“华为杯”第十五届中国研究生数学建模竞赛

学　　校	华东师范大学	
参赛队号	18102690127	
队员姓名	1.	魏文憬
	2.	王成钊
	3.	周尚伟

“华为杯”第十五届中国研究生
数学建模竞赛

题 目　　对恐怖袭击事件记录数据的量化分析

摘　　要：

恐怖袭击对全球的发展存在着毁灭性的攻击，并能持久地影响国际政治、公民自由和经济发展，对其进行研究分析显得愈来愈重要。大量数据显示，恐怖袭击事件存在着一定规律，通过对真实案例进行分析得出对未来反恐态势的分析评估有助于提高反恐斗争的针对性和效率。本文使用机器学习算法建模，通过对恐怖袭击事件数据的分析，针对不同事件的属性之间的关联性建立多种数学模型：K-Means 模型、AFW-Kmeans 模型、多层感知机、随机森林模型与 k 近邻模型的结合、支持向量机、时间序列预测框架 Prophet、以及 Seq2Seq 模型，综合应用这些模型实现了对于恐怖袭击事件数据的多个不同维度特征的关联性的挖掘，从而应用这些关联性建立模型对一些恐怖袭击事件预测分析。

针对问题一：

首先对数据集进行了预处理，通过算法和恐怖袭击的广泛定义挑选出重要的特征构建数学模型对样本进行量化分级。为了对比模型的优劣及适用性，我们选用了 K-Means 模型与 AFW-Kmeans 模型分别对恐怖袭击事件按危险等级进行量化分析，实验结果表明 K-Means 模型和 AFW-Kmeans 模型均能实现对危险等级的量化分级，但 K-Means 算法在计算距离时认为所有恐怖袭击事件的输入属性同等重要，没有考虑加入权重，所以在一些难以界定的恐怖事件中分类结果没有 AFW-Kmeans 模型好。综合对比模型的运算效率、危险等级的量化分析效果，我们发现针对问题一的情况，AFW-Kmeans 模型具有更好的适应性，更好的分级效果。

针对问题二：

首先使用多层感知机和随机森林分别进行分类任务的数学建模，对组织或个人名称进行分类。综合对比后选用了随机森林模型结合 kNN 算法的方案，分别对 2015、2016 年发生的未知事件进行相似案件的归类，利用随机森林选取出最有可能的五个嫌疑人，结合表 2 给定的典型事件，使用 kNN 依据距离作为划分嫌疑程度的标准，对给定事件的嫌疑人进行排序。

针对问题三：

对近三年来恐怖袭击事件的样本通过时间序列预测框架 Prophet 进行了建模，通过数据分析，找出了前三名易遭受恐怖袭击的国家。结合数据可视化。对重点地区进行了下一年的恐怖袭击态势的预测，并且针对恐怖袭击发生的规律从反恐态势、蔓延特性、级别分布等方面进行综合分析，给出相应反恐斗争的见解和建议。

针对问题四：

使用支持向量机模型通过选取几组典型特征进行恐怖袭击武器类型的预测，通过对测试数据产生的混淆矩阵分析，该模型的预测效果较好，基本可以准确预测恐怖袭击所使用的武器，从而针对具体情况提出针对性的反恐斗争措施。为使数据集进一步利用，从自然语言处理角度出发，使用 Seq2Seq 模型根据输入数据（样本中 summary 字段文本内容）预测发生恐怖袭击事件的 country_txt 国家信息，对恐怖袭击事件的文本描述内容进行分析挖掘出有用信息。

关键词：AFW-Kmeans 模型，随机森林，k 近邻模型，支持向量机，Seq2Seq 模型

目　录

一. 问题重述

1.1 研究背景

恐怖袭击是极端分子人为制造，不符合国际道义的攻击行为。因其巨大杀伤和破坏力给社会造成人员伤亡、财产损失和心理压力，所造成的恶劣后果成为人类共同的威胁。如何深入分析恐怖袭击事件相关数据定义恐怖袭击事件等级、预测恐怖事件制造者和预计反恐形势的分析成为当下反恐研究热点，这一问题的挑战在于恐怖事件不同于灾难事件的划分，因其常常与例如时机、地域、针对的对象等诸多因素有关。

