项目2 基本中文处理

- # 项目背景
  - 新闻文本类别预测，主要使用fastText, testCNN, RNN, RCNN, text-bi-RNN等模型
- # 数据介绍
  - 新闻数据，共有五类：科技、汽车、娱乐、军事、运动

    (据统计，2016年仅在中国田径协会注册的马拉松赛事便达到了328场，继续呈现出爆发式增长的态势，2015年，这个数字还仅仅停留在134场。如果算上未在中国田协注册的纯“民间”赛事，国内全年的路跑赛事还要更多。，sport)
- # 数据分析与预处理
  - ## 切分数据集
    - train_test_split

      ```
      from sklearn.model_selection import train_test_split
      x, y = zip(*sentences)
      x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=1234)
      ```
    - StratifiedKFold交叉验证

      ```
      from sklearn.model_selection import StratifiedKFold
      stratifiedk_fold = StratifiedKFold(n_splits=n_folds, shuffle=shuffle)
      for train_index, test_index in stratifiedk_fold.split(x, y):
          model.fit()
          .......
      ```
  - 由于此次提供的数据集已经清洗过，在此没有特别进行预处理
- # 特征工程
  - ## 词袋模型
    - CountVectorizer

      ```
      定义： vec = CountVectorizer(
        analyzer='word', # tokenise by character ngrams
        max_features=4000,  # keep the most common 4000 ngrams
        ngram_range(1,4) 1-4 grams
      )
      训练： vec.fit(x_train)
      使用： vec.transform(x)
      ```
  - ## 基于TF-IDF算法的关键字抽取（jieba）

    import jieba.analyse
    - jieba.analyse.**extract_tags**(sentence, topK=20, withWeight=False, allowPOS=())
      - sentence 为待提取的文本

            - topK 为返回几个 TF/IDF 权重最大的关键词，默认值为 20
            - withWeight 为是否一并返回关键词权重值，默认值为 False
            - allowPOS 仅包括指定词性的词，默认值为空，即不筛选
    - 基于 TextRank 算法的关键词抽取
        - 基本思想:
            - 将待抽取关键词的文本进行分词
            - 以固定窗口大小(默认为5，通过span属性调整)，词之间的共现关系，构建图
            - 计算图中节点的PageRank，注意是无向带权图
        - 使用
            - jieba.analyse.**textrank**(sentence, topK=20, withWeight=False, allowPOS=('ns', 'n', 'vn', 'v')) 直接使用，接口相同，注意默认过滤词性。
            - jieba.analyse.TextRank() 新建自定义 TextRank 实例
- 建模的方法
    - 朴素贝叶斯
        - 最简单的模型，可以作为baseline

```
from sklearn.naive_bayes import MultinomialNB
classifier = MultinomialNB()
classifier.fit(vec.transform(x_train), y_train)
classifier.score(vec.transform(x_test), y_test)
```

    - 支持向量机
        - 通过变换不同的核，尝试不同的效果

```
from sklearn.svm import SVC
svm = SVC(kernel='linear')
svm.fit(vec.transform(x_train), y_train)
svm.score(vec.transform(x_test), y_test)
```

    - fastText
        - 安装：使用pip install fasttext
        - 数据格式：需要存储为fasttext所需的格式（__label__1, '一段文字，属于label 1'）
        - 调用模型：supervised

