机器学习工程师纳米学位

毕业项目报告

项目题目

自然语言处理-文档归类

一、问题的定义

项目概述

自然语言处理是计算机科学领域与人工智能领域中的一个重要方向。所谓的自然语言处理，是用计算机通过可计算的方法对自然语言的各级语言单位（字、词、语句、篇章等等）进行转换、传输、存储、分析等加工处理的科学。自然语言处理技术是一门与语言学、计算机科学、数学、心理学、信息论、声学相联系的交叉性学科，与自然科学和社会科学的许多主要学科都有着千丝万缕的联系，其中，又与语言学、计算机科学和数学的关系最为密切。在更加细微的层面上，与自然语言处理技术密切相关的学科有计算语言学、智能化人机接口、自然语言理解等等。

自然语言处理的关键是将文本内容（字、词、语句、篇章等等）进行转换，目的是将其转换成计算机模型和算法能够识别和作为数据输入的形式。该项目的出发点就是研究这个问题。

使用到的数据集为 20newsgroups。20newsgroups 数据集是用于文本分类、文本挖据和信息检索研究的国际标准数据集之一。数据集收集了大约 20,000 左右的新闻组文档，均匀分为 20 个不同主题的新闻组集合。辅助数据集是 text8 数据包。text8 来源于 enwiki8，enwiki8 是从 wikipedia 上得到的前 100，000，000 个字符；而 text8 就是把这些字符当中各种奇怪的符号，非英文字符全都去掉，再把大写字符转化成小写字符，把数字转化成对应的英语单词之后，得到的。

问题陈述

计算机系统识别和计算的内容是以数字作为基础的，文本内容不能作为计算模型的输入，不能被模型识别。因此，我们的问题是：文本应该用什么样的形式表示？才能有利于模型识别和计算。

首先使用词袋子模型表示文本。词袋模型可以看成是独热编码的一种扩展，它为每个单词设置一个特征值。也就是将一个文本文件分成单词的集合，建立词典，每篇文档表示成特征词的频率向量或者加权词频 TF-IDF 向量，这样可以得到每个文档的特征表。此时就可以使用监督学习分类模型进行训练和评估。在该模型下，每个词是独立存在的，没有上下文的关联性。

然后使用词向量模型表示文本。利用文本数据对词向量进行训练，将每个单词表示成向量形式。词向量的基本思想是把自然语言中的每一个词，表示成一个统一意义统一维度的短向量。使用词向量的关键是怎样用文档中每个词的向量来表达整个文档。方法是：将文档处理成单词索引序列形式的词典，然后文档就可以使用词典中每个单词的序号进行表示，文档就转变成了矩阵形式。此时文档就可以作为分类器的输入数据了。然后根据词向量和刚刚得到的索引字典构建嵌入层 Embedding 的权重矩阵。再将文档类别标签进行独热编码处理，此时就可以使用深度学习模型进行训练和评估了。

期待的结果是：训练好的模型，可以根据输入的新闻文档准确的预测该新闻属于哪个类别。

评价指标

选用了两个评估指标，log_loss 损失函数是主要评估指标，accuracy_score 准确率分数是辅助评估指标，下面分别对其进行说明：

主要评估指标为：log_loss 损失函数。损失函数越小，模型的鲁棒性就越好。

计算方法：在二分类的时候，真实标签集合为：$y\in\{0,1\}$，而分类器预测得到的概率分布：$p=Pr(y=1)$

那么，每一个样本的对数损失就是在给定真实样本标签的条件下，分类器的负对数似然函数，如下所示：

$$L_{\log}(y,p) = -\log \Pr(y|p) = -(y\log(p) + (1-y)\log(1-p))$$

当某个样本的真实标签 y=1 时，Loss=-log(p)，所以分类器的预测概率 p=Pr(y=1)的概率越大，则损失越小；如果 p=Pr(y=1)的概率越小，则分类器损失就越大。对于真实标签 y=0，Loss=-log(1-p)，所以分类器的预测概率 p=Pr(y=1)的概率越大，则损失越大。

在多元分类的时候，假定有 K 个类，则类标签集合就是 labels={1,2,3,…,K}。如果第 i 个样本的类标签是 k 的话，就记作$y_{i,k}$=1。采用 one-hot 记法，每个样本的真实标签就是一个 one-hot 向量，其中只有一个地方标记为 1。这样 N 个样本的真实类标签就是一个 N 行 K 列的矩阵 Y。分类器对 N 个样本的每一个样本都会预测出它属于每个类的概率，这样预测概率矩阵 P 就是 N 行 K 列的$p_{i,k}$=Pr($t_{i,k}=1$)，整个样本集合上分类器的对数损失就可以如下定义：

$$L_{\log}(Y,P) = -\log \Pr(Y|P) = -\frac{1}{N}\sum_{i=0}^{N-1}\sum_{k=0}^{K-1} y_{i,k}\log p_{i,k}$$

合理性：对数损失也被成为交叉熵损失，是定义在概率分布基础上的。用来度量分类器的预测输出的概率分布和真实分布的差异，而不是去比较离散的类标签是否相等。因此，对数损失比较合理。

辅助评估指标是 accuracy_score。即模型根据预测的准确率的高低进行评估。

计算方法：如果在 m 个样本中有 a 个样本分类错误，则错误率 E=a/m；相应的，1-a/m 称为“准确率”（accuracy）。

合理性：由公式可以看出，准确率越高，分类正确的样本就更多，分类器的表现就越好。所以 accuracy_score 指标也是比较合理的。

二、分析

数据的探索

数据集为 20newsgroups。Subset 为‘train’的合计 11314 条数据；subset 为‘test’的合计 7532 条数据。

新闻数据共分为 20 个类别。分别是'alt.atheism'， 'comp.graphics'， 'comp.os.ms-windows.misc'， 'comp.sys.ibm.pc.hardware'， 'comp.sys.mac.hardware'， 'comp.windows.x'， 'misc.forsale'，'rec.autos'，'rec.motorcycles','rec.sport.baseball'，'rec.sport.hockey'，'sci.crypt'，'sci.electronics'， 'sci.med'， 'sci.space'， 'soc.religion.christian'， 'talk.politics.guns'， 'talk.politics.mideast'， 'talk.politics.misc'， 'talk.religion.misc'。

数据集的特征（features)为新闻的具体内容；标签（labels）为新闻的类别 ID，即 0-19 这 20 个数字。

Sklearn 已经对新闻的内容进行了处理，不包含重复文档和新闻组名，因此我们获取到的数据集是已经处理过的。因为是对新闻文档的处理，所以暂时不涉及分类变数、缺失数据、离群值等异常情况。