全球恐怖主义数据库（GTD）是一个开源数据库，包括 1970 年至 2017 年全球恐怖主义事件的信息。本文选取了 GTD 中 1998-2017 年世界上发生的恐怖袭击事件的记录，与许多其他事件数据库不同，GTD 包括在此期间发生的国内以及跨国和国际恐怖事件的系统数据。对于每条 GTD 事件，都可以获得有关事件发生的日期和地点、使用的武器和目标的性质、伤亡人数以及负责的团体或个人的信息。

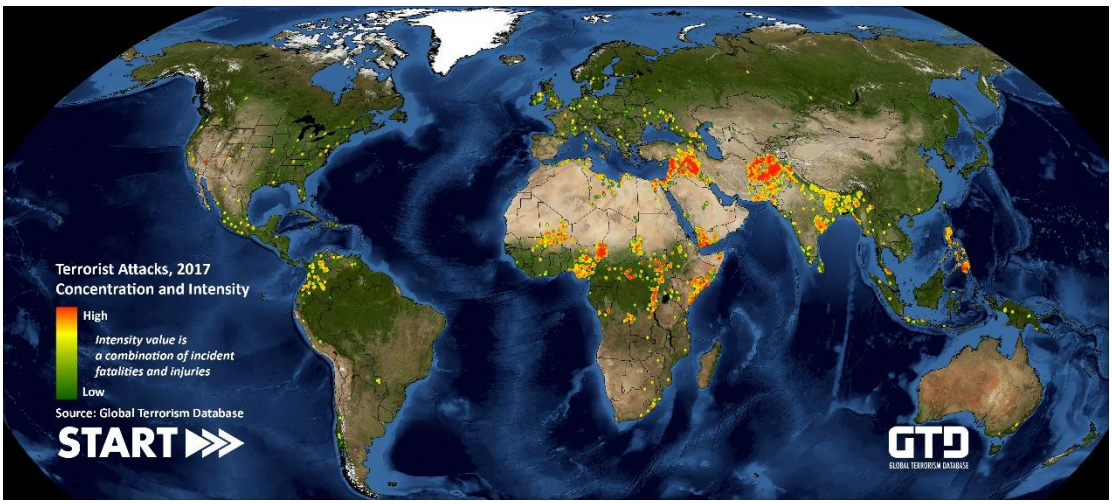

图 1-1 2017 全球恐怖袭击分布情况

1.2 问题提出

本文提出的恐怖主义袭击所有事件总和可以理解为一个二维的数据 $X = R^{i \times j}$，其中 i 代表恐怖袭击事件个数，j 代表恐怖袭击事件某一个描述属性，恐怖袭击数据库由一系列的事件组成，即 $X = \{x_1, x_2, \ldots, x_i\}$，其中 $x_i \in R^{i \times j}$ 为其中的一个恐怖袭击事件。附件 2 中给出恐怖袭击事件的变量的描述，附件 3 提供了对于附件 2 部分内容的摘要。

现要求设计有效的模型与方法完成以下任务：

（1）对于灾难性事件的等级分级，由权威组织或部门选择若干重要指标，依照某些重要指标，主观强制规定分级标准，本问题结合多种因素构建量化分级模型，将事件按危害等级分为一至五级，同时列出近 20 年危害程度最高的十大恐怖袭击事件，并给出表 1 事件的分级。

（2）运用数学建模将 2015、2016 尚未有组织或个人宣称负责的恐怖袭击事件并且可能是同一个恐怖组织或个人在不同时间、不同地点多次作案的若干案件

归为一类，对应的未知作案组织或个人标记不同的代号，并按该组织或个人的危害性从大到小选出其中的前 5 名，记为 1 号-5 号。按照前 5 名嫌疑程度进行排序，完成表 2 恐怖分子关于典型事件的嫌疑度排序。

（3）建立数学模型研究近三年来恐怖袭击事件的相关规律，进而预测下一年全球或某些地区的反恐态势，并将结果以图或者表的形式给出。通过数据分析从而对反恐斗争提出见解和建议。

（4）充分利用数据集，建立数学模型发挥数据的更多作用。

二. 问题分析

全球恐怖主义数据库统计了非国家行为者通过恐惧和胁迫实现政治，经济，宗教或社会目标的暴力案件。根据这个定义，该数据库包括 2015 年查尔斯顿射击等意识形态动机攻击的案例，但不包括同样可怕的但是不是政治驱动的攻击，如 2016 年的极光射击。由于恐怖袭击明显地影响着人类生活，也会对国际政治、公民自由和经济等造成严重冲击，所以对恐怖袭击事件数据进行统计分析和建模对于确定未来防范有重要的参考意义。