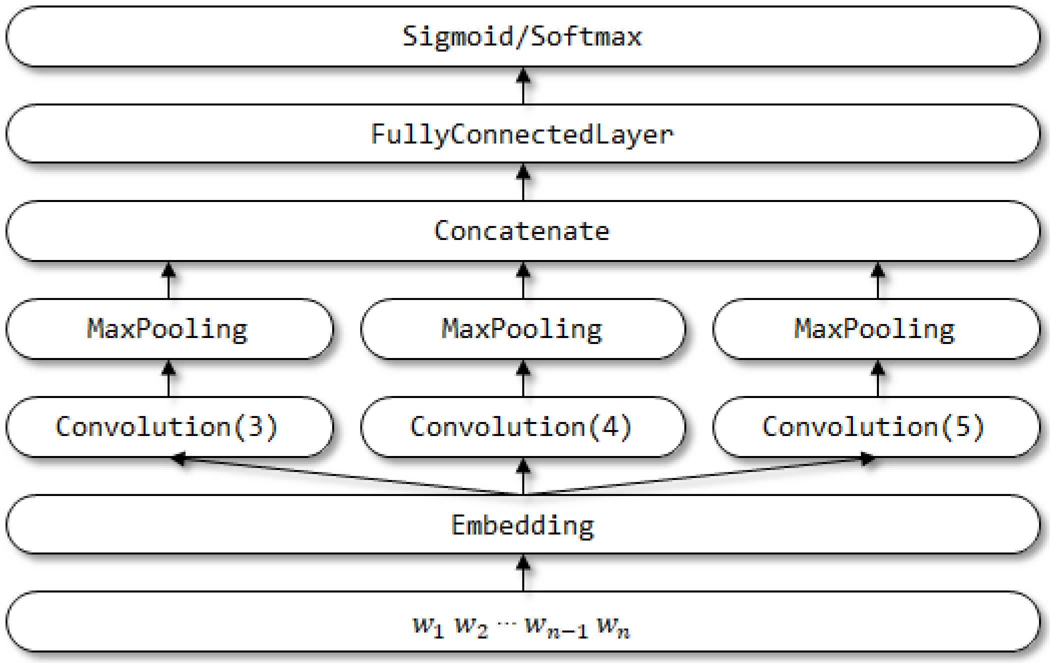
```
import fasttext
classifier = fasttext.supervised('train_data.txt', 'classifier.model', label_prefix='__label__')
```

        - 评估模型：test

```
result = classifier.test('train_data.txt')
print 'P@1:', result.precision
print 'R@1:', result.recall
print 'Number of examples:', result.nexamples
```

    - TextCNN

- TextCNN出处：论文Convolutional Neural Networks for Sentence Classification
- 论文中模型定义

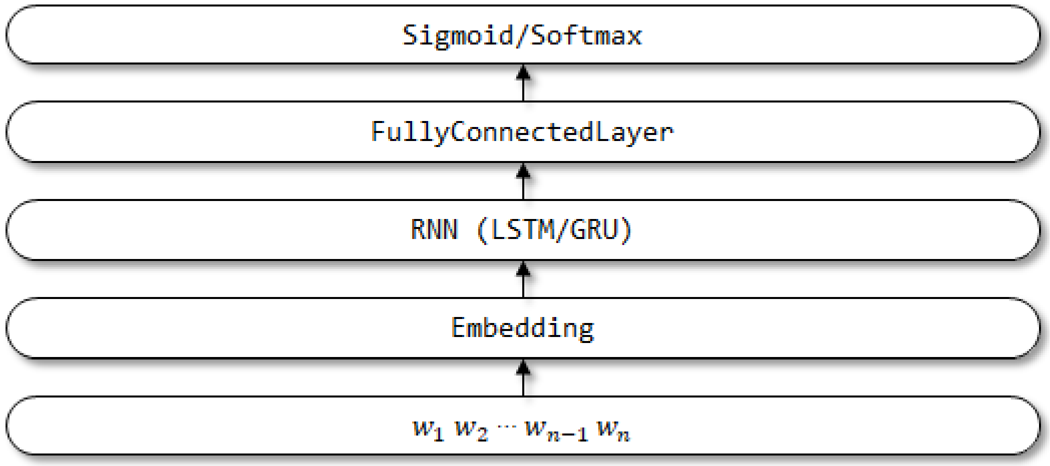



```
from tensorflow.keras import Input, Model
from tensorflow.keras.layers import Embedding, Dense, Conv1D, GlobalMaxPooling1D, Concatenate,
Dropout

class TextCNN(object):
  def __init__(self, maxlen, max_features, embedding_dims,
         class_num=5,
         last_activation='softmax'):
    self.maxlen = maxlen
    self.max_features = max_features
    self.embedding_dims = embedding_dims
    self.class_num = class_num
    self.last_activation = last_activation

 def get_model(self):
    input = Input((self.maxlen,))
    embedding = Embedding(self.max_features, self.embedding_dims, input_length=self.maxlen)
(input)
    convs = []
    for kernel_size in [3, 4, 5]:
      c = Conv1D(128, kernel_size, activation='relu')(embedding)
      c = GlobalMaxPooling1D()(c)
      convs.append(c)
    x = Concatenate()(convs)
    output = Dense(self.class_num, activation=self.last_activation)(x)
    model = Model(inputs=input, outputs=output)
    return model
```