探索性可视化

数据集中新闻内容如下图（有些新闻内容很长，不便于展示，在这里我们选取的是比较短的第一条新闻）：

```
第一条新闻内容为：
From: ab4z@Virginia.EDU ("Andi Beyer")
Subject: Re: Israeli Terrorism
Organization: University of Virginia
Lines: 15

Well i'm not sure about the story nad it did seem biased. What
I disagree with is your statement that the U.S. Media is out to
ruin Israels reputation. That is rediculous. The U.S. media is
the most pro-israeli media in the world. Having lived in Europe
I realize that incidences such as the one described in the
letter have occured. The U.S. media as a whole seem to try to
ignore them. The U.S. is subsidizing Israels existance and the
Europeans are not (at least not to the same degree). So I think
that might be a reason they report more clearly on the
atrocities.
        What is a shame is that in Austria, daily reports of
the inhuman acts commited by Israeli soldiers and the blessing
received from the Government makes some of the Holocaust guilt
go away. After all, look how the Jews are treating other races
when they got power. It is unfortunate.
```

选取第一条新闻内容作为展示。通过随机分析观察不同的新闻内容，可以得出如下结论：新闻内容虽然各不相同，有些新闻比较短，有些新闻比较长，但是新闻的格式大致一样，尤其是新闻的头部，版式都是一致的，基本上都是包含 From，Subject，Organization，Lines 等。新闻的中间部分则是比较长篇的具体新闻描述。因此，该数据集比较的数据比较规范，这对于数据预处理很有帮助。

各个类别新闻的分布情况如下图，图中横坐标为新闻类别 ID，纵坐标为新闻数目，红

色的是 Subset 为‘train’的新闻，蓝色的是 subset 为‘test’的新闻：

通过上图可以观察到：Subset 为‘train’的新闻和 subset 为‘test’的新闻，首部和尾部类别的新闻数据稍微少一些，其他类别的新闻数据分布还是比较均匀的。总体来说，这些数据比较适合进行训练分类的。

算法和技术

数据预处理用到了NLTK工具包（Natural Language Toolkit ，简称NLTK）。使用其中的word_tokenize对文档内容进行分词处理；使用stopwords语料库对停用词进行处理；使用WordNetLemmatizer对单词进行同一词处理；使用pos_tag对单词进行词性标注（如and 是CC，并列连词；now 和completely 是RB，副词；for 是IN，介词；something 是NN，名词；different 是JJ，形容词；refuse 和permit 都以一般现在时动词（VBP）和名词（NN）形式出现，refuse是一个动词，意为“拒绝”，也是一个名词，意思是“垃圾”；以及其他形式的动词，以V开头）。对词性标记后，选择词性第一个属性为动词或者名词的单词进行同一词处理，也就是对名词处理单复数，对动词处理不同的时态。

词袋子模型：特征提取使用 sklearn.feature_extraction.text 中的 CountVectorizer 和 TfidfVectorizer。

CountVectorizer 算法描述：

概括为从语料库创建一个单词的字典，将每一个样本转化成一个关于每个单词在文档中出现次数的向量。

例如，文档分词为下面两个单词列表(“a”,”b”,”c”) 和 (“a”,”b”,”b”,”c”,”a”)。调用 CountVectorizer 产生词汇(a,b,c)的 CountVectorizerModel。然后根据每个单词在文档中出现的次数转换为输出向量如下(1,1,1)和(2,2,1)。向量(1,1,1)代表词典(a,b,c)中每个单词在单词列表(“a”,”b”,”c”)的出现次数，a 出现 1 次，b 出现 1 次，c 出现 1 次，所以向量为(1,1,1)；向量(2,2,1)代表词典(a,b,c)中每个单词在单词列表(“a”,”b”,”b”,”c”,”a”)的出现次数，a 出现 2 次，b 出现 2 次，c 出现 1 次，所以向量为(2,2,1)。

TF-IDF 算法描述：

TF: Term Frequency，词频，用于衡量一个单词在一个文档中的出现频率。TF 相当于对单词的出现次数做了一次归一化。这是因为每个文档的长度各不相同，有些甚至差别很大，所以一个单词在某个文档中出现的次数可能远远大于另一个文档，词频就是一个单词在文档中出现的次数除以文档的总长度。TF(t) = (词 t 在文档中出现的总次数) / (文档的词总数)。

IDF: Inverse Document Frequency，逆向文件频率，用于衡量一个单词的重要性。当我们在计算词频 TF 的时候，所有单词都被看成是一样的重要性，但是某些单词，比如“is”,“of”,

“the”，“this”等可能会出现很多次，但是可能根本不重要。因此，我们需要降低在多个文档中都频繁出现的单词的权重。IDF(t)=ln(总文档数/词 t 出现的文档数)。

TF-IDF：词频－逆向文件频率，将 TF 和 IDF 相乘就得到 TF-IDF 了。TF-IDF=TF*IDF

参数说明：analyzer 设置为‘word’，定义特征为词(word)。其他参数均取默认值。参数中的 stop_words，lowercase 都在数据预处理阶段处理完毕。因此该参数就不需要设置了。

词向量模型：特征提取使用 gensim.models 中的 word2vec。

Word2Vec 算法描述：

简单来说是一个具有一个隐含层的神经元网络。它的输入是词汇表向量，当看到一个训练样本时，对于样本中的每一个词，就把相应的在词汇表中出现的位置的值置为 1，否则置为 0。它的输出也是词汇表向量，对于训练样本的标签中的每一个词，就把相应的在词汇表中出现的位置的值置为 1，否则置为 0。那么，对所有的样本，训练这个神经元网络。收敛之后，将从输入层到隐含层的那些权重，作为每一个词汇表中的词的向量。

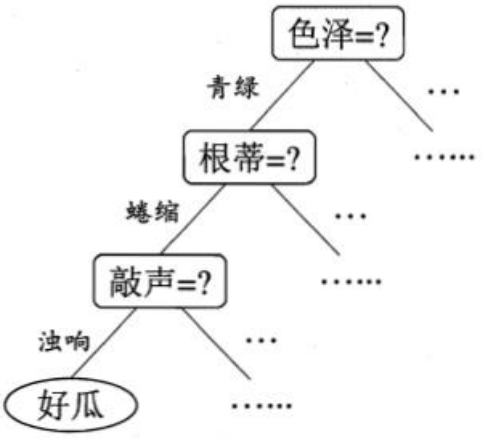
Word2Vec 有两种训练模型，CBOW (Continuous Bag-of-Words Model) 和 Skip-gram (Continuous Skip-gram Model)。CBOW 的做法是，将一个词所在的上下文中的词作为输入，而那个词本身作为输出，即看到一个上下文，希望预测出这个词和它的意思。Skip-gram 的做法是，将一个词所在的上下文中的词作为输出，而那个词本身作为输入，即给出一个词，希望预测可能出现的上下文的词。

参数说明：这里使用默认的参数 CBOW。size 是输出词向量的维数，设置为 200。window 是句子中当前词与目标词之间的最大距离，3 表示在目标词前看 3-b 个词，后面看 b 个词（b 在 0-3 之间随机），该参数设置为 5。其他参数使用默认值。

基准模型

词袋模型的训练与评估：使用 sklearn 中的决策树分类器 DecisionTreeClassifier，支持向量机分类器 SVC，朴素贝叶斯分类器 MultinomialNB。DecisionTreeClassifier 与 MultinomialNB 使用默认参数。