针对任务一，先挑选出与危险等级相关的事件属性，通过综合对比K-Means算法和AFW-Kmeans算法的性能后，使用AFW-Kmeans算法先对所有恐怖袭击事件进行聚类分析，然后根据聚类结果得到的危险程度中心挑选关键属性进行加权计算得分，根据得分可以划分每个危险等级中心的危险等级，对于全球10大恐怖事件按照距离危险等级中心最近的事件取出其中前10个即可，分析典型事件的危害级别，计算待分析事件到各个危险等级中心的距离，选取最近的危险等级中心，然后将待分析事件的危险等级划分到这个所属危险等级中心的级别，完成表1内容。

针对任务二，分析数据得知，2015年和2016年度发生的、尚未有组织或个人宣称负责的恐怖袭击事件占了数据的少部分，我们无法利用2015年和2016年的数据进行单独建模。针对此问题，我们使用1998年至2017年的有组织或个人名称的样本集视作有标签的部分，作为接下来建立模型的数据。将未有组织或个人宣称负责的事件作为测试集，利用随机森林分类算法将可能是同一个组织或个人在不同时间、不同地点多次作案的若干案件进行分类预测。

在上一阶段完成的基础上，针对嫌疑程度排序问题，对表2列出的恐袭事件，采用了K近邻分类算法。根据预测事件距离所属类别样本点的欧式距离，进行前五名的排序，完成表2内容。

针对任务三，研究了近三年的恐怖袭击事件数据后，对其进行时间序列建模，依照我们对数据分析的可视化结果，挑选出了恐怖袭击事件的几个热点地区。我们发现这些热点地区存在着一定的共性，比如都信仰伊斯兰教。所以将节假日和恐怖袭击事件进行关联，在与日期相关的几个维度上，分析得出了几个热点地区在2018年的反恐态势。以及结合数据所呈现出的蔓延特性和级别分布等规律，提出了反恐斗争的见解和建议。

针对任务四，为了利用恐怖袭击事件数据集，建立了两种模型。第一，通过数据分析结合互联网资料显示，恐怖组织或个人采用一定的战术，引起人们的恐惧和迫使政府向恐怖分子屈服。这些战术包括抢劫、刺杀、临时爆炸装置等，因此十分有必要对恐怖分子所携带的武器进行分类，以做好针对该武器的防范措施。使用数据进行了 SVM 算法模型的构建，从而得到未知事件可能采用的武器类

型，以有效提高反恐斗争的针对性。第二，从自然语言处理角度出发，我们对数据集中的摘要信息和国家名称这两个属性构建 Seq2Seq 模型，来达到我们通过摘要信息便可产生事件发生的地点的目的。

三. 模型假设与符号说明

3.1 模型假设

（一）输入空间中的所有样本服从一个隐含未知的分布，训练数据所有样本都是独立地从这个分布上采样而得，即数据满足独立同分布假设。

（二）假设如果在训练样例中能够很好地逼近目标函数，那么在未见实例中也能很好的逼近目标函数。

3.2 符号说明

表 3.1 符号说明

N	样本数量
d	距离
δ	算法终止阈值
$G_t(x)$	第 t 个决策树模型
D	训练集
$y(t)$	时间序列模型
$h(t)$	节假日模型
$s(t)$	季节性模型
$K(x,z)$	核函数
$f(x)$	决策函数
$\mathcal{H}$	特征空间
$\phi(x)$	映射函数

四. 模型的建立及求解

4.1 任务一模型的建立及求解

4.1.1 数据预处理

恐怖袭击事件的数据中有些特征与问题求解无关且有数据不一致等诸多问题，严重影响到数据挖掘建模的执行效率和结果。数据预处理过程如下：

第一步，计算每个属性的空值总数和百分比。结果显示，许多属性的缺失值超过 50%，对于缺失值较多的属性，由此选择缺失值小于 20%且不与其他属性相似的属性列表用于接下来的建模。