- 训练模型

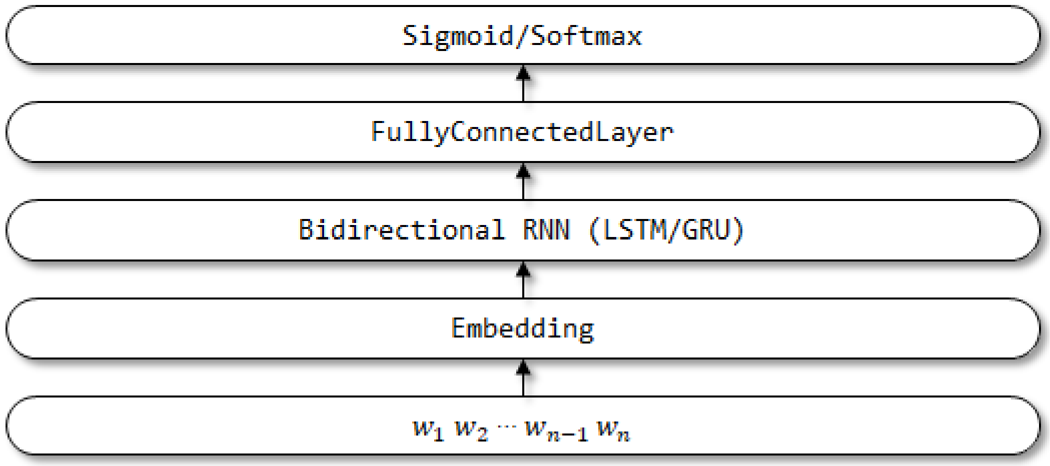
```
print('构建模型...')
model = TextRNN(maxlen, max_features,embedding_dims).get_model()
model.compile('adam', 'categorical_crossentropy', metrics=['accuracy'])
print('Train...')
early_stopping = EarlyStopping(monitor='val_accuracy', patience=2, mode='max')
history = model.fit(x_train, y_train,
         batch_size=batch_size,
        epochs=epochs,
        callbacks=[early_stopping],
        validation_data=(x_test, y_test))
```

- TextRNN
  - TextRNN相关论文：Recurrent Neural Network for Text Classification with Multi-Task Learning
  - 论文中模型定义与使用



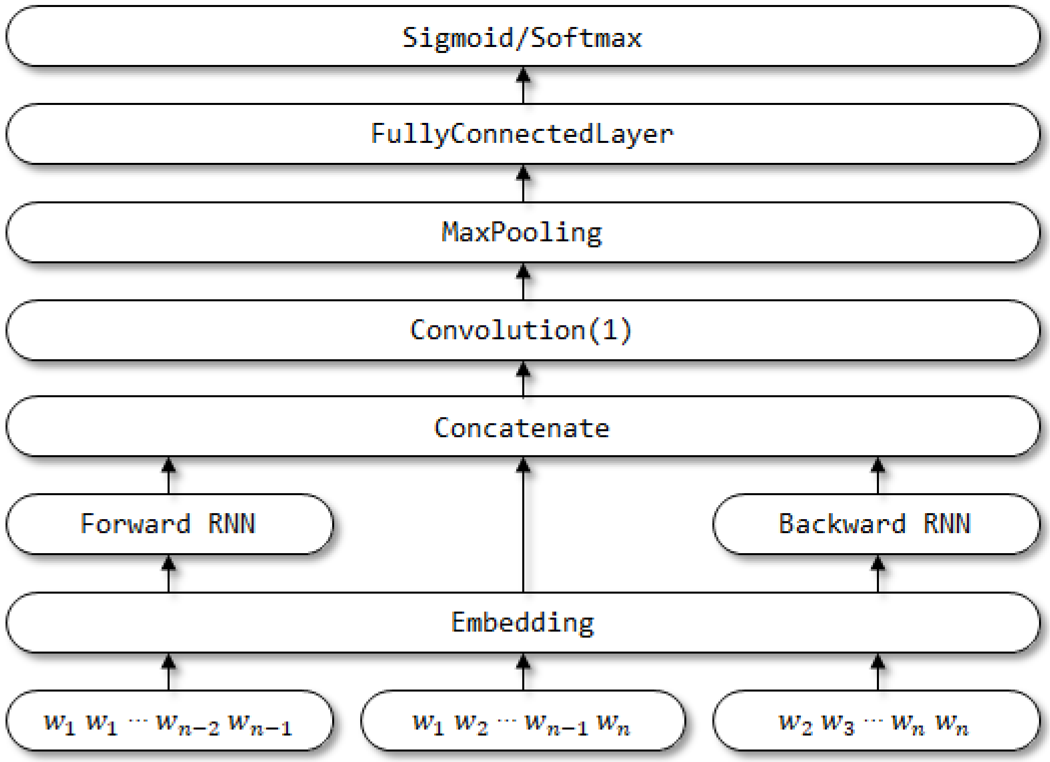

```
from tensorflow.keras import Input, Model
from tensorflow.keras.layers import Embedding, Dense, Dropout, LSTM
class TextRNN(object):
  def __init__(self, maxlen, max_features, embedding_dims,
         class_num=5,
         last_activation='softmax'):
    self.maxlen = maxlen
    self.max_features = max_features
    self.embedding_dims = embedding_dims
    self.class_num = class_num
    self.last_activation = last_activation

def get_model(self):
    input = Input((self.maxlen,))
    embedding = Embedding(self.max_features, self.embedding_dims, input_length=self.maxlen)
(input)
    x = LSTM(128)(embedding)
    output = Dense(self.class_num, activation=self.last_activation)(x)
    model = Model(inputs=input, outputs=output)
    return model
```

