus)

list(cv.vocabulary_.keys())[:10]
```

（2）分别对uni-grams,bi-grams和tri-grams的topN进行可视化

```
#出现最频繁的单词
def get_top_n_words(corpus, n=None):
    vec = CountVectorizer().fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in
                  vec.vocabulary_.items()]
```

```
    words_freq =sorted(words_freq, key = lambda x: x[1],
                       reverse=True)
    return words_freq[:n]
#Convert most freq words to dataframe for plotting bar plot
top_words = get_top_n_words(corpus, n=20)
top_df = pd.DataFrame(top_words)
top_df.columns=["Word", "Freq"]
#Barplot of most freq words
import seaborn as sns
sns.set(rc={'figure.figsize':(13,8)})
g = sns.barplot(x="Word", y="Freq", data=top_df)
g.set_xticklabels(g.get_xticklabels(), rotation=30)
```

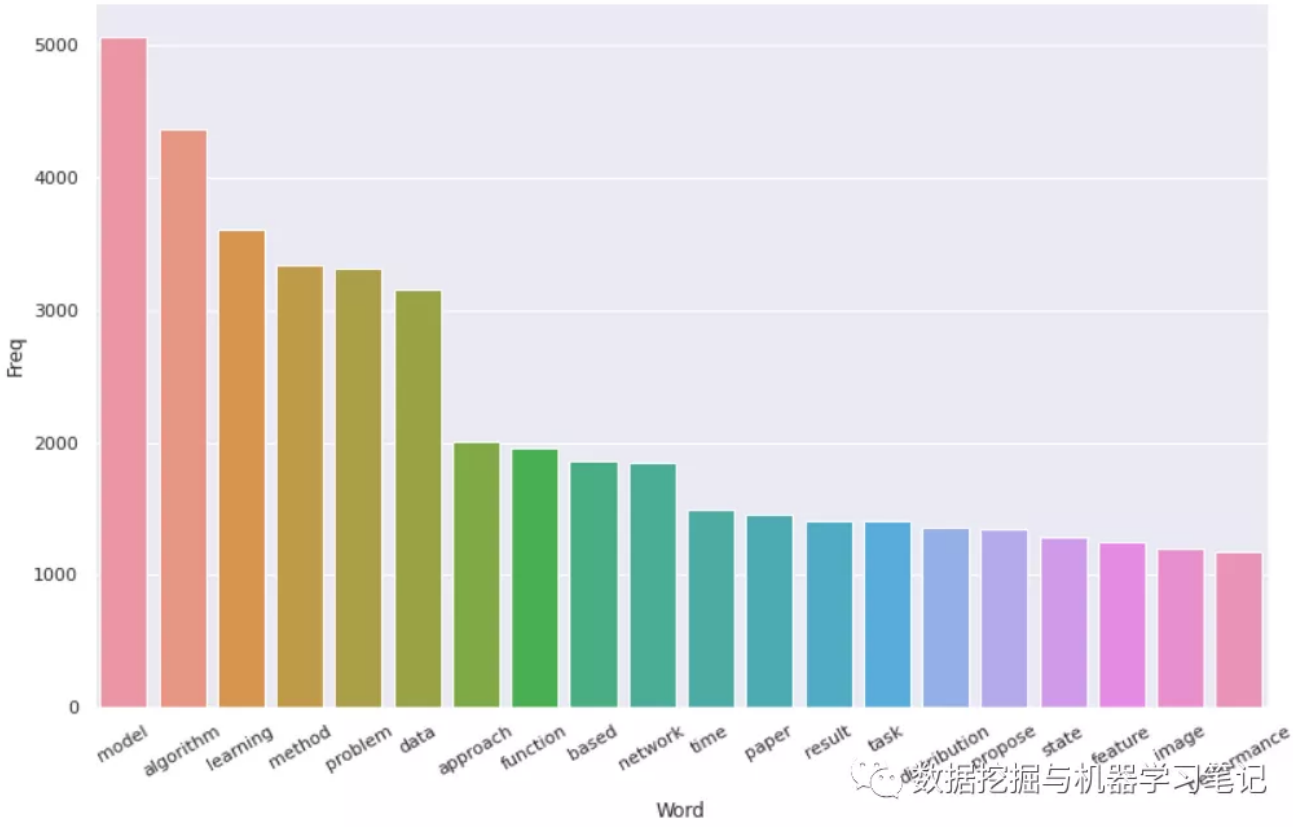
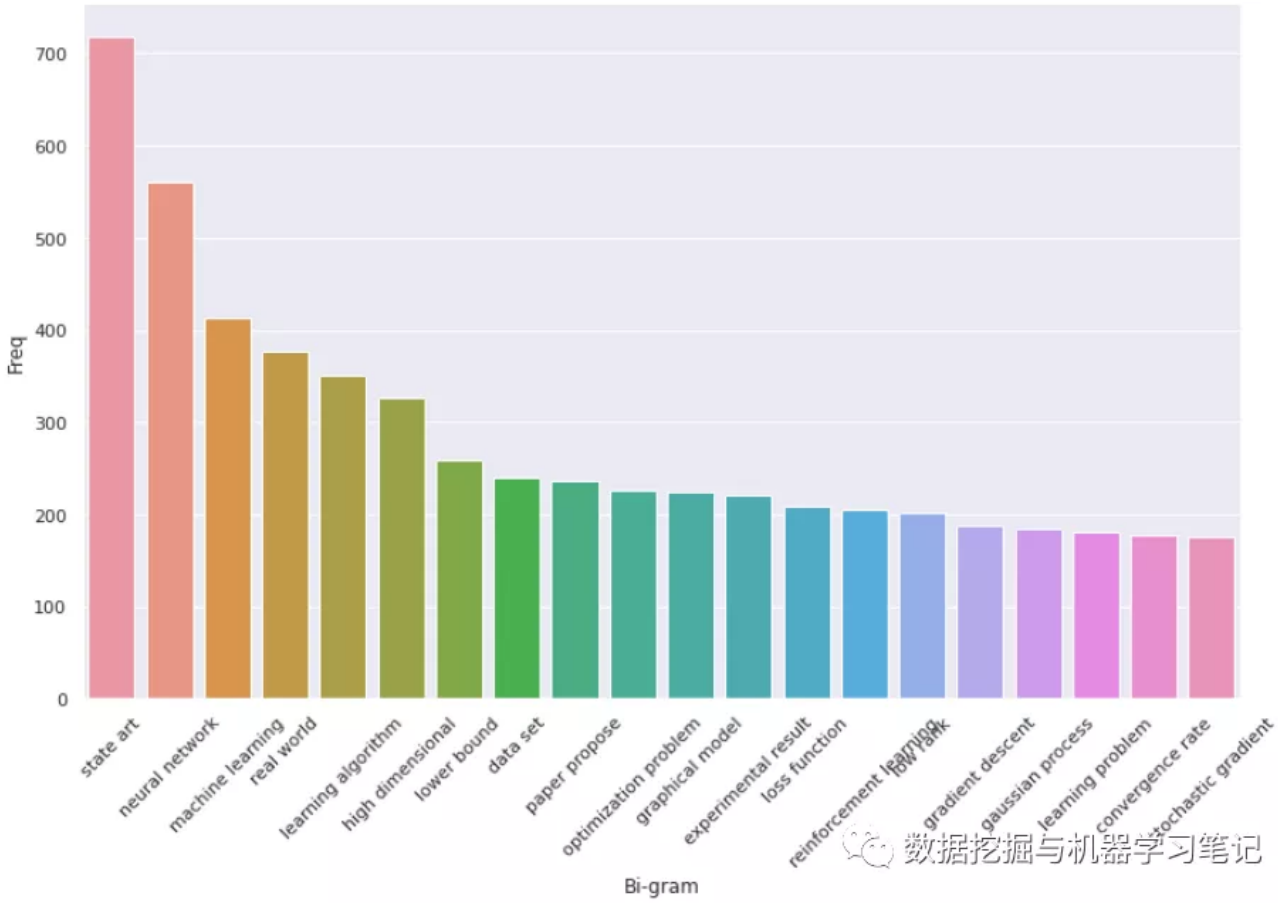


image-20200829140155713

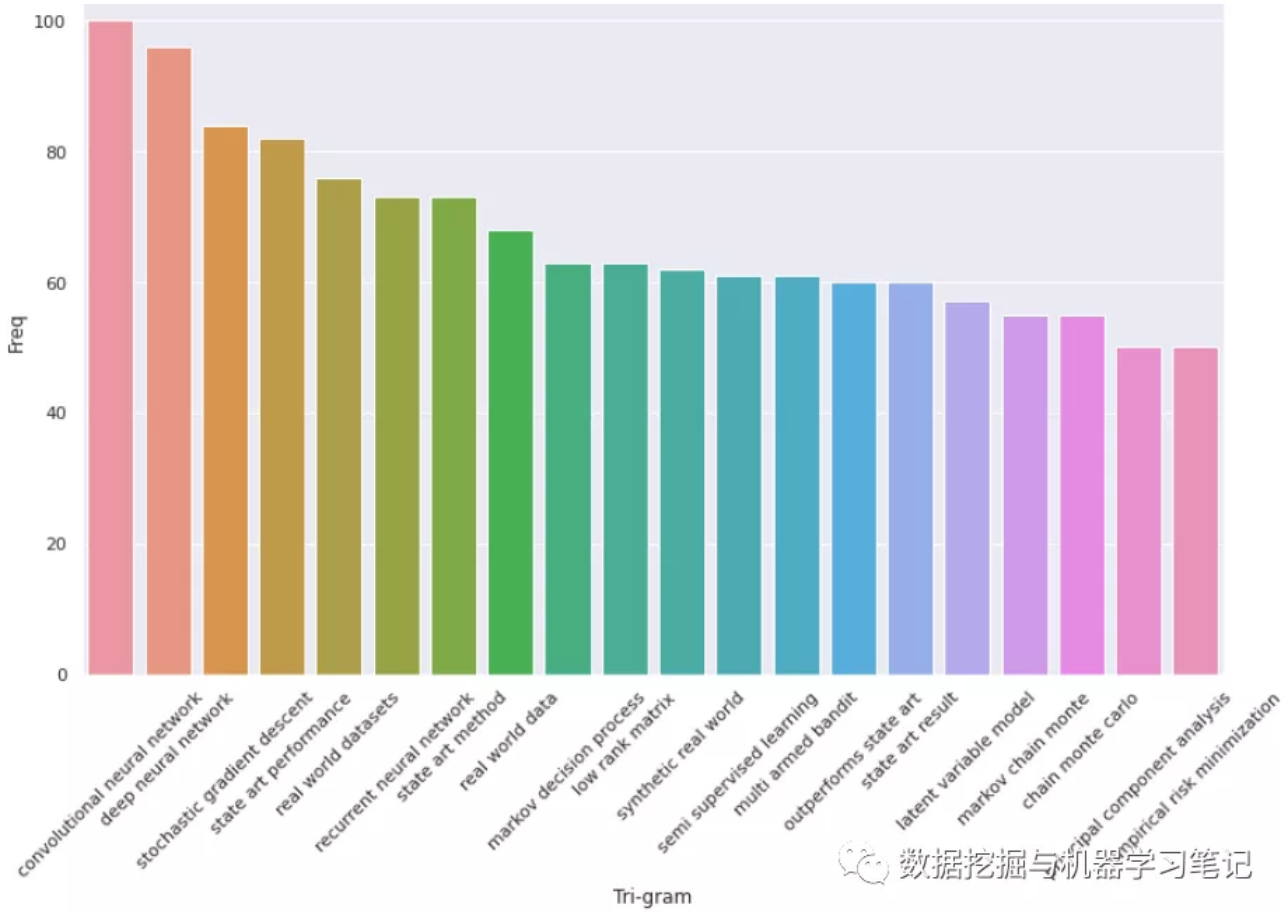
```
#最频繁出现的bi-gram
def get_top_n2_words(corpus, n=None):
    vec1 = CountVectorizer(ngram_range=(2,2),
            max_features=2000).fit(corpus)
    bag_of_words = vec1.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in
                  vec1.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1],
```

