有监督学习-朴素贝叶斯

Supervised Learning Naive Bayes Classification

- 贝叶斯定理
- 朴素贝叶斯
- 实践

贝叶斯定理

贝叶斯分类器

- 是对属性集和类变量的概率关系建模的方法
- 常用于垃圾邮件检测、文本分类中
- 考虑一个概率问题：
  - 两队间的足球比赛，队0与对1。假设65%的比赛队0胜出，剩余的比赛队1胜出。队0获胜的比赛只有30%是在对1的主场，而队1取胜的比赛中75%是主场获胜。如果下一场比赛在队1的主场进行，哪一个队伍有可能获胜呢?

贝叶斯分类器

- 概率论中的基本定义
- 假设X，Y是一对随机变量，
  - 联合概率P(X=x, Y=y)是指X=x, Y=y 同时发生的概率
  - 条件概率是指一个随机变量在另一个随机变量已知的情况下取某一值的概率。例如P(Y=y|X=x)

贝叶斯分类器

- 联合概率与条件概率的关系　$P(X,Y) = P(Y|X) * P(X) = P(X|Y) * P(Y)$
- 最后两个公式得到的公式，称为贝叶斯定理

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

- X代表东道主，Y代表比赛的胜利者
  - 队0取胜的概率P(Y=0)=0.65
  - 队1取胜的概率P(Y=1) = 1- P(Y=0)=0.35
  - 队1取胜时作为东道主的概率是P(X=1|Y=1)=0.75
  - 队0取胜时队1作为东道主的概率是P(X=1|Y=0)=0.3
  - 我们要求P(Y=1|X=1)

贝叶斯分类器

$$P(Y=1|X=1) = \frac{P(X=1|Y=1)\,P(Y=1)}{P(X=1)}$$

$$= \frac{P(X=1|Y=1)\,P(Y=1)}{P(X=1, Y=1) + P(X=1, Y=0)}$$

$$= \frac{P(X=1|Y=1)\cdot P(Y=1)}{P(X=1|Y=1)\cdot P(Y=1) + P(X=1|Y=0)\cdot P(Y=0)}$$

$$= \frac{0.75 \times 0.35}{0.75 \times 0.35 + 0.3 \times 0.65}$$

$$= 0.5738$$

贝叶斯分类器

- 从统计学角度将问题形式化，X为数据集，Y表示类变量
- 用P(Y|X)捕捉二者的关系，这个条件概率又称为Y的后验概率
- 准确估计类标号和属性值的每一种组合的后验概率非常困难，因为即使类的数目不是很大，仍然需要很大的训练集，因此贝叶斯定理很有用，因为它允许我们用先验概率P(Y)、类的条件概率P(X|Y)和P(X)来表示后验概率

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

朴素贝叶斯

- 朴素贝叶斯分类模型采用了基于属性条件是相互独立的假设，认为每个属性的互相没有影响

- 基于条件独立的假设，类别c的条件概率可以写为

$$P(Y|X) = \frac{P(Y)P(X|Y)}{P(X)} = \frac{P(Y)}{P(X)}\prod_{i=1}^{d} P(X_i|Y)$$

朴素贝叶斯

- 对于所有类别来说P(X)是相同的，所以类别Y的条件概率可以改写为

$$P(Y|X) = argmaxP(Y)\prod_{i=1}^{d} P(X_i|Y)$$

- 这就是朴素贝叶斯的表达式

朴素贝叶斯

- 朴素贝叶斯分类器的训练过程是基于训练集T估计类别y的先验概率P(Y)和每个属性的条件概率 $P(X_i|Y)$
  - 令Ty 表示T中第y类样本的集合，则 $P(y)=\frac{|T_y|}{|T|}$
  - 对于离散属性，T_{y,x_i} 表示属于类别y的数据，切在第i个属性上取值为 x_i 的样本组成的集合，则条件概率 $P(x_i|y)$ 为

$$P(x_i|y)=\frac{|T_{y,x_i}|}{|T|}$$

朴素贝叶斯

- 对于连续属性，我们认为 $p(x_i|y) \sim N(\mu_i, \sigma_i^2)$，$\mu_i, \sigma_i^2$ 分别为属性xi上的均值与方差

$$p(x_i|y) = \frac{1}{\sqrt{2\pi}\sigma_{c,i}} exp(\frac{(x-\mu_{c,i})^2}{\sigma_{c,i}^2})$$