决策树：决策树（decision tree）是一类常见的机器学习方法。以二分类任务为例，我们希望从给定训练数据集学得一个模型用以对新示例进行分类，这个把样本分类的任务，可看作对“当前样本属于正类吗？”这个问题的“决策”或“判断”的过程。顾名思义，决策树是基于树结构来进行决策的，这恰是人类在面临决策问题时一种很自然的处理机制。如周志华老师的书《机器学习》中描述的挑选西瓜的问题，见下图：



决策树学习的关键在于，如何选择最优划分属性。分别有信息增益（information gain）、基尼指数（Gini index）等。DecisionTreeClassifier 的参数 criterion 默认的是 gini。假定当前样本集合 D 中第 k 类样本所占的比例为p_k，则数据集的纯度可用基尼值来度量：

$$\begin{aligned}\mathrm{Gini}(D) &= \sum_{k=1}^{|\mathcal{Y}|}\sum_{k'\neq k} p_k p_{k'} \\ &= 1 - \sum_{k=1}^{|\mathcal{Y}|} p_k^2 \ .\end{aligned}$$

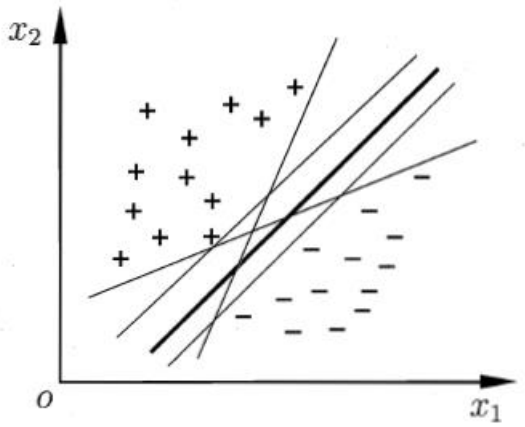
Gini(D)越小，则数据集 D 的纯度越高。属性 a 的基尼指数定义为

$$\mathrm{Gini\_index}(D,a) = \sum_{v=1}^{V}\frac{|D^v|}{|D|}\mathrm{Gini}(D^v) \ .$$

于是，在候选集合 A 中，选择那个使得划分后基尼指数最小的属性作为最优划分属性。

DecisionTreeClassifier 参数选择：都采用默认值。

支持向量机：（Support Vector Machine，简称 SVM）。给定训练样本集 D={$(x_1,y_1),(x_2,y_2),\ldots,(x_m,y_m)$}，$y_i$∈{-1,+1}，分类学习最基本的想法是基于训练集 D 在样本空间中找到一个划分超平面，将不同类别的样本分开。如下图：



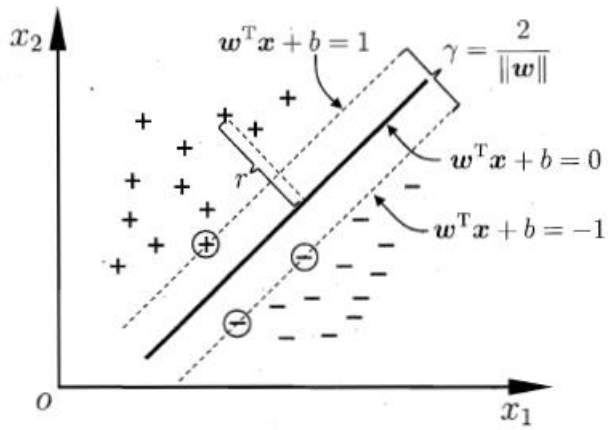
我们看到超平面可能会有很多。SVM 解决的是如何挑选最合适的超平面。超平面用法向量 w 和位移 b 确定。表示为w^Tx+b=0。样本空间中任意点 x 到超平面的距离可写为：

$$r = \frac{|\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b|}{||\boldsymbol{w}||} \ .$$

假设超平面能将样本正确分类，对于(x_i,y_i)∈D，若y_i=+1，则有$w^T x_i$+b>0；若y_i=−1，则有$w^T x_i$+b<0，距离超平面最近的这几个样本点被成为“支持向量”，两个异类支持向量到超平面的距离之和为

$$\gamma = \frac{2}{||\boldsymbol{w}||}$$

它被成为“间隔”。



寻找超平面就是找到“最大间隔”，也就是找到参数 w 和 b，是的 γ 最大。

SVC（支持向量机分类器 Support Vector Classification）参数说明：SVC 参数设置为 kernel = 'linear'即采用线性核函数，该核函数参数少，速度快，对于一般数据，分类效果已经比较理想，尤其是数据集经过词袋模型的特征提取，已经比较庞大，跟样本的数量差不多，此时选用该线性核函数比较理想。probability =True 即采用概率估计，也就是预测时不是简单的预测属于哪个分类，而是预测属于每个分类的概率，依此来方便模型计算 log_loss 损失。

朴素贝叶斯：先验与后验概率：以机器生产产品为例，机器调整良好的概率 P(B)=0.9，是根据以往的数据分析所得到的，称为先验概率；而条件概率 P(B|A)=0.945 是在得到产品合格的信息之后再重新加以修正的概率，称为后验概率。贝叶斯定理为：P(A|B)=P(B|A)P(A)/P(B)。样本 x，类标记 c，对类条件概率 P(x|c)来说，由于涉及关于 x 所有属性的联合概率，如果属性不相互独立的话，直接根据样本出现的频率来估计将会遇到严重的困难。为了方便计算联合概率，朴素贝叶斯分类器（naïve Bayes classifier）采用了“属性条件独立性假设”：对已知类别，假设所有属性相互独立。换言之，假设每个属性独立地对分类结果发生影响。基于属性条件独立性假设，可得：

$$P(c \mid \boldsymbol{x}) = \frac{P(c)\, P(\boldsymbol{x} \mid c)}{P(\boldsymbol{x})} = \frac{P(c)}{P(\boldsymbol{x})} \prod_{i=1}^{d} P(x_i \mid c) \ ,$$

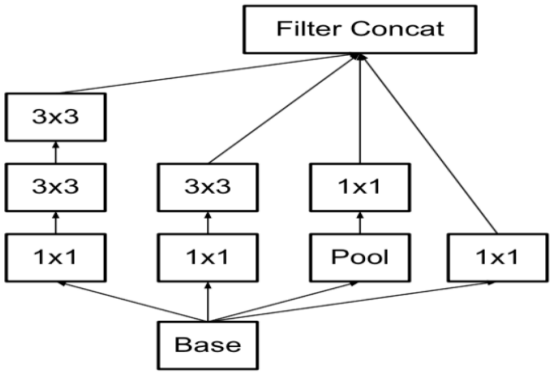
其中 d 为属性数目，x_i为 x 在第 i 个属性上的取值。由于对所有类别来说，P(x)相同，则有朴素贝叶斯分类器表达式：

$$h_{nb}(\boldsymbol{x}) = \arg\max_{c \in \mathcal{Y}} \ P(c) \prod_{i=1}^{d} P(x_i \mid c) \ ,$$