```
                reverse=True)
    return words_freq[:n]
top2_words = get_top_n2_words(corpus, n=20)
top2_df = pd.DataFrame(top2_words)
top2_df.columns=["Bi-gram", "Freq"]
print(top2_df)
#Barplot of most freq Bi-grams
import seaborn as sns
sns.set(rc={'figure.figsize':(13,8)})
h=sns.barplot(x="Bi-gram", y="Freq", data=top2_df)
h.set_xticklabels(h.get_xticklabels(), rotation=45)
```

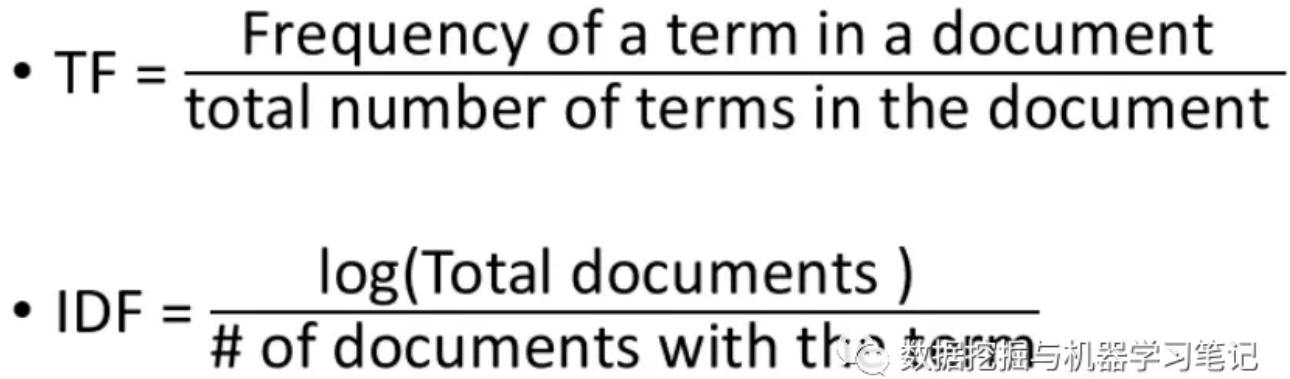