朴素贝叶斯

- 某一个文本数据集，共有2类文章，如下表。

包含java	包含python	疾病
no	yes	java类文章
yes	no	java类文章
no	yes	python类文章
yes	yes	java类文章
no	no	python类文章
no	yes	python类文章

朴素贝叶斯

- 又来了第七篇文章，是包含java和python关键字的文章，求属于java类文章的关键字

- P(类型=java类|包含java=yes，包含python=yes)

=P(包含java=yes，包含python=yes|类型=java类) * P(类型=java类) /P(包含java=yes，包含python=yes）

=P(包含java=yes|类型=java类)*P(包含python=yes|类型=java类)*P(类型=java类))/P(包含java=yes，包含python=yes）

= 0.66 x 0.66 x 0.5 / 0.66 = 0.33

朴素贝叶斯-拉普拉斯平滑

- 假设某一个文本数据集，共有2类文章，如下表。

包含java	包含python	疾病
no	no	java类文章
yes	no	java类文章
no	no	python类文章
yes	no	java类文章
no	no	python类文章
no	no	python类文章

朴素贝叶斯

- 又来了第七篇文章，是包含java和python关键字的文章，求属于python类文章的关键字

- P(类型=python类|包含java=yes，包含python=yes)

=P(包含java=yes，包含python=yes|类型=python类) * P(python类) /P(包含java=yes，包含python=yes）

=P(包含java=yes|类型=python类)*P(包含python=yes|类型=python类)*P(类型=java类))/P(包含java=yes，包含python=yes）

- 数据集中并不存在是python类文章但又包含python关键字的数据，所以P(包含python=yes|类型=python类) =0，导致整个概率为0

朴素贝叶斯-拉普拉斯平滑

- 对朴素贝叶公式展开后，很有可能某一项在的概率为0

$$P(Y|X) = argmaxP(Y)\prod_{i=1}^{d} P(X_i|Y)$$

- 为了解决这个问题，我们引入Laplace校准，它的思想非常简单，就是对每个类别下所有划分的计数加1，这样如果训练样本集数量充分大时，并不会对结果产生影响，并且解决了上述频率为0的尴尬局面。

朴素贝叶斯-拉普拉斯平滑

- 加入拉普拉斯平滑的计算方式为
- N为类别的个数, Sxi为等于xi的特征数，λ 为拉普拉斯系数

$$P(y) = \frac{|T_y| + 1}{|T| + N}$$

$$P(x_i|y) = \frac{|T_{y,x_i}| + \lambda}{|T| + S_{xi}\lambda}$$

实践

MultinomialNB

- 多项式朴素贝叶斯适用于离散变量较多的数据集

```
class sklearn.naive_bayes.MultinomialNB(
    alpha=1.0, # 拉普拉斯系数
    fit_prior=True, # 是否计算先验概率，如果为False则所有先验概率一样
    class_prior=None  # 是否指定类的先验概率
    )
```

文档自动分类

- 使用朴素贝叶斯模型构建一个文档分类模型，自动将文章分为属于哪一个新组织机构
  - 类别(新闻组织)：
    - alt.atheism
    - comp.graphics
    - comp.os.ms-windows.misc
    - comp.sys.ibm.pc.hardware
    - comp.sys.mac.hardware
- 每个类别中有1000篇文章
- 要求：
  - 训练集和测试集各占75%与25%
  - 输出每一个类的准确与召回

文档自动分类

朴素贝叶斯文档自动分类模型

使用朴素贝叶斯训练一个文档分类模型，当给定一篇文章的时候，将文章自动分类为属于哪一个新闻组织结构的文章。 数据中共有5种新闻组织机构（5个类别），每个类别中有1000篇张文章。

1. 导入包

[1]:
```
import os
import string
import numpy as np
import matplotlib.pyplot as plt
from sklearn import model_selection
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report
```

文档自动分类

2. 加载数据并构建训练集与测试集

[3]:
```
data_path = './data/5_newsgroups'
# X存储初始化的数据，以(filename, text)的形式存储
X = []
# 将文章对应的类别存储在Y中
Y = []
for category in os.listdir(data_path):
    for document in os.listdir(data_path + os.sep + category):
        with open(data_path + os.sep + category + os.sep + document, 'rb') as f:
            X.append((document,str(f.read())))
            Y.append(category)
print('Data load done')

X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y, test_size=0.25, random_state=0)
```

```
Data load done
```