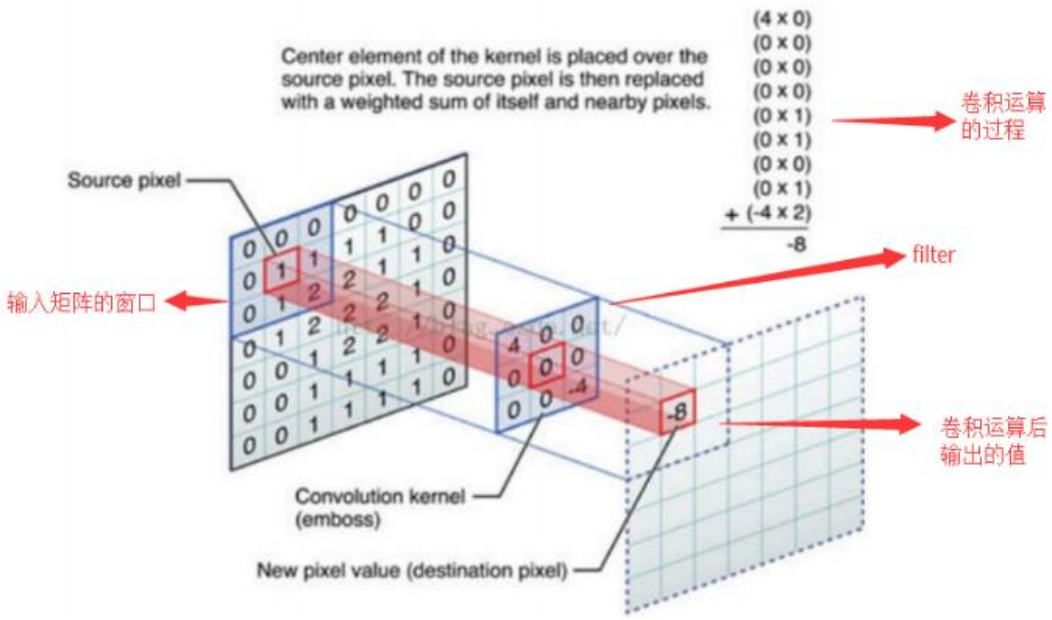
MultinomialNB（Naive Bayes classifier for multinomial models）参数：朴素贝叶斯分类器的多项模型。多项式朴素贝叶斯分类器适用于具有离散特征的分类(例如，文本分类的字数)。采用默认参数。

词向量模型的训练与评估：使用 keras.preprocessing.text 构造单词与序号之间的对应索引表 word_index；使用填充序列 pad_sequences 将数据集中的文档内容转换为形如(nb_samples,nb_timesteps)的 2D 张量。参数 maxlen：None 或整数，为序列的最大长度。大于此长度的序列将被截短，小于此长度的序列将在后部填 0。这里取值为 800。其他皆为默认值；使用 to_categorical 将标签处理成 one-hot 向量；根据 word_index 和 word2vec 词向量构造 weights 矩阵。然后就可以构造 Embedding 层了，第一个参数 input_dim 表示字典长度，第二个参数 output_dim 代表全连接嵌入的维度，trainable=False 设置该层的权重不可再训练；使用 keras 构建 Inception v2、CNN、LSTM 模型进行训练。

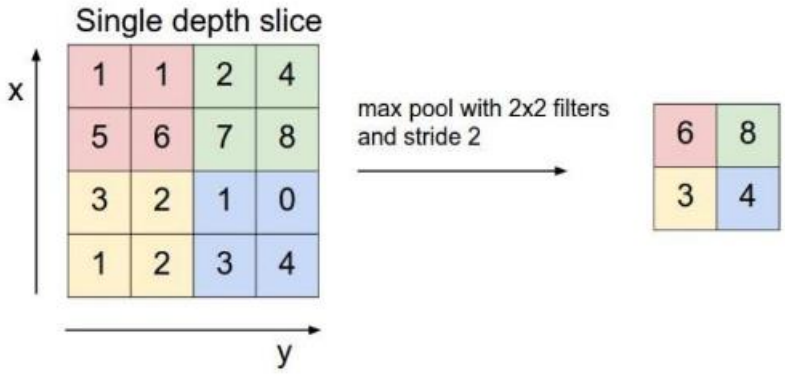
Inception v2 模型：卷积网络的变形。InceptionV2 的核心思想来自 Google 的《Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift》和《Rethinking the Inception Architecture for Computer Vision》这两篇论文。它根据第一篇论文加入了 BN 层。根据第二篇论文用一系列更小的卷积核（3x3）替代了原来的大卷积核（5x5,7x7）。大尺寸的卷积核可以带来更大的感受野，但也意味着更多的参数，比如 5x5 卷积核参数是 3x3 卷积核的 25/9=2.78 倍。为此，作者提出可以用 2 个连续的 3x3 卷积层(stride=1)组成的小网络来代替单个的 5x5 卷积层，(保持感受野范围的同时又减少了参数量)[6]，并且可以避免表达瓶颈，加深非线性表达能力。



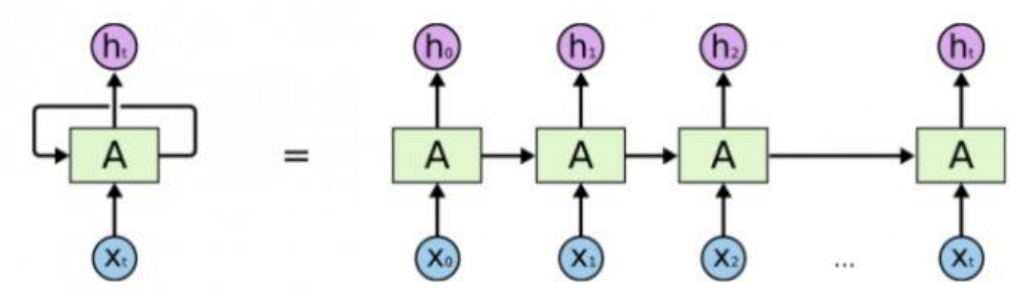
CNN 模型：Convolutional Neural Network（卷积神经网络）。在 MLP 多层感知器模型下，隐藏层的每个节点都可以看到输入层的所有信息，也就是说每个输入都与每个隐藏层节点相连。这样造成参数过多，容易过拟合的问题。采用局部连接层的思想，也就是将输入层分为 n 个部分，每个隐藏层节点仅能看到原来的 1/n，每个隐藏节点都能发现输入的 1/n 个区域的规律，然后每个隐藏节点依次向输出层汇报，输出层将从每个区域单独发现的规律，结合到一起。并通过权重共享更加有效的减少了参数。CNN 会逐渐获取空间数据，并将数组转换为包含输入的内容的表示，所有空间信息最终会丢失，一旦我们获得的表示不再具有输入的空间信息，就可以扁平化该数组，并将其提供给一个或多个完全连接层，判断输入信息中包含什么对象。卷积层计算过程：将输入节点与对应的权重相乘，然后对结果求和，添加偏差，如果激活函数为 ReLu 将正值保持不变，负值变为 0。