- TextBiRNN
  - TextBiRNN 是基于 TextRNN 的改进版本，将网络结构中的 RNN 层改进成了双向（Bidirectional）的 RNN 层，希望不仅能考虑正向编码的信息，也能考虑反向编码的信息。
  - 网路定义




```
from tensorflow.keras import Input, Model
from tensorflow.keras.layers import Embedding, Dense, Dropout, Bidirectional, LSTM
class TextBiRNN(object):
  def __init__(self, maxlen, max_features, embedding_dims,
         class_num=5,
         last_activation='softmax'):
    self.maxlen = maxlen
    self.max_features = max_features
    self.embedding_dims = embedding_dims
    self.class_num = class_num
    self.last_activation = last_activation

def get_model(self):
    input = Input((self.maxlen,))
    embedding = Embedding(self.max_features, self.embedding_dims, input_length=self.maxlen)
(input)
    x = Bidirectional(LSTM(128))(embedding)
    output = Dense(self.class_num, activation=self.last_activation)(x)
    model = Model(inputs=input, outputs=output)
    return model
```

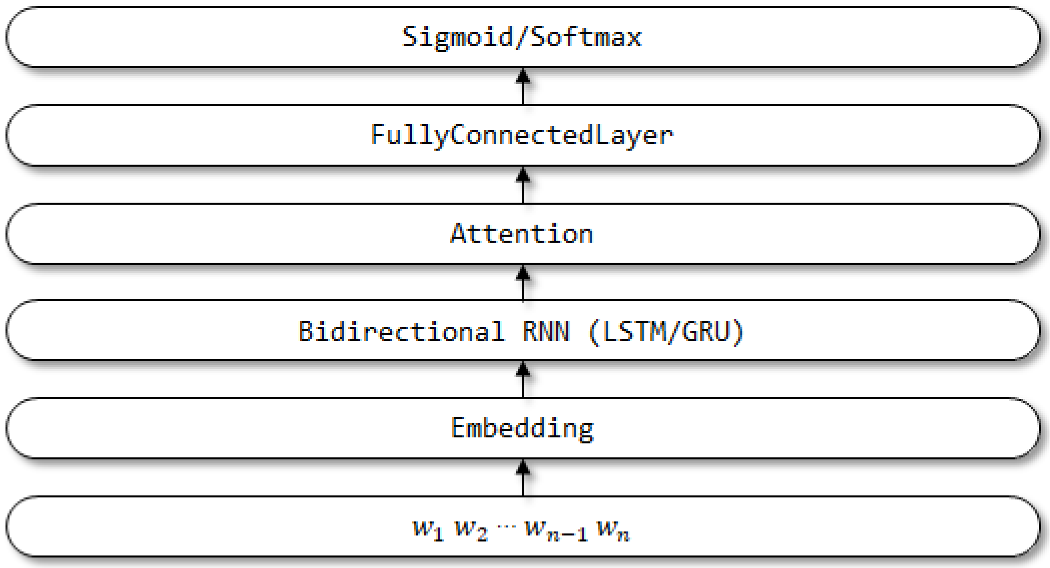
- TextRCNN
  - RCNN出处：论文Recurrent Convolutional Neural Networks for Text Classification
  - 网路定义




```
from tensorflow.keras import Input, Model
from tensorflow.keras import backend as K
from tensorflow.keras.layers import Embedding, Dense, SimpleRNN, Lambda, Concatenate, Conv1D,
GlobalMaxPooling1D
class RCNN(object):
  def __init__(self, maxlen, max_features, embedding_dims,
         class_num=5,
         last_activation='softmax'):
    self.maxlen = maxlen
    self.max_features = max_features
    self.embedding_dims = embedding_dims
    self.class_num = class_num
    self.last_activation = last_activation

def get_model(self):
    input_current = Input((self.maxlen,))
    input_left = Input((self.maxlen,))
    input_right = Input((self.maxlen,))

    embedder = Embedding(self.max_features, self.embedding_dims, input_length=self.maxlen)
    embedding_current = embedder(input_current)
    embedding_left = embedder(input_left)
    embedding_right = embedder(input_right)

    x_left = SimpleRNN(128, return_sequences=True)(embedding_left)
    x_right = SimpleRNN(128, return_sequences=True, go_backwards=True)(embedding_right)
    x_right = Lambda(lambda x: K.reverse(x, axes=1))(x_right)
    x = Concatenate(axis=2)([x_left, embedding_current, x_right])
    x = Conv1D(64, kernel_size=1, activation='tanh')(x)
```

```
        x = GlobalMaxPooling1D()(x)

        output = Dense(self.class_num, activation=self.last_activation)(x)
        model = Model(inputs=[input_current, input_left, input_right], outputs=output)
        return model
```

