数据挖掘 Homework#2 实验报告

计63 肖朝军 2016011302

一、数据预处理和可视化

1）解析xml文档，并建立数据框对象

```
# （1）解析xml文档，并建立数据框对象
dir_path = '../../HW2/nyt_corpus/samples_500/'
filenames = list.files(path = dir_path)

news_data = data.frame()
all_class = c()
for (i in 1:length(filenames)){
    filename = filenames[i]
    file = xmlParse(file = paste(dir_path, filename, sep = ""))
    metaName = xpathSApply(file, '//meta', xmlGetAttr, 'name')
    metaContent = xpathSApply(file, '//meta', xmlGetAttr, 'content')

    full_text = xpathSApply(file, "//block[@class='full_text']", xmlValue)
    lead_paragraph = xpathSApply(file, "//block[@class='lead_paragraph']", xmlValue)

    classes = xpathSApply(file, '//classifier', xmlValue)

    if (length(full_text) == 0){
        next
    }
    news_data[filename, 'full_text'] = full_text[1]
    if (length(lead_paragraph) != 0){
        news_data[filename, 'lead_paragraph'] = lead_paragraph[1]
    }
    else{
        news_data[filename, 'lead_paragraph'] = NA
    }

    for (i in 1:length(metaName)){
        news_data[filename, metaName[i]] = metaContent[i]
    }

    classes = unique(str_extract(classes, '((?<=Top/News/).*?((?=/)))|((?
<=Top/Features/).*?((?=/)))'))
    for (c in classes){
        if (!is.na(c)){
            news_data[filename, c] = 1
            all_class[length(all_class) + 1] = c
        }
    }
```

```
}
print(colnames(news_data))
all_class = levels(factor(all_class))
print(all_class)
```

这里主要运用了xml库来解析新闻文档，其中xpathSApply函数，可以通过对应的限制条件找到xml文档中对应的节点，获取它的属性值、文本值。在这里，我主要抽取了文档中的 `meta` 节点， `classifier` 节点，包含有 `full_text` 和 `lead_paragraph` 的 `block`。

其中， `classifier` 节点的解析利用了正则表达式，按照作业中的要求提取了新闻的类别。并且，为了方便后续的对于新闻类别的统计，我用 `all_class` 存储了所有的类别。

根据输出结果，数据框对象总共有34个属性值。如下：

```
 [1] "full_text"                "lead_paragraph"
 [3] "publication_day_of_month" "publication_month"
 [5] "publication_year"         "publication_day_of_week"
 [7] "dsk"                      "print_page_number"
 [9] "print_section"            "print_column"
[11] "online_sections"          "U.S."
[13] "Travel"                   "World"
[15] "Sports"                   "Arts"
[17] "Theater"                  "Style"
[19] "Science"                  "Health"
[21] "Washington"               "Books"
[23] "New York and Region"      "Movies"
[25] "Business"                 "banner"
[27] "correction_date"          "feature_page"
[29] "slug"                     "column_name"
[31] "series_name"              "Dining and Wine"
[33] "alternate_url"            "Magazine"
```

其中，代表新闻类别的属性值有16种，如下：

```
 [1] "Arts"               "Books"              "Business"
 [4] "Dining and Wine"    "Health"             "Magazine"
 [7] "Movies"             "New York and Region" "Science"
[10] "Sports"             "Style"              "Theater"
[13] "Travel"             "U.S."               "Washington"
[16] "World"
```

2）预处理文本

```
# （2）文本预处理
pre_process <- function(corpus){
    corpus = tm_map(corpus, stripWhitespace) # 消除空格
    corpus = tm_map(corpus, removePunctuation) # 去除标点符号
    corpus = tm_map(corpus, content_transformer(tolower)) #小写
    corpus = tm_map(corpus, removeWords, stopwords('en')) #停用词
    corpus = tm_map(corpus, removeNumbers)# 数字
    corpus = tm_map(corpus, stemDocument)# 词干化
    return (corpus)
}

corpus = VCorpus(VectorSource(news_data$full_text))
corpus = pre_process(corpus)
```

在这个步骤中，我主要使用了 `tm` 库中的 `tm_map` 函数，该库中提供了很多现成的函数来进行文本预处理。

3）转成BagOfWord向量

```
# （3）文章向量转化
bag_of_words = DocumentTermMatrix(corpus)

bag_of_words = as.matrix(bag_of_words)
print(bag_of_words[1:3, 100:120])
```