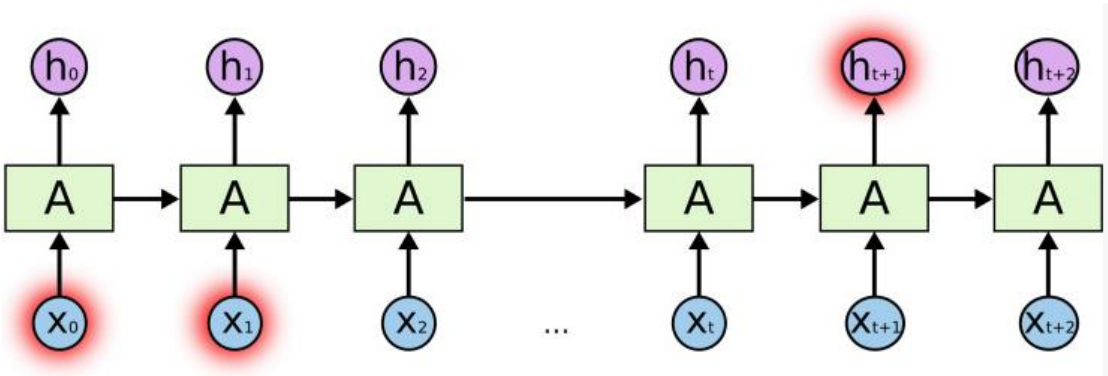
最大池化层计算过程：拿出窗口中包含的最大值。



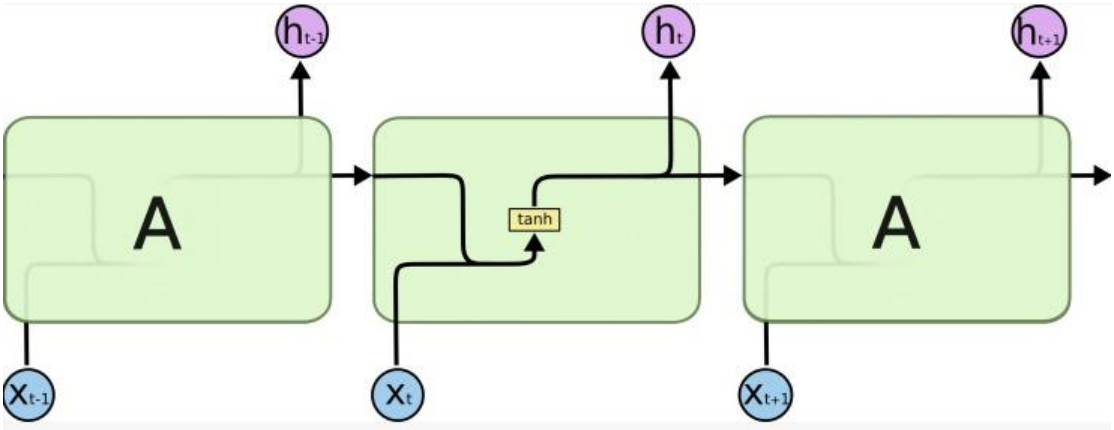
LSTM 模型：长短时记忆网络(Long Short Term Memory Network, LSTM)，是一种改进之后的循环神经网络，可以解决 RNN 无法处理长距离的依赖的问题。RNN（循环神经网络）顾名思义，也就是能够让信息在网络中再次循环的网络。



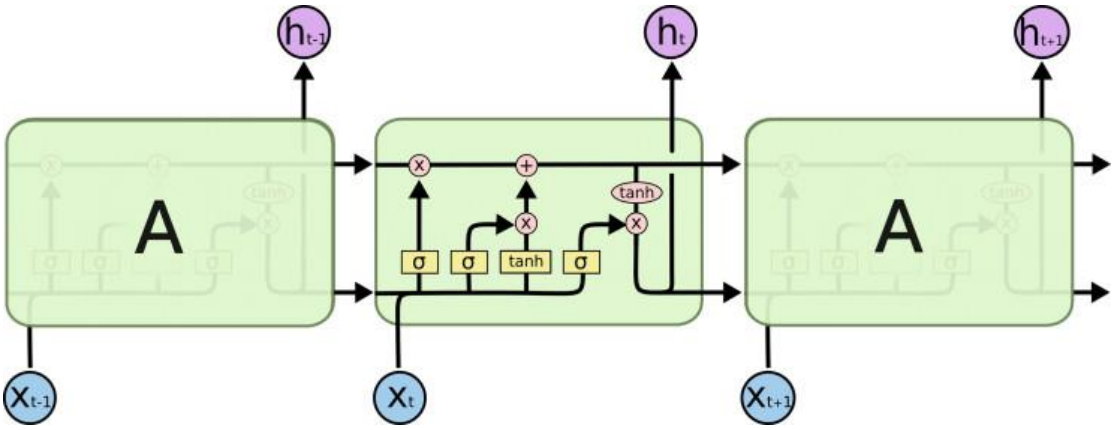
有时，我们只需要利用近期信息来处理当前任务。在这种情况下，相关信息所隔的距离并不远，因此 RNN 能够学会使用此前的信息。但是，就像我们做完型填空时，可能需要整合全文来填某一个句子，如果网络只知道邻近的几个单词，可能它会知道此处需要填写一门语言，但至于应该填什么，就需要找到更远前的信息，直到找到才行。这种需要寻找相距很远信息的情况，实际上非常常见。而糟糕的是，距离增加时，RNN 能将相关信息串联起来的能力也就越弱。



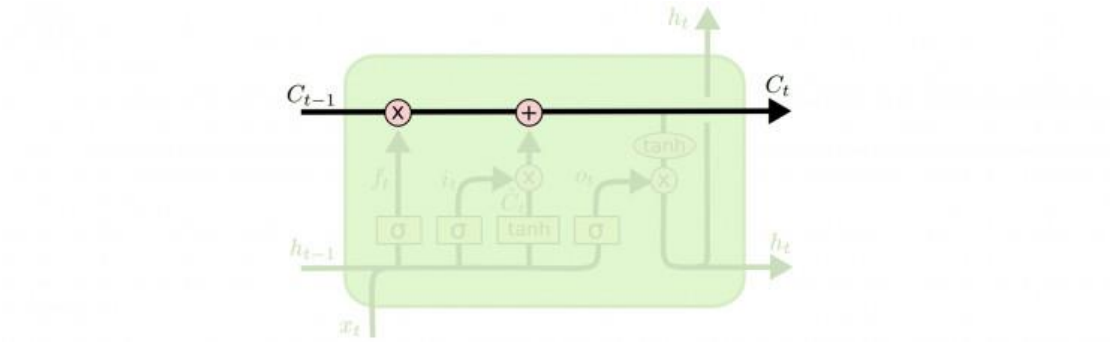
LSTM 就是为了解决长期依赖问题而生，也就是说，对信息的长期记忆是它们的自发行为，而不是刻意去学习的。所有的 RNN 都具备重复性的链条形式，而在标准的 RNN 中，这个重复模式都有着一个简单的结构，比如单层的 tanh 层。