```
#Most frequently occuring Tri-grams
def get_top_n3_words(corpus, n=None):
    vec1 = CountVectorizer(ngram_range=(3,3),
           max_features=2000).fit(corpus)
    bag_of_words = vec1.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in
                  vec1.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1],
                reverse=True)
```

```
    return words_freq[:n]
top3_words = get_top_n3_words(corpus, n=20)
top3_df = pd.DataFrame(top3_words)
top3_df.columns=["Tri-gram", "Freq"]
print(top3_df)
#Barplot of most freq Tri-grams
import seaborn as sns
sns.set(rc={'figure.figsize':(13,8)})
j=sns.barplot(x="Tri-gram", y="Freq", data=top3_df)
j.set_xticklabels(j.get_xticklabels(), rotation=45)
```



3.3 使用TF-IDF提取关键字




```
from sklearn.feature_extraction.text import TfidfTransformer

tfidf_transformer=TfidfTransformer(smooth_idf=True,use_idf=True)
tfidf_transformer.fit(X)
# get feature names
feature_names=cv.get_feature_names()

# fetch document for which keywords needs to be extracted
doc=corpus[532]

#generate tf-idf for the given document
tf_idf_vector=tfidf_transformer.transform(cv.transform([doc]))


#Function for sorting tf_idf in descending order
from scipy.sparse import coo_matrix
def sort_coo(coo_matrix):
    tuples = zip(coo_matrix.col, coo_matrix.data)
    return sorted(tuples, key=lambda x: (x[1], x[0]), reverse=True)

def extract_topn_from_vector(feature_names, sorted_items, topn=10):
    """get the feature names and tf-idf score of top n items"""

    #use only topn items from vector
    sorted_items = sorted_items[:topn]

    score_vals = []
    feature_vals = []

    # word index and corresponding tf-idf score
    for idx, score in sorted_items:
```

```
            #keep track of feature name and its corresponding score
            score_vals.append(round(score, 3))
            feature_vals.append(feature_names[idx])

        #create a tuples of feature,score
        #results = zip(feature_vals,score_vals)
        results= {}
        for idx in range(len(feature_vals)):
            results[feature_vals[idx]]=score_vals[idx]

        return results
    #sort the tf-idf vectors by descending order of scores
    sorted_items=sort_coo(tf_idf_vector.tocoo())
    #extract only the top n; n here is 10
    keywords=extract_topn_from_vector(feature_names,sorted_items,5)

    # now print the results
    print("\nAbstract:")
    print(doc)
    print("\nKeywords:")
    for k in keywords:
        print(k,keywords[k])
```

阅读原文