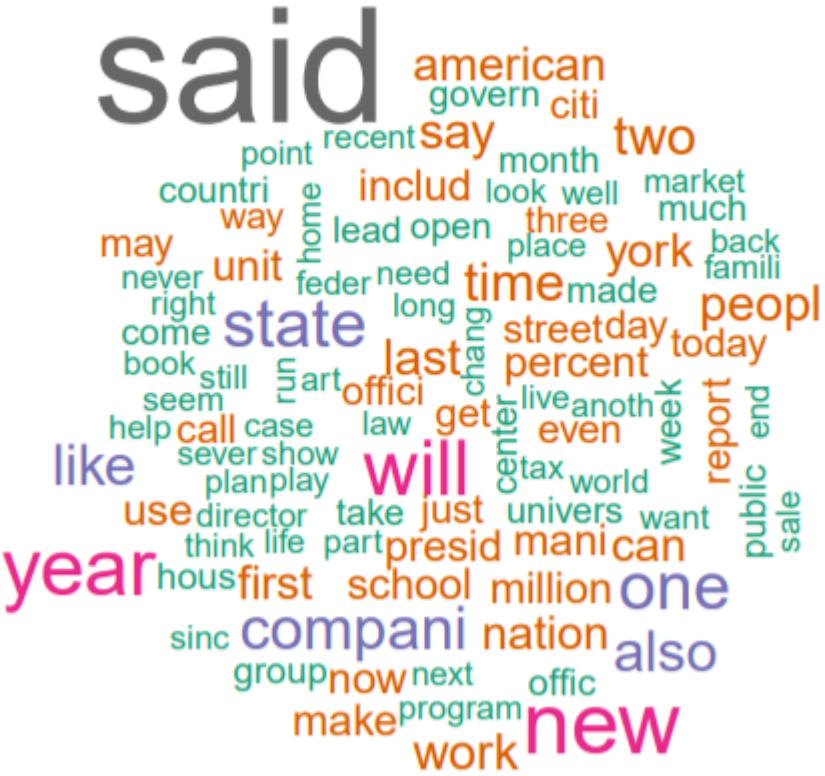
利用了 `DocumentTermMatrix` 函数，将预处理后的文本，构建了文档-单词矩阵，BOW忽略词语在文本中的位置，矩阵中每个元素都代表了某个单词在文档中出现的次数。从输出的矩阵的部分可以看出，这个矩阵显然是非常稀疏的。

```
    Terms
Docs accumul accur accuraci accuracyth accus accustom ace acerb ach achev achiev
   1       0     0        0          0     0        0   0     0   0     0     0
   2       0     0        0          0     0        0   0     0   0     0     0
   3       0     0        0          0     0        0   0     0   0     0     0
```

4）生成词云

```
# （4）词云生成
word_count = sort(colSums(bag_of_words), decreasing = TRUE)
most_100 = word_count[1:100]
wordcloud(names(most_100), most_100, colors = brewer.pal(8, "Dark2"))
```

在这里，通过对第3步中获得的BOW向量进行列求和操作，求得每一个词语的总出现次数。利用R语言自带的sort函数，进行从高到低排序，取了前100的词语，利用了 `wordcloud` 库生成了词云。其中词云图片如下：