[4]:
```
# 一些常见不会对分类有影响的单词
stopwords = ['a', 'about', 'above', 'across', 'after', 'afterwards', 'again', 'against', 'all', 'almost', 'alone',
             'along', 'already', 'also', 'although', 'always', 'am', 'among', 'amongst', 'amoungst', 'amount',
             'an', 'and', 'another', 'any', 'anyhow', 'anyone', 'anything', 'anyway', 'anywhere', 'are', 'around',
             'as', 'at', 'back', 'be', 'became', 'because', 'become', 'becomes', 'becoming', 'been', 'before',
             'beforehand', 'behind', 'being', 'below', 'beside', 'besides', 'between', 'beyond', 'bill', 'both',
             'bottom', 'but', 'by', 'call', 'can', 'cannot', 'cant', 'co', 'con', 'could', 'couldnt', 'cry', 'de',
             'describe', 'detail', 'did', 'do', 'does', 'doing', 'don', 'done', 'down', 'due', 'during', 'each', 'eg',
             'eight', 'either', 'eleven', 'else', 'elsewhere', 'empty', 'enough', 'etc', 'even', 'ever', 'every', 'everyone',
             'everything', 'everywhere', 'except', 'few', 'fifteen', 'fify', 'fill', 'find', 'fire', 'first', 'five', 'for',
             'former', 'formerly', 'forty', 'found', 'four', 'from', 'front', 'full', 'further', 'get', 'give', 'go', 'had',
             'has', 'hasnt', 'have', 'having', 'he', 'hence', 'her', 'here', 'hereafter', 'hereby', 'herein', 'hereupon',
             'hers', 'herself', 'him', 'himself', 'his', 'how', 'however', 'hundred', 'i', 'ie', 'if', 'in', 'inc', 'indeed',
             'interest', 'into', 'is', 'it', 'its', 'itself', 'just', 'keep', 'last', 'latter', 'latterly', 'least', 'less',
             'ltd', 'made', 'many', 'may', 'me', 'meanwhile', 'might', 'mill', 'mine', 'more', 'moreover', 'most', 'mostly',
             'move', 'much', 'must', 'my', 'myself', 'name', 'namely', 'neither', 'never', 'nevertheless', 'next', 'nine',
             'no', 'nobody', 'none', 'noone', 'nor', 'not', 'nothing', 'now', 'nowhere', 'of', 'off', 'often', 'on', 'once',
             'one', 'only', 'onto', 'or', 'other', 'others', 'otherwise', 'our', 'ours', 'ourselves', 'out', 'over', 'own',
```

文档自动分类

[5]:
```
# 根据数据集中给定的文章构建字典
# key为数据集中出现的单词，value为出现单词的数目
vocab = {}
for i in range(len(X_train)):
    word_list = []
    for word in X_train[i][1].split():
        # 删除字符串（单词）前后的标点符号
        word_new  = word.strip(string.punctuation).lower()
        # 将两个字母以上的单词并且不在stopwords列表中的单词加入到字典中
        if (len(word_new)>2)  and (word_new not in stopwords):
            if word_new in vocab:
                vocab[word_new]+=1
            else:
                vocab[word_new]=1
```