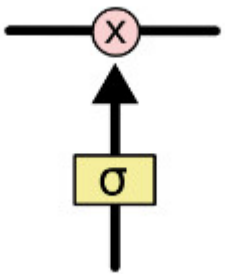
LSTM 也有这种结构化的链条，但重复性模块有着不同的结构。与 RNN 不同的是，LSTM 中有一个四层的网络，并以一种特殊的方式进行交互。



LSTM 的核心所在是 cell 的状态（cell state），也就是下图这条向右的线。Cell 的状态就像是传送带，它的状态会沿着整条链条传送，而只有少数地方有一些线性交互。信息如果以这样的方式传递，实际上会保持不变。



LSTM 通过一种名为门（gate）的结构控制 cell 的状态，并向其中删减或增加信息。



Sigmoid 层的输出值在 0 到 1 间，表示每个部分所通过的信息。0 表示“对所有信息关上大门”；1 表示“我家大门常打开”。一个 LSTM 有三个这样的门，控制 cell 的状态。

对于分类器的比较分析：因为我们有 20 个分类，如果随机猜测的话，准确率大约为 1/20，也就是 5%，因此，我们的模型准确率应该至少高于这个准确率。另外，对于模型的选择和调优，可以对比模型与模型之间的准确率，对比调优前后的准确率的高低。如果准确率相近，那么可以对比 log loss 损失，以此作为评价近似准确率的模型的基准，也就是 log loss 损失越低越好。因为与 log loss 比较来说，accuracy 更加直观，所以可以快速的通过 accuracy 判断模型表现的好坏，细致分析就得依靠 log loss 损失了。

下图为随机猜测的结果：

加载测试数据集

```
In [4]: import numpy as np
        y_test = np.load("y_test.npy")
        print('前50个测试集结果为',y_test[:50])
```

```
前50个测试集结果为 [ 9  1 10  5  4 18 17 16  8 13  5 19  5  9  8 13  2  5  5 13  8  1 15 16  6
 14  0  4  0 18 15 14  5  9 15  7  5  8 15  2  5  2 18  6  4  6 10 19  9  5]
```

使用随机分类结果进行预测

```
In [8]: predictions_val = np.random.randint(0,19,np.shape(y_test))
        print('前50个预测结果为',predictions_val[:50])
```

```
前50个预测结果为 [10  2  0 17 10 14  9  4  1  4  3  7  4  2  4 13 18  0 17 16  0  8 17  9  7
  0 14  9  2  6  4 10 11 16 13 17 16  5 13 10 15 10  6 14 15 10 16 12  5  9]
```

```
In [9]: from sklearn.metrics import fbeta_score, accuracy_score, log_loss
        # 计算准确率
        acc_val = accuracy_score(y_test,predictions_val)
        print('acc_val:{0:.3f}'.format(acc_val))
```

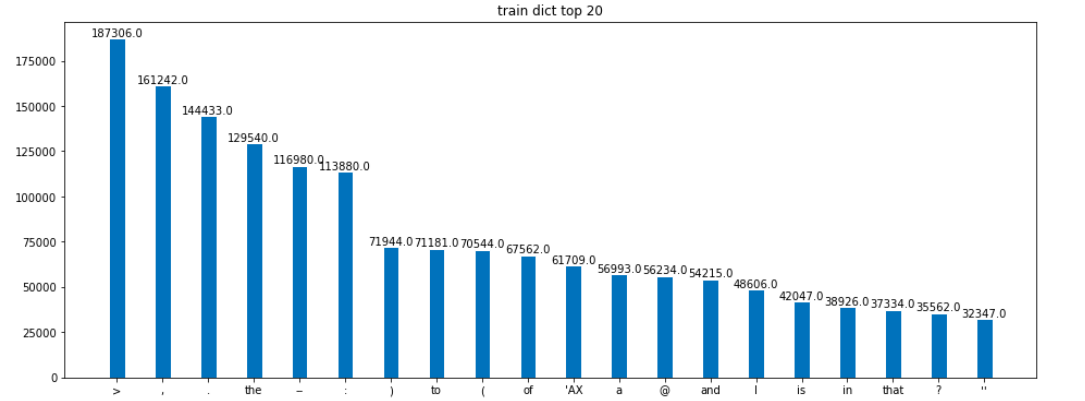
```
acc_val:0.052
```

可以看到，我们随机猜测的结果为 5.2%，与预期基本一致。

三、 方法

数据预处理

对 subset 为‘train’的数据集中的词汇进行统计分析，可得到下图：



上图展示了出现频率最高的前 20 个词汇，里面包括了标点符号，特殊符号，没有实际意义的功能词汇等等。下面进行描述预处理过程：

①使用正则式去掉符号，数字等；去掉剩余的单个字母，对数据集中的文档进行基本的数据清理。

②使用 nltk 的 word_tokenize 对文档进行分词处理。

③将单词转为小写。

④使用 nltk.corpus. stopwords 停用词语料库去掉停用词。停用词语料库如下图：

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yo
urselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',
 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'b
e', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'a
s', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'abov
e', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'wh
en', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own',
 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 'r
e', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'hav
en', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "should
n't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

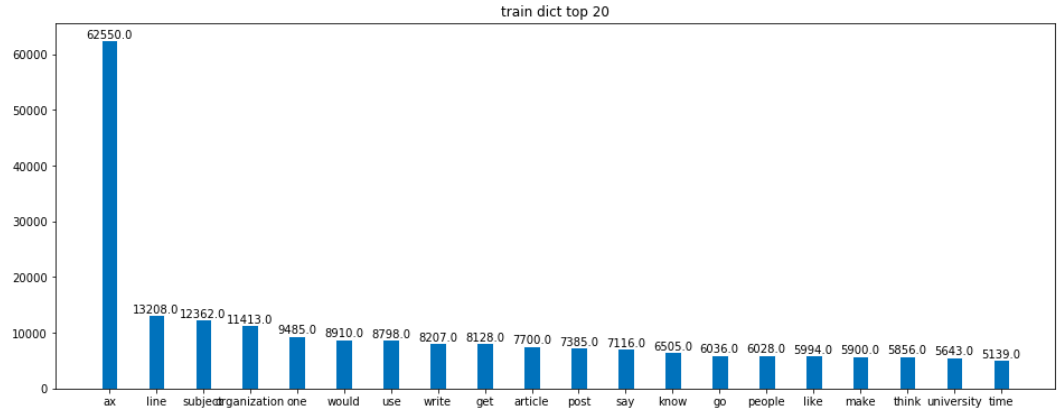
⑤使用 nltk. pos_tag 对单词进行词性标记，然后使用 nltk.stem.wordnet. WordNetLemmatizer 对单词进行同一词的统一的处理。下图为同一词处理的操作示例：

```
lemmatizer = WordNetLemmatizer()
print([lemmatize(token, tag) for token, tag in pos_tag(['lying','had','women','shoes'])])
```

```
['lie', 'have', 'woman', 'shoe']
```