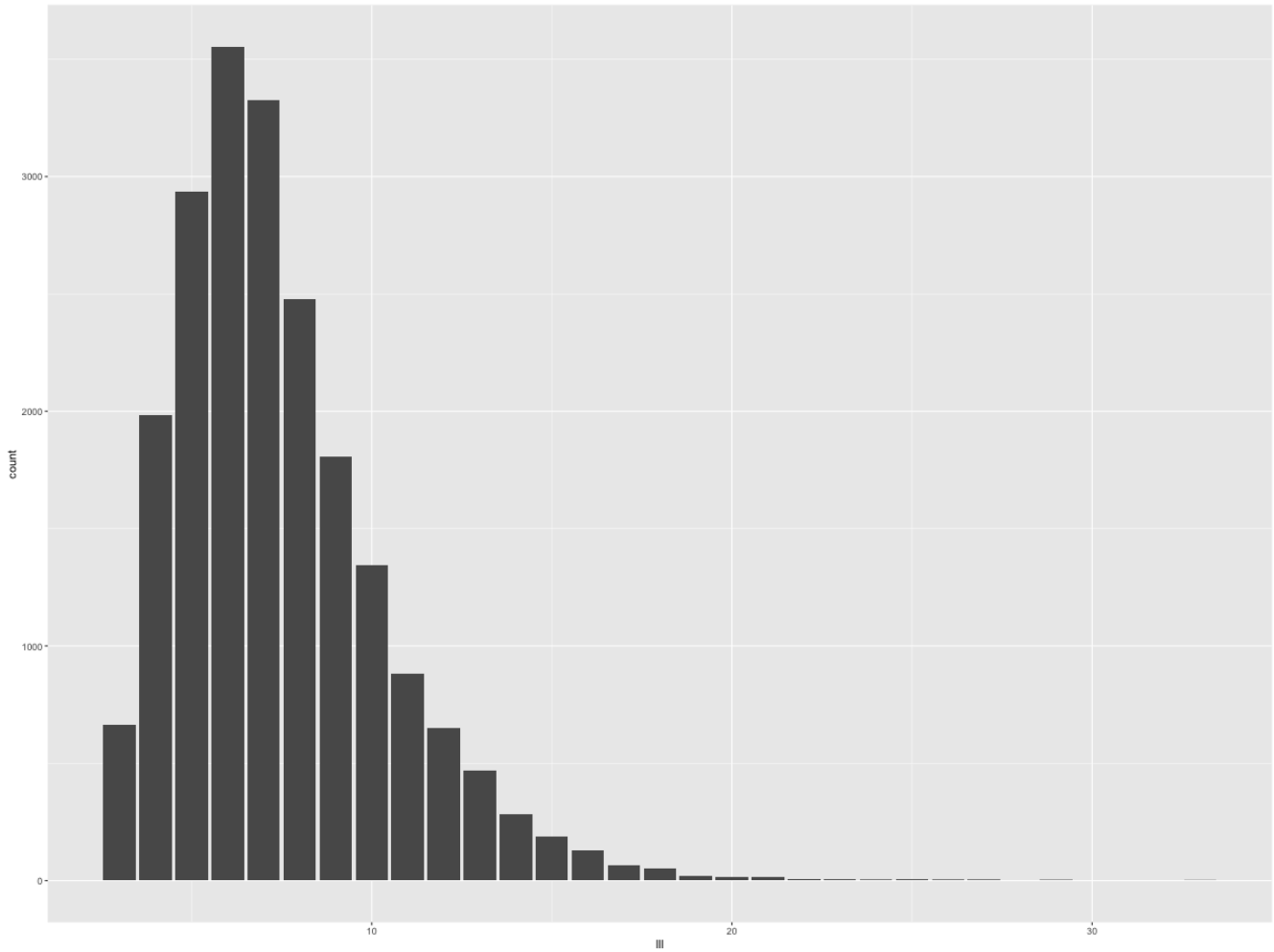


可以看出，在文档中出现频率最高的词语为said，其中new、year、will等词语也同样高频出现。

5）单词长度分布直方图

```
# （5）单词长度分布
png(filename = "p.png", width = 1200,height = 900)
leng = sapply(names(word_count), nchar)
leng_frame = data.frame(lll = leng)
ggplot(leng_frame, aes(lll)) + geom_bar()

dev.off()
```