- TextAttBiRNN
  - TextAttBiRNN是在双向LSTM文本分类模型的基础上改进的，主要是引入了注意力机制（Attention）。对于双向LSTM编码得到的表征向量，模型能够通过注意力机制，关注与决策最相关的信息。
  - 注意力机制最先在论文 Neural Machine Translation by Jointly Learning to Align and Translate 中被提出，而此处对于注意力机制的实现参照了论文 Feed-Forward Networks with Attention Can Solve Some Long-Term Memory Problems。
  - 模型定义（代码仅作参考）

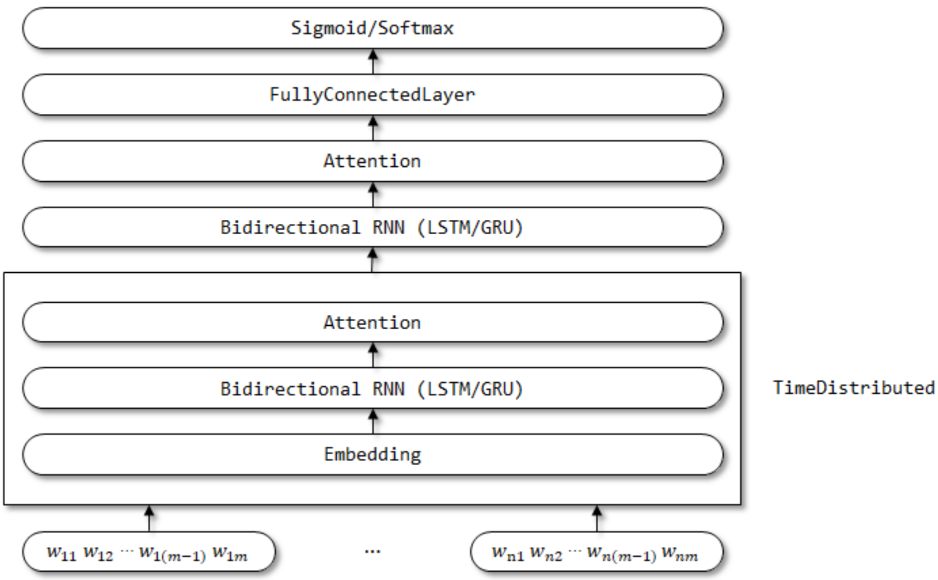



```
from tensorflow.keras import backend as K
from tensorflow.keras import initializers, regularizers, constraints
from tensorflow.keras.layers import Layer
class Attention(Layer):
    def __init__(self, step_dim,
                 W_regularizer=None, b_regularizer=None,
                 W_constraint=None, b_constraint=None,
                 bias=True, **kwargs):
        self.supports_masking = True
        self.init = initializers.get('glorot_uniform')
        self.W_regularizer = regularizers.get(W_regularizer)
        self.b_regularizer = regularizers.get(b_regularizer)
        self.W_constraint = constraints.get(W_constraint)
        self.b_constraint = constraints.get(b_constraint)
        self.bias = bias
        self.step_dim = step_dim
        self.features_dim = 0
        super(Attention, self).__init__(**kwargs)
```

```
def build(self, input_shape):
    assert len(input_shape) == 3
    self.W = self.add_weight(shape=(input_shape[-1],),
                    initializer=self.init,
                    name='{}_W'.format(self.name),
                    regularizer=self.W_regularizer,
                    constraint=self.W_constraint)
    self.features_dim = input_shape[-1]
    if self.bias:
      self.b = self.add_weight(shape=(input_shape[1],),
                      initializer='zero',
                      name='{}_b'.format(self.name),
                      regularizer=self.b_regularizer,
                      constraint=self.b_constraint)
    else:
      self.b = None
    self.built = True
def compute_mask(self, input, input_mask=None):
    # do not pass the mask to the next layers
    return None
def call(self, x, mask=None):
    features_dim = self.features_dim
    step_dim = self.step_dim
    e = K.reshape(K.dot(K.reshape(x, (-1, features_dim)), K.reshape(self.W, (features_dim, 1))), (-1,
step_dim))  # e = K.dot(x, self.W)
    if self.bias:
      e += self.b
    e = K.tanh(e)
    a = K.exp(e)
    if mask is not None:
      a *= K.cast(mask, K.floatx())
    a /= K.cast(K.sum(a, axis=1, keepdims=True) + K.epsilon(), K.floatx())
    a = K.expand_dims(a)
    c = K.sum(a * x, axis=1)
    return c