通过上图例子可以看出，同一词处理有效的将单词进行了统一处理。

经过上述预处理过程，再次对 subset 为‘train’的数据集中的词汇进行统计分析，可得到下图：



可以看出，预处理有效的达到了对文本数据的清洗整理工作。下图中展示预处理过程中词典长度的变化：

处理步骤	开始	数据清理	转小写	去掉停用词	同一词统一
词典长度	211081	104860	82530	82386	74902

可以看出，经过预处理，词典长度逐渐降低了。有效的达到了降维的目的。

执行过程

使用 sklearn.model_selection. train_test_split 对数据集进行划分。Subset 为‘train’的新闻划分为 80%的训练数据 X_train , y_train，20%的验证数据 X_val, y_val；subset 为‘test’的新闻作为测试数据 X_test,y_test。

词袋子模型：使用 CountVectorizer 进行文本特征提取，直接使用 sklearn.feature_extraction.text. CountVectorizer 进行处理，使用 X_train 进行 fit 训练后，对其他数据集进行 transform 操作。然后使用 sklearn 中的决策树分类器 DecisionTreeClassifier，支持向量机分类器 SVC(kernel = 'linear',probability=True)，朴素贝叶斯分类器 MultinomialNB 对 CountVectorizer 处理后的数据进行训练和评估；使用 TfidfVectorizer 进行文本特征提取，直接使用 sklearn.feature_extraction.text. TfidfVectorizer 进行处理，使用 X_train 进行 fit 训练后，对其他数据集进行 transform 操作。然后使用 sklearn 中的决策树分类器 DecisionTreeClassifier，支持向量机分类器 SVC(kernel = 'linear',probability=True)，朴素贝叶斯分类器 MultinomialNB 对 TfidfVectorizer 处理后的数据进行训练和评估。

词向量模型：使用使用 Word2Vec 进行文本特征提取，直接使用 gensim.models. word2vec 对数据集中的文档进行词向量训练。对词向量进行简单评测。使用 keras.preprocessing.text. Tokenizer 对所有文本构造单词的索引表 word_index；使用 texts_to_sequences 表示文本；使用 pad_sequences 将数据集中的文档内容转换为形如(nb_samples,nb_timesteps)的 2D 张量；利用 Word2Vec 和 word_index 构建词向量矩阵；将这个词向量矩阵加载到 Embedding 层中，设置 trainable=False 使得这个编码层不可再训练；标签处理成 one-hot 向量，用 keras 的 to_categorical 实现；利用 Embedding 层构造 Inception v2、CNN、LSTM 模型，对数据进行训练和评估。

在执行过程中，首先遇到的一个问题是在预处理阶段，同一词等处理过程耗时较长。因此，在数据集预处理完毕，将其保存到了本地，下次就只需要直接加载已经预处理过的数据即可；另外还遇到了支持向量机分类器 SVC 的训练时间和预测时间较长，并且每次重新训练预测后得到的结果会有非常细微的差别，因此，增加了一个 init_flag 参数来控制模型是训练一个新模型还是加载已经保存到本地的模型。

完善

词袋模型首先使用 sklearn 中的决策树分类器 DecisionTreeClassifier，支持向量机分类器 SVC(kernel = 'linear',probability=True)，朴素贝叶斯分类器 MultinomialNB 对 CountVectorizer 和 TfidfVectorizer 文本特征提取的数据进行训练，在测试集上预测；然后选出表现好的模型和特征提取方法，对其使用 GridSearchCV 进行模型调优；最后使用表现更好的模型对测试数据集进行最终预测。

词向量模型分别对新闻数据和 text8 语料库使用 Word2Vec 进行文本特征提取，对两者结果进行简单评测，选出表现较好的语料库形成的 Word2Vec 模型；构建 Embedding 层，搭建 Inception v2、CNN、LSTM 模型然后进行训练和评估；进行调优工作，调优过程中，卷积层个数、dropout 的比例、optimizer 优化器、epoch 训练步数等都进行了调整；经过调试，最终挑出一个比较好的模型，然后对测试数据集进行最终预测。

在 Inception v2、CNN、LSTM 调试过程中，使用 ModelCheckpoint，参数 save_best_only 设置为 True，只保存在验证集上性能最好的模型。然后就可以加载具有最佳验证 loss 的模型权重进行预测评估。

四、 结果

模型的评价与验证

词袋模型：对 CountVectorizer 和 TfidfVectorizer 文本特征提取的数据分别使用决策树分类器、支持向量机分类器、朴素贝叶斯分类器进行训练和评估，结果如下：

模型预测准确率	DecisionTreeClassifier	SVC	MultinomialNB
CountVectorizer	0.654	0.848	0.849
TfidfVectorizer	0.641	0.914	0.865

模型预测log_loss	DecisionTreeClassifier	SVC	MultinomialNB
CountVectorizer	11.935	0.586	2.613
TfidfVectorizer	12.393	0.308	1.008

通过对比分析：模型在决策树分类器上表现不佳；在支持向量机分类器和朴素贝叶斯分类器上表现不错。总体来说，使用 TfidfVectorizer 进行文本特征提取，准确率有所提升，log loss 有所下降。并且在支持向量机分类器上表现很不错，准确率达到了 0.914，而 log loss 降到了 0.308，表现不错。

决策树模型在这里表现不佳，原因有：当具有包含大量特征的数据时，复杂的决策树可能会过拟合数据，分类的类别过于复杂时决策树也会表现很差。我们的数据集中有 20 个分类，属于比较多的分类任务了。

通过上述分析，我们选择 TF-IDF 模型进行 GridSearchCV 调优。结果如下（因为 SVC 调优没有效果，因此，又重新选择了预测表现稍微次之的朴素贝叶斯分类器进行调优）：

模型预测准确率	SVC	MultinomialNB
调优前	0.9143	0.8648
调优后	0.9138	0.9081

模型预测log loss	SVC	MultinomialNB
调优前	0.3094	0.3162
调优后	0.3094	0.3162

通过对比，调优前后 log loss 损失没有变化，SVC 准确率没有提升，MultinomialNB 虽有明显提升，但是还是低于原来的 SVC 模型。因此，选择原来的 SVC 模型进行测试集预测。测试数据集上，准确率达到 0.8216，Log_loss 为 0.6271，效果还是不错的。

词向量模型：Inception v2、CNN、LSTM 模型对比如下：

模型预测	Inception v2	CNN	LSTM
准确率	0.61	0.66	0.83
log loss	1.29	1.08	0.54
单Epoch耗时	13s	12s	69s