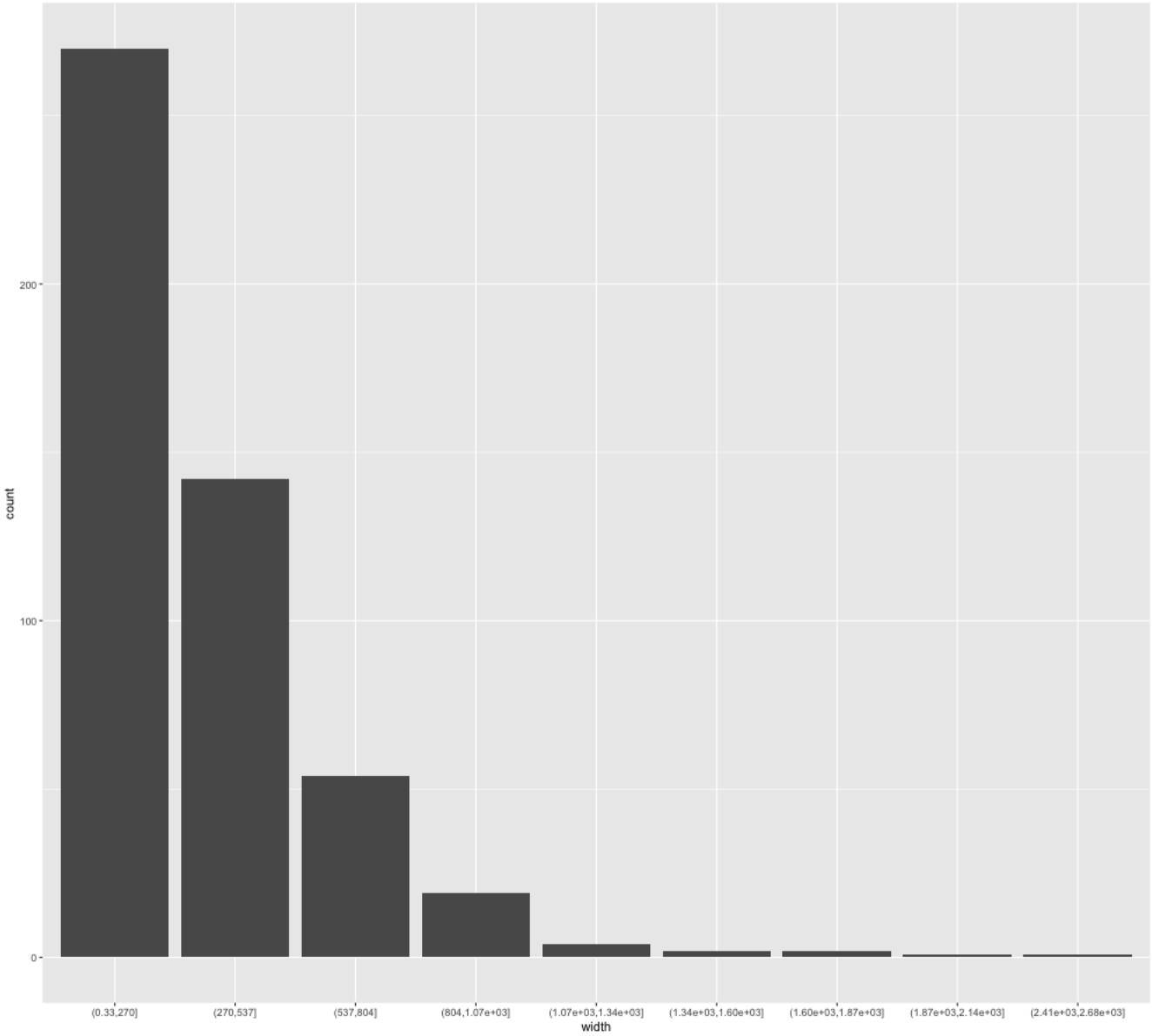
运行代码得到以下直方图，可以看出，单词长度主要集中在5-9这样一个长度，只有极少部分单词长度超过了20.