def compute_output_shape(self, input_shape):
    return input_shape[0], self.features_dim


from tensorflow.keras import Input, Model
from tensorflow.keras.layers import Embedding, Dense, Dropout, Bidirectional, LSTM
class TextAttBiRNN(object):
  def __init__(self, maxlen, max_features, embedding_dims,
         class_num=5,
         last_activation='softmax'):
    self.maxlen = maxlen
    self.max_features = max_features
```

```
        self.embedding_dims = embedding_dims
        self.class_num = class_num
        self.last_activation = last_activation

    def get_model(self):
        input = Input((self.maxlen,))
        embedding = Embedding(self.max_features, self.embedding_dims, input_length=self.maxlen)
(input)
        x = Bidirectional(LSTM(128, return_sequences=True))(embedding)  # LSTM or GRU
        x = Attention(self.maxlen)(x)
        output = Dense(self.class_num, activation=self.last_activation)(x)
        model = Model(inputs=input, outputs=output)
        return model
```

- HAN(层叠注意力)
  - HAN出处：论文Hierarchical Attention Networks for Document Classification
  - 网路定义：分别对词，句区分对应网络层级




```
class HAN(object):
    def __init__(self, maxlen_sentence, maxlen_word, max_features, embedding_dims,
            class_num=5,
            last_activation='softmax'):
        self.maxlen_sentence = maxlen_sentence
        self.maxlen_word = maxlen_word
        self.max_features = max_features
        self.embedding_dims = embedding_dims
        self.class_num = class_num
        self.last_activation = last_activation

    def get_model(self):
        # Word part
        input_word = Input(shape=(self.maxlen_word,))
        x_word = Embedding(self.max_features, self.embedding_dims, input_length=self.maxlen_word)
```

```
(input_word)
    x_word = Bidirectional(LSTM(128, return_sequences=True))(x_word)  # LSTM or GRU
    x_word = Attention(self.maxlen_word)(x_word)
    model_word = Model(input_word, x_word)
    # Sentence part
    input = Input(shape=(self.maxlen_sentence, self.maxlen_word))
    x_sentence = TimeDistributed(model_word)(input)
    x_sentence = Bidirectional(LSTM(128, return_sequences=True))(x_sentence)  # LSTM or GRU
    x_sentence = Attention(self.maxlen_sentence)(x_sentence)
    output = Dense(self.class_num, activation=self.last_activation)(x_sentence)
    model = Model(inputs=input, outputs=output)
    return model
```

- # 最终效果
  - 以上几种模型的最终效果均为输入一段文字，输出对应的新闻主题分类。
- # 其他
  - ## 词云WordCloud

    ```
    from wordcloud import WordCloud#词云包
    wordcloud.fit_words(word_frequence)
    ```