通过对比分析：LSTML 的准确率有大幅提升，达到了 0.83，log loss 损失也明显减少，达到了 0.54。但是 LSTM 的耗时更长一些。因此，我们选择 LSTM 模型对测试集进行最终的预测。；对测试数据集进行最终预测，在测试集上，准确率为 0.77，log_loss 为 0.82，该结果相对来说还可以。

总结：最终得到的模型还是比较合理的，虽然没有达到 100%预测正确，但是还是可以接受的；模型还是比较稳健的，虽然每次重新进行数据划分，训练后会有一些细微的变化，但是预测的准确率和 log loss 损失不会发生剧烈的变化，也不会对结果产生巨大的影响；因此，可以得出模型得到的结果还是比较可信的。

合理性分析

进过对比分析，词袋模型和词向量模型各有各的特色。在两种模型下，分类模型的预测结果还是比较合理的。该结果可以说，对我们开始提出的问题：“文本表示的方式？”可以使用词袋模型和词向量模型进行表示。但是如果进行实际应用的话，还需要进行细致的分析和研究，以便提升预测的准确率和加快训练的时间。

五、 项目结论

结果可视化

词袋模型：不同的分类器的训练和预测结果使用表格进行了对比展示：

文本特征提取对比

模型预测准确率	DecisionTreeClassifier	SVC	MultinomialNB
CountVectorizer	0.654	0.848	0.849
TfidfVectorizer	0.641	0.914	0.865

模型预测log_loss	DecisionTreeClassifier	SVC	MultinomialNB
CountVectorizer	11.935	0.586	2.613
TfidfVectorizer	12.393	0.308	1.008

通过对比分析：模型在决策树分类器上表现不佳；在支持向量机分类器和朴素贝叶斯分类器上表现不错。总体来说，使用TfidfVectorizer进行文本特征提取，准确率有所提升，log loss有所下降。并且在支持向量机分类器上表现很不错，准确率达到了0.914，而log loss降到了0.308，表现不错。

通过表格的对比分析，可以很快的锁定表现好的词袋模型和分类器，然后就可以用该词袋模型和分类器进行下一步的优化。

词袋模型的调优结果也进行了表格展示：

模型预测准确率	SVC	MultinomialNB
调优前	0.9143	0.8648
调优后	0.9138	0.9081

模型预测log loss	SVC	MultinomialNB
调优前	0.3094	0.3162
调优后	0.3094	0.3162

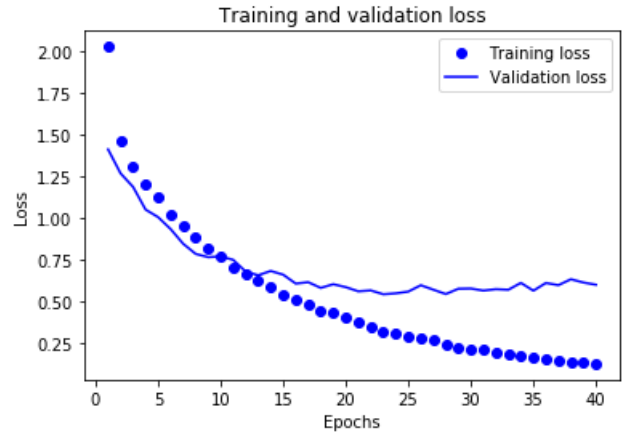
通过对表格分析，我们选择了表现最好的 SVC 模型，对测试数据集进行最终预测。

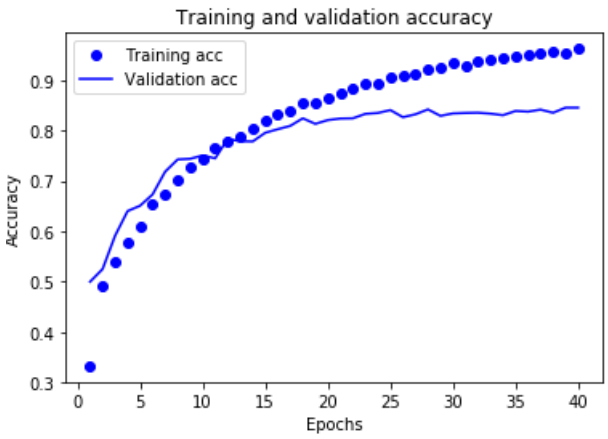
词向量模型：首先使用 model.summary()比较直观的展示了 Inception v2、CNN、LSTM 模型的结构。下图为 LSTM:

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 500, 200)	19636600
lstm_1 (LSTM)	(None, 200)	320800
dropout_1 (Dropout)	(None, 200)	0
dense_1 (Dense)	(None, 20)	4020

Total params: 19,961,420
Trainable params: 324,820
Non-trainable params: 19,636,600

然后使用图像描述了损失曲线和准确率曲线的变化（横轴为训练轮数，纵轴为 log loss 损失和准确率，点为训练结果，线为验证结果）。下图为 LSTM 的结果：





通过对曲线的分析，可以很直观的迅速找到模型是否过拟合，需要增加还是减少训练轮数等等；以此就可以对模型进行快速调节，以便达到更好的预测效果。

对项目的思考

对项目的整个流程已经做了比较详尽的描述。通过整个项目，对监督学习分类器有了更多的了解，对 NLP 的流程有了一定的认识。

项目中比较有意思的地方是：通过做项目的过程中，不断的加深了对词袋模型和词向量模型的理解，发现取名字是一件很有意思的事情。词袋子和词向量这两个词非常形象，将其代表的意思已经比较明显的表现出来了。

项目中比较困难的地方：首先支持向量机的训练速度太慢，在数据量稍微大一些的情况下尤为明显。在项目里面用的是线性核函数，已经很慢了。如果对 SVC 进行 GridSearchCV 调优，运行时间非常非常漫长；另外就是深度学习模型对显卡的要求还是比较高的，在笔记本上运行，有些参数设置的比较大一些就会出现异常，并且每训练一次，python 进程就会死掉一次。

最终模型和结果还算符合我对这个问题的期望。如果在通用的场景下解决这类问题还需要进行进一步的优化。

需要作出的改进

词袋模型对 SVC 进行 GridSearchCV 调优，发现调优后的 log loss 没有变化，而准确率有稍微下降。具体产生这个现象的原因还需要进一步的深入研究。为此程序中继续选择了使用朴素贝叶斯分类器进行调优，以作为对比。

深度学习模型在笔记本上运行比较费事，如果有条件进行设备升级或者购买服务器的话，还需要进一步对深度学习模型的参数和结构进行调优。

另外，如果数据集比较大的话，特征维度也就比较大，因此降维也需要进一步的研究。

词向量模型如果采用更大更好的数据集进行训练，效果会更佳。

六、参考文献

【1】王晓龙，《计算机自然语言处理》，清华大学出版社，2005 年

【2】Steven Bird, Ewan Klein & Edward Loper，《PYTHON 自然语言处理》，O'REILLY，2012 年

【3】周志华，《机器学习》，清华大学出版社，2016 年

【4】Blog：http://colah.github.io/posts/2015-08-Understanding-LSTMs/