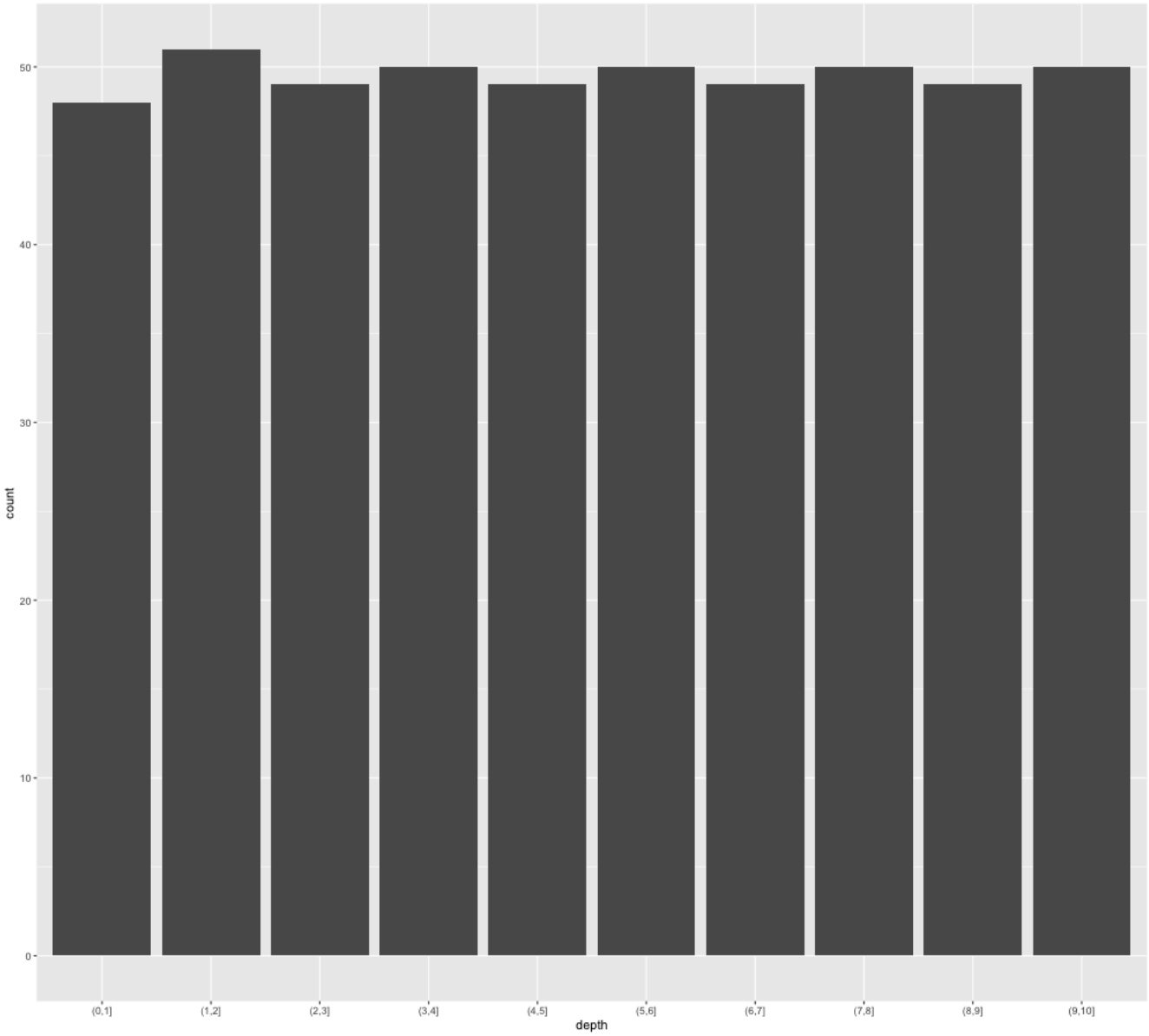
6）新闻长度分布直方图

```
article_count = sapply(corpus, function(x) return(length( str_split(x$content, pattern = '
')[[1]])))

# 等深分箱、等宽分箱
width = cut(article_count, 10)

cut_depth = function(x, n){
  cut(rank(x)/length(x)*n,breaks = 0:n)
}
depth = cut_depth(article_count, 10)

data_length = data.frame(depth_cut = depth, width_cut = width)
#等深分箱
png(filename = "depth.png", width = 1000,height = 900)
ggplot(data_length, aes(depth_cut)) + geom_bar() + xlab("depth")
dev.off()
#等宽分箱
png(filename = "width.png", width = 1000,height = 900)
ggplot(data_length, aes(width_cut)) + geom_bar() + xlab("width")
dev.off()
```