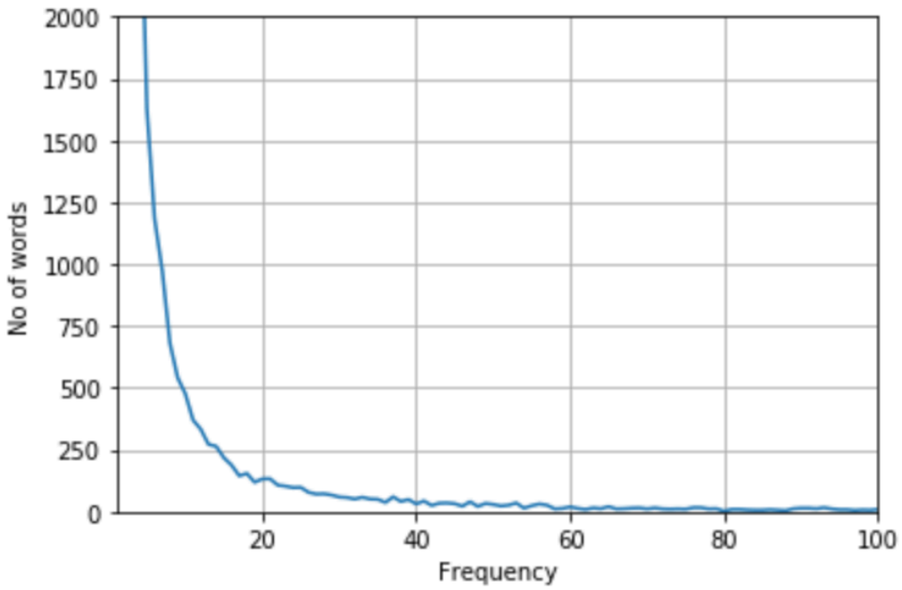
3. 数据预处理

删除出现频次很低的单词。

[14]:
```
# 打印单词在指定频次下出现的次数
# x轴：单词出现的频次，y轴，在某一个频次下，共有多少个单词
num_words = [0 for i in range(max(vocab.values()) + 1)]

freq = [i for i in range(max(vocab.values()) + 1)]
for key in vocab:
    num_words[vocab[key]] += 1
plt.plot(freq, num_words)
plt.axis([1, 100, 0, 2000])
plt.xlabel("Frequency")
plt.ylabel("No of words")
plt.grid()
plt.show()
```

文档自动分类



15]:
```python
cutoff_freq = 60
# For deciding cutoff frequency
num_words_above_cutoff = len(vocab) - sum(num_words[0: cutoff_freq])
print("Number of words with frequency higher than cutoff frequency({}) :".format(cutoff_freq),num_words_above_cutoff)
```

```
Number of words with frequency higher than cutoff frequency(60) : 1029
```

16]:
```python
# 将出现次数高于60的单词作为特征
features = []
for key in vocab:
    if vocab[key] >= cutoff_freq:
        features.append(key)
```

文档自动分类

重新构建训练集和测试集，训练集是一个len(X_train),len(features)的二维数组。
数值对应着每个单词在某一篇文章中出现的次数

[17]:
```
X_train_dataset = np.zeros((len(X_train),len(features)))
print('Train set is being constructed.')
for i in range(len(X_train)):
    if i%1000 == 0:
        print('.', end="")
    word_list = [ word.strip(string.punctuation).lower() for word in X_train[i][1].split()]
    for word in word_list:
        if word in features:
            X_train_dataset[i][features.index(word)] += 1
print('Train set', X_train_dataset.shape)
print('Done.')
```

```
Train set is being constructed.
....Train set (3750, 1029)
Done.
```

[18]:
```
X_test_dataset = np.zeros((len(X_test),len(features)))
print('Test set is being constructed.')

# This can take some time to complete
for i in range(len(X_test)):
    if i%1000 == 0:
        print('.', end="")
    # print(i) # Uncomment to see progress
    word_list = [ word.strip(string.punctuation).lower() for word in X_test[i][1].split()]
    for word in word_list:
        if word in features:
            X_test_dataset[i][features.index(word)] += 1
print('Test set', X_test_dataset.shape)
print('Test set construct. Done.')
```

文档自动分类

4. 训练与评估模型

提示：classification_report函数产生的support的解释如下，
sklearn官方文档的解释是“The support is the number of occurrences of each class in y_true.”
class I的suppport是k，意思就是说该测试集中有k个样本的真实分类为class I.
表格中class alt.atheism support = 238就是说，测试集里有2238个样本的真实标签是alt.atheism

[20]:
```
# Using sklearn's Multinomial Naive Bayes
clf = MultinomialNB()
clf.fit(X_train_dataset,Y_train)
Y_test_pred = clf.predict(X_test_dataset)
sklearn_score_train = clf.score(X_train_dataset,Y_train)
print("Sklearn's score on training data :",sklearn_score_train)
sklearn_score_test = clf.score(X_test_dataset,Y_test)
print("Sklearn's score on testing data :",sklearn_score_test)
print("Classification report for testing data :-")
print(classification_report(Y_test, Y_test_pred))
```

```
Sklearn's score on training data : 0.924
Sklearn's score on testing data : 0.9056
Classification report for testing data :-
                          precision    recall  f1-score   support

             alt.atheism       0.97      0.98      0.97       238
           comp.graphics       0.93      0.85      0.89       244
 comp.os.ms-windows.misc       0.83      0.92      0.87       252
comp.sys.ibm.pc.hardware       0.91      0.84      0.87       272
   comp.sys.mac.hardware       0.91      0.95      0.93       244

               micro avg       0.91      0.91      0.91      1250
               macro avg       0.91      0.91      0.91      1250
            weighted avg       0.91      0.91      0.91      1250
```