    - 输入：词词频列表

      ```
      电影 1000
      ```

    - 参数：
      - font_path字体
      - background_color背景色
      - max_font_size字体大小
      - mask 自定义背景做词云
    - 输出：词云图片
  - ## LDA主题模型
    - 载入停用词：读取stopwords.txt文件

      ```
      stopwords=pd.read_csv("origin_data/stopwords.txt",index_col=False,quoting=3,sep="\t",names=['stopword'], encoding='utf-8')stopwords=stopwords['stopword'].values
      ```

    - 分词：把句子转换为词，移除停用词

      ```
      segs=jieba.lcut(line)segs = list(filter(lambda x:len(x)>1, segs))segs = list(filter(lambda x:x not in stopwords, segs))sentences.append(list(segs))
      ```

    - 词袋模型
      - 把之前的每一句，转换为[(word1,freq1),(word2,freq2),...]的格式
        - dictionary = corpora.Dictionary(sentences)
        - corpus = [dictionary.doc2bow(sentence) for sentence in sentences]
    - LDA建模

- lda = gensim.models.**ldamodel.LdaModel**(corpus=corpus, id2word=dictionary, num_topics=20)

  print(lda.print_topic(3, topn=5)) 0.040*"产品" + 0.016*"品牌" + 0.016*"消费者" + 0.015*"市场" + 0.012*"体验"

- 机器学习方法完成中文文本分类
  - 文本分类 = 文本表示 + 分类模型
  - 文本表示：BOW/N-gram/TF-IDF/word2vec/word embedding/ELMo
    - 词袋模型（中文）①分词：

      第1句话：[w1 w3 w5 w2 w1...]
      第2句话：[w11 w32 w51 w21 w15...]
      第3句话...
      ...
    - ②统计词频：

      w3 count3
      w7 count7
      wi count_i
      ...
    - ③构建词典：

      选出频次最高的N个词
      开[1*n]这样的向量空间
      （每个位置是哪个词）
    - ④映射：把每句话共构建的词典进行映射

      第1句话：[1 0 1 0 1 0...]
      第2句话：[0 0 0 0 0 0...1, 0...1,0...]
    - ⑤提升信息的表达充分度：

      把是否出现替换成频次
      不只记录每个词，我还记录连续的**n-gram**"李雷喜欢韩梅梅" => ("李雷","喜欢","韩梅梅")
      "韩梅梅喜欢李雷" => ("李雷","喜欢","韩梅梅")
      "李雷喜欢韩梅梅" => ("李雷","喜欢","韩梅梅","李雷喜欢", "喜欢韩梅梅")
      "韩梅梅喜欢李雷" => ("李雷","喜欢","韩梅梅","韩梅梅喜欢","喜欢李雷")
      不只是使用频次信息，需要知道词对于句子的重要度**TF-IDF** = TF(term frequency) + IDF(inverse document frequency)
    - ⑥上述的表达都是独立表达（没有词和词在含义空间的分布）

      喜欢 = 在乎 = “稀罕” = “中意”
      - 文本预处理

        时态语态Normalize
        近义词替换
        stemming
        ...
      - 希望能够基于海量数据的分布去学习到一种表示

        nnlm => 词向量
        word2vec（周边词类似的这样一些词，是可以互相替换，相同的语境）捕捉的是相关

> 的词，不是近义词我 讨厌 你
> 我 喜欢 你
> word2vec优化...
> 用监督学习去调整word2vec的结果(word embedding/词嵌入)

- word-net （把词汇根据关系构建一张网：近义词、反义词、上位词、下位词...）

  > 怎么更新?
  > 个体差异?

- 分类模型：NB/LR/SVM/LSTM(GRU)/CNN

  > 对向量化的输入去做建模

  - 语种判断：拉丁语系，字母组成的，甚至字母也一样 => 字母的使用(次序、频次)不一样
  - ①NB/LR/SVM...建模
    - - 可以接受特别高维度的稀疏表示
  - ②MLP/CNN/LSTM
    - - 不适合稀疏高维度数据输入 => word2vec

- 将自己的文本分类器写为类
  - 可以直接调用，自动化执行特征提取到模型训练的过程
- 模型结构打印
  - from tensorflow.keras.utils import plot_model
  - plot_model(model, show_shapes=True, show_layer_names=True)