先统计每一篇文章按照空格分隔后的长度，作为其单词数量。



等宽分箱的条件下，画图只展示了9个长条块，可以发现有一个区间没有新闻，所以只展示了9个。从图中可以发现大多数文章都是相对比较短的。

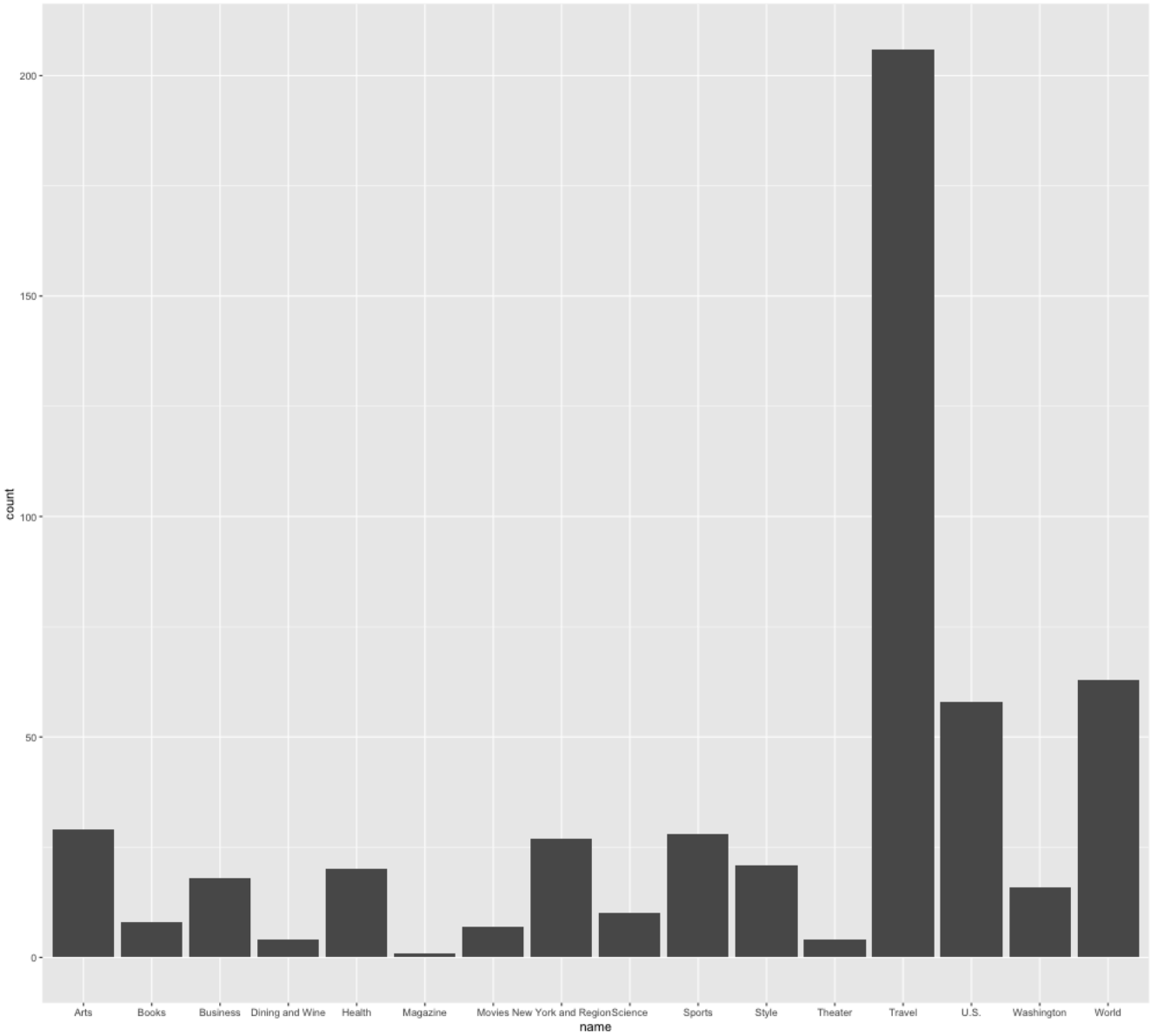


等深分箱，每一个区间的文章数量基本一致。等深情况下，先计算每一个新闻在排序中的rank，然后将rank归到1-10类。

7）新闻类别分布直方图

```
class_count = data.frame(name = all_class)
for (c in all_class){
    class_count[class_count$name == c, 'count'] = length(news_data[, c]) -
sum(is.na(news_data[, c]))
}
png(filename = "class_count.png", width = 1000,height = 900)
ggplot(class_count, aes(x=name, y=count)) + geom_bar(stat = "identity")
dev.off()
```

