$\prod_{t=1}^{k+1} p(w_t|w_{t-1}) \to -\sum_{t=1}^{k+1} \log p(w_t|w_{t-1})$ 防止浮点数连续相乘之后向下溢出，取对数

OOV：(out of vacabulary) 在字典外

文本生成也可以生成摘要，不过要进行语句检测。 { pagerank 有向图 / textrank 无向图

textrank 也可以进行关键词提取。

文本聚类 (demo_text_clustering.py)：没有标签的聚类 (无监督)。K-means (K均值聚类)

距离<=>相似性，迭代求解，随机选K个对象作为初始的聚类中心)，☆ microsoft edge 可以画图. ↳ 腾讯会议 可以开批注

把点划分最近的那一类。(K个中哪个点) 根据簇间距离判断，停止迭代。

K-means：jianshu.com/p/fc91fed8c77b，簇内尽量小，簇间尽量大。(目标函数)

↓ 可计算文本相似度 文本聚类 => 一般用主题模型，

↓ 切词，去停用词，文档向量 (doc2vec)

代码. cnblogs.com/bicoding/p/8878098.html

sigmoid 分类，.demo_word2vec 词向量，word2vec 词向量，一个词用一个向量表示 (13年谷歌)

跳字 (skip-gram) 和词袋模型 (CBOW)，其中 skip-gram 用中心词预测周围词。

用周围词(上下文)预测中心词. softmax 多分类.

丰度：是指一种化学元素在某个自然体中重量占这个自然体总重量的相对份额。

命名实体识别，使用BMES 标注集，不构成命名实体的单词，统一标注为 O(outside) (复合词之外)

↓ 现实存在的实体，

```
class ClusterAnalyzer():
    def __init__(self)
        pass   < self.document={}
               \ self.document_tfidf={}

    def adddocument(self, path):
        self.document=self.read_text(path)

def read_txt(self):
    pass

def text_process(self):
    pass

def cal_tfidf(self):
    pass

def cal_dist(self, arr1, arr2):
    pass

def cal_zone(self):
    pass
```

```
def save_text(self):
  pass
if __name__ == '__main__':
                   __file__
  data_path=os.path.join(data)
  kmeans=ClusterAanalyzer()
  kmeans.add_document(data_path)
  print(kmeans.document)
```

```
def add_document(self, path):
  for i in range(100):
    file_path=os.path.join(path, '{}.txt'.format(i+1))
    with open(file_path, encoding='utf-8') as f:
      self.document[str(i+1)]=f.read()
  self.~~text~~_process()
```

文件名不要和模块名一样的否则会出错

```
os.path.dirname(__file__)
```

上一层

有 debug 进程有 debug probe 可以用

init中再写 self.process_document={}, self.word_table=[]

```
segment=DoubleArrayTrieSegment()     stopwords=Stopwords()
def text_process(self):
  for file_id, text in self.document.items():
    self.process_document[file_id] = [term.word for term in self.segment.seg(text)
                                       if not self.stopwords.containsKey(term.word)]
      word_list.extend(~~self.process_document~~[file_id])
  self.word_table =list(set(word_list))
  def fit(self,k): #训练kmeans
    self.cal_tfidf()  #要表示为固定的维度
def cal_tfidf(self):
  for file_id, text in self.process_document.items():
    tfidf_list=[0]*len(self.word_table)
    for word in text:
      tfidf_list.~~insert~~(self.word_table.index(word), self.tf(word, text)*self.idf(word, text))
self.document_itidf[file_id] = tfidf_list
```

可生成定长的全部0的列表

不是插入,是更新

词

文本

```
def tf(self, word):
6   return text.count(word)/len(text)
                                                     file_id, text
def idf(self, word,t):                                 process_
    return np.log(len(self.document)/sum([1 for file_id, text self.process_document.items() if word in
                                                                          text]))
def fit(self, k=5 → 簇数, num=50 → 迭代次数):
                                                       文本id
    self.cal_tfidf()                            {'1':[1,2,3], '2':[]}
                                                  簇类
    self.cluster_dict=dict.fromkeys                    改为 {'center':[ ], 'file_id':[ ]}
            (range(1,k+1), {})
                  'center': None, 'file_id': None   dict.fromkeys(seq[,value])
                                                         键        → 对应值
    随机初始化K个质心:
    self.cluster_dict=dict.fromkeys(range(1,k+1),{'center': ~~None~~ [], 'file_name': ~~None~~ []})
    print(random.sample(range(1,len(self.process_document)), k))
                        用 指定序列生成
    for index, i in enumerate(random.sample(range(1,len(self.process_document)),k)):
        self.cluster_dict[~~self~~(index+1)]['center'] = self.document_itidf['i'] → 改为str(i)
    counter
    while count<num:  #计算所有文本到质心的距离
        for file_id, text in self.document_itidf.items():
            #遍历所有质心
            for cluster_id, cluster in self.cluster_dict.items():
                text_dist.append(self.dist(text, cluster['center']))
            min_dist_idx = text_dist.index(min(text_dist))
            self.cluster_dict[min_dist_idx]['file_id'].append(file_id)
        count+=1
def cal_dist(self, arr1, arr2):                 用numpy更快，底层用cpath,
    return np.sqrt(np.sum((np.array(arr1) - np.array(arr2))**2))
```

```
for i in range(1, k+1):    #fromkey: 对应value 创建了的值都是一样的
    self.cluster_dict[i] ={"center":[], "file_id":[]} #前面的初始化有问题
更新聚中心: self.update_cluster_center()
def update_cluster_center(self):
    for cluster_id, cluster in self.cluster_dict.items():
                                                  text_ifidf=[]
        for file_id in cluster['file_id']:
text_ifidf.append
            self.document_itidf[str(file_id)] = tfidf_list
    arr_text_ifidf=np.array(text_ifidf)
    avg_cluster_center = np.mean(arr_text_ifidf, axis=0).
    print(avg_cluster_center.shape) 验证是2333维

    self.cluster_dict[cluster_id]['center'] = avg_cluster_center,
更新完后, 在 while count<num 之后清空 self.cluster_dict
    for i in range(1, k+1):
        self.cluster_dict[i]['file_id'] = []
def save_text(self):
    for cluster_id, cluster in self.cluster_dict.items():
        if os.path.exists(str(cluster_id)):
            os.mkdir(str(cluster_id)): #创建文件夹
        for file_id in cluster['cluster']:
            file_path = os.path.join(str(cluster_id), '{}.text'.format(file_id))
            with open(file_path, 'w', encoding='utf-8') as f:
```

```
f.write(self.document[file_id])
```

主题模型 分布问题 .(用的较多).

现在文本分类一般用深度学习，不用机器学习。

word2vec有上下文关系的，(在pyhanlp下static下data中word2vec文件

nlu：自然语言理解
- 词法分析
  - 新词发现 (互信息 信息熵)。
  - 形态分析 → 拼写校正
  - 词性标注
- 句法分析
  - 成分句法分析
  - 依存句法分析 (从 从属词指向支配词)
- 语义分析
  - 词义消歧 (elmo解决一词多义)
  - 语义角色标注
  - 词汇 句子 段落的向量化表示
- 文本分类/情感分析
- 文本匹配
- (难) 信息抽取
  - 实体抽取 (实体识别 crf)
  - 关系抽取
  - 事件抽取
- 阅读理解 (bert)

应用
- 机器翻译
- 搜索推荐 (工资高)
- 知识图谱
- 对话系统
- 问答系统

NLG：
- 文本摘要
- 机器翻译