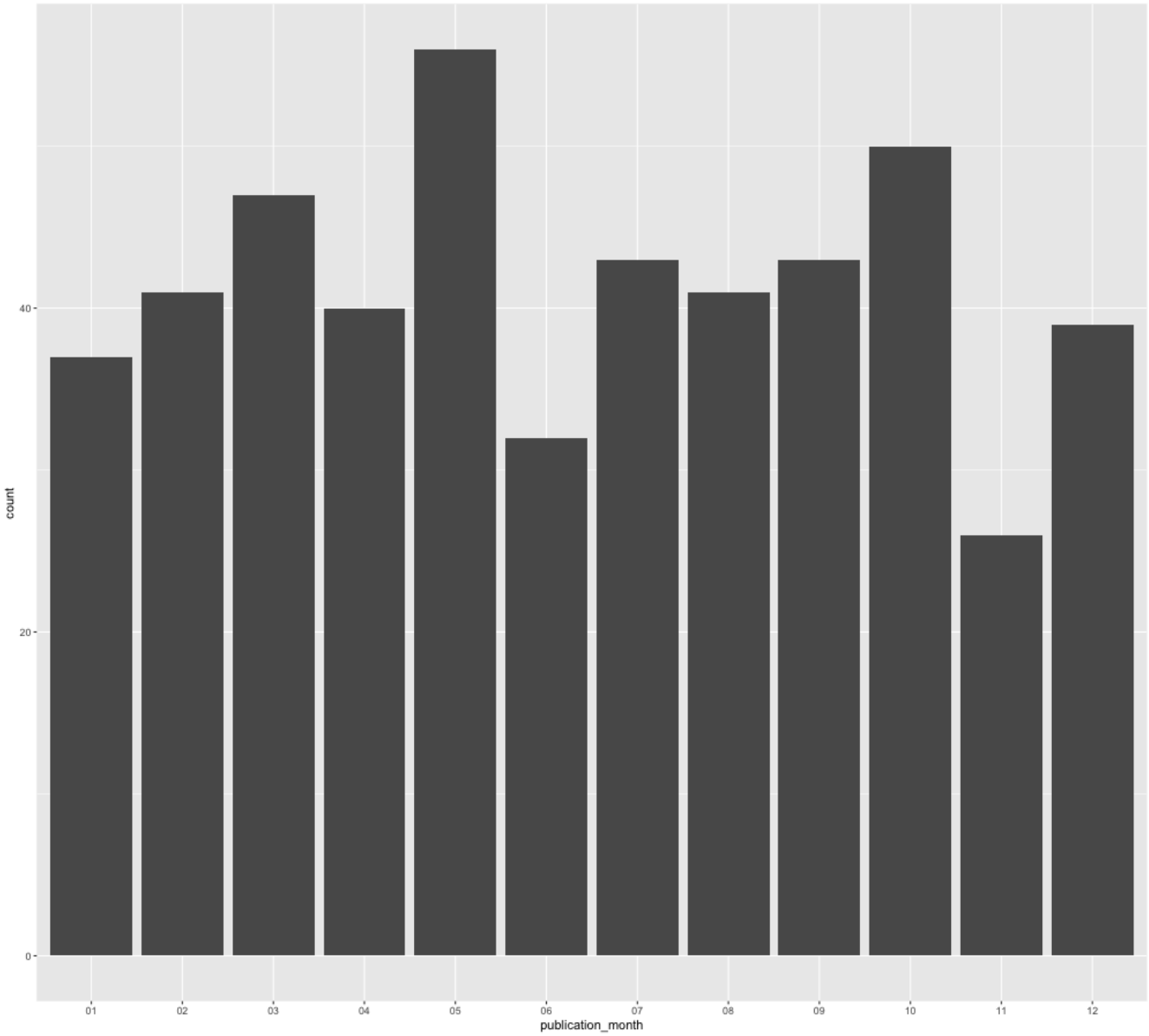
利用在建立数据的时候统计的16个类别属性，先计算每个类别的新闻数量，再将其画成直方图。可以发现，`Travel`类具有最多的新闻，其中关于`Magazine`的内容是最少的。



8）新闻时间分布直方图

```
for (i in 1:9){
  news_data$publication_month[news_data$publication_month == as.character(i)] = paste("0",
as.character(i), sep = "")
}
png(filename = "month_count.png", width = 1000,height = 900)
ggplot(news_data, aes(x=publication_month)) + geom_bar()
dev.off()
```

这个统计与之前的没有太大的区别，从图中可以看出5月份是新闻最多的一个月份，而6月与11月新闻数量明显少于其他的月份。



二、高维向量可视化

在这里，需要对100个词向量的100维降到2维平面查看。作业中采用了两种将维方法：

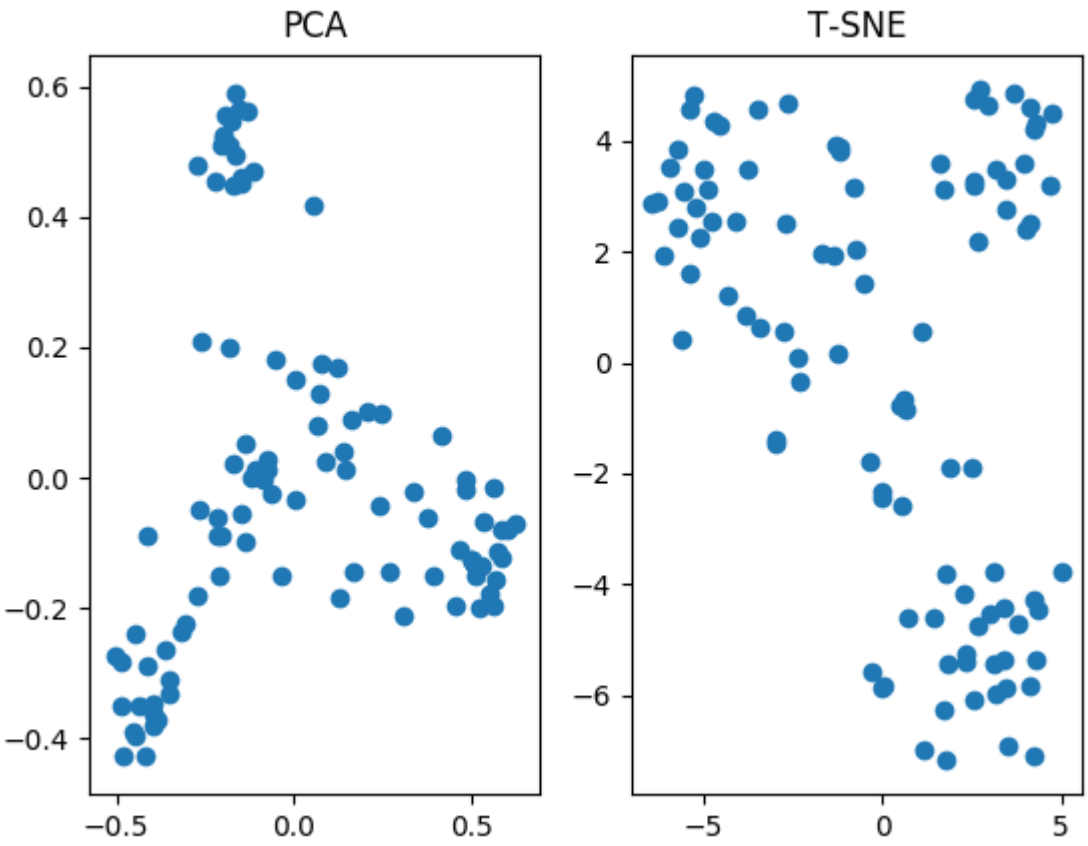
- PCA
- t-SNE

代码如下：

```
import sklearn
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

# 读取词向量，利用numpy存储
def readvec(path):
    fin = open(path, 'r')
```

```
    wordvecs = {}
    for line in fin:
        word, vec = line.split('\t')
        vec = np.array([float(d) for d in vec.split(' ')])
        wordvecs[word] = vec
    return wordvecs

if __name__ == '__main__':
    path = '../../HW2/100_word_vector.txt'
    wordvecs = readvec(path)

    X = [wordvecs[key] for key in wordvecs]

    plt.figure()
    p1 = plt.subplot(121)
    # 利用PCA进行降维
    p1.set_title('PCA')
    pca = PCA(n_components=2)
    X_pca = pca.fit_transform(X)
    plt.scatter(X_pca[:, 0], X_pca[:, 1])

    #利用t-sne进行降维
    p2 = plt.subplot(122)
    p2.set_title('T-SNE')
    tsne = TSNE(n_components=2, learning_rate = 100)
    X_tsne = tsne.fit_transform(X)
    plt.scatter(X_tsne[:, 0], X_tsne[:, 1])

    plt.savefig('PCA_TSNE.png')
```

降维后散点图如下：



从图上可以发现，降维之后PCA和T-SNE两种方法都让词向量有聚类的趋势，图上可以较清晰看出词向量大致可以聚成3类。相比于PCA降维，T-SNE的词向量更加分散（分散程度与learning_rate有很大的关系），但是不同类之间间隔明显，有着更好的降维效果。