第二步，因为对缺失值进行分类会给模型造成一定的分类障碍，所以我们修复了缺失值。其中包括原始数据空白值，-9，-99。为了保持一致性，规定-1 用于数字的分类属性，Unknown 用于文本的分类属性，数字属性将替换为 NAN。部分属性由于平均值不稳定且受异常值的影响较大，我们采用中值进行插补。

第三步，由于很多属性是用 Yes/No/Unknown 进行标记的，这种标记方法不利于我们进行数据分析，用 1，0，-1 分别表示“Yes”，“No”和“Unknow“。不止这些，用数字标签替换数据集中的其他文本属性以改进原始数据集。

第四步，由于不同评价指标往往具有不同的量纲，数值间的差别可能很大，不进行处理可能会影响到数据分析的结果。为了消除指标之间的量纲和取值范围差异的影响，需要进行标准化处理，将数据按照比例进行缩放，使其落入一个特定的区域，便于进行综合分析。实验预处理部分采用的是最小-最大规范化，对原始数据进行线性变换，将数值映射到[0,1]之间。进行数据规范化的特征有：'nperpcap'，'nkill'，'nkillus'，'nkillter'，'nwound'，'nwoundus'，'nwoundte'

转换公式如下：

$$x^* = \frac{x-min}{max-min} \tag{4.1.1}$$

第五步：用随机森林模型挑选出来了前 20 个重要特征，根据互联网查找资料所显示的“恐怖袭击事件”的广泛定义，作为接下来每个任务的具体求解参照，删除了不属于此范畴内的特征，利用符合每个任务题意的特征进行了建模。前 20 个重要特征有：

表 4.1 随机森林产生的 20 个特征

iyear	region_txt_South Asia	INT_IDEO_txt_YES	natltyl_txt_Iraq	INT_ANY_txt_YES
region_txt_Middle East&North Africa	country_txt_Iraq	country_txt_Somalia	iday	imonth
region_txt_Sub-Saharan Africa	INT_LOG_txt_UKNOWN	INT_IDEO_txt_UKNOWN	nkill	INT_ANY_txt_UKNOWN
country_txt_Pakistan	natltyl_txt_Somalia	natltyl_txt_India	natltyl_txt_Pakistan	natltyl_txt_Nigeria

查询因特网上有关资料，分别针对袭击者信息与伤亡损失两大方面对恐怖袭击事件进行分级。针对两个方面建立重要特征体系如下。

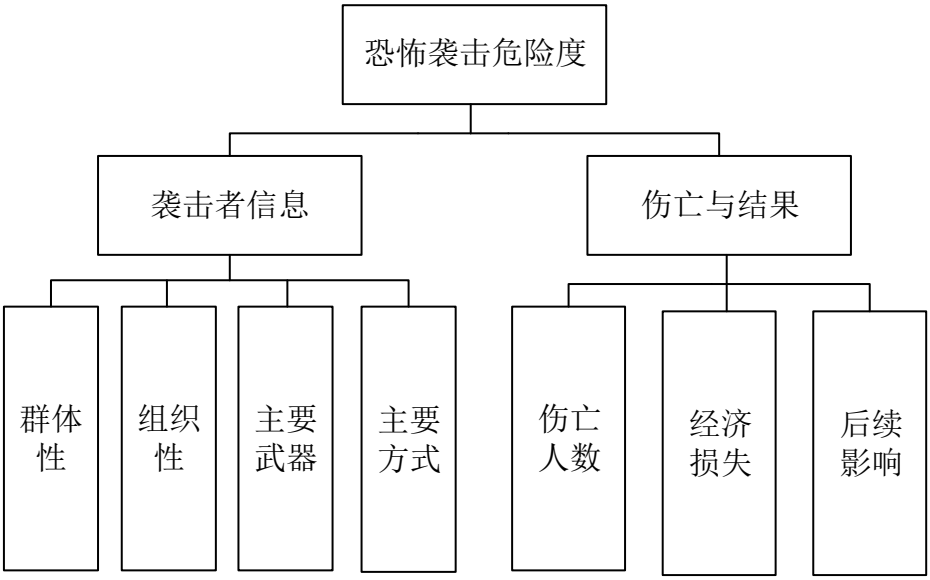

图 4-1 特征评估体系

4.1.2 K-Means 算法、AFW-Kmeans 算法

（1）K-Means

K-Means 是对距离聚类中的一种方法，通过计算距离的大小计算恐怖事件的相似性程度，从而实现相似数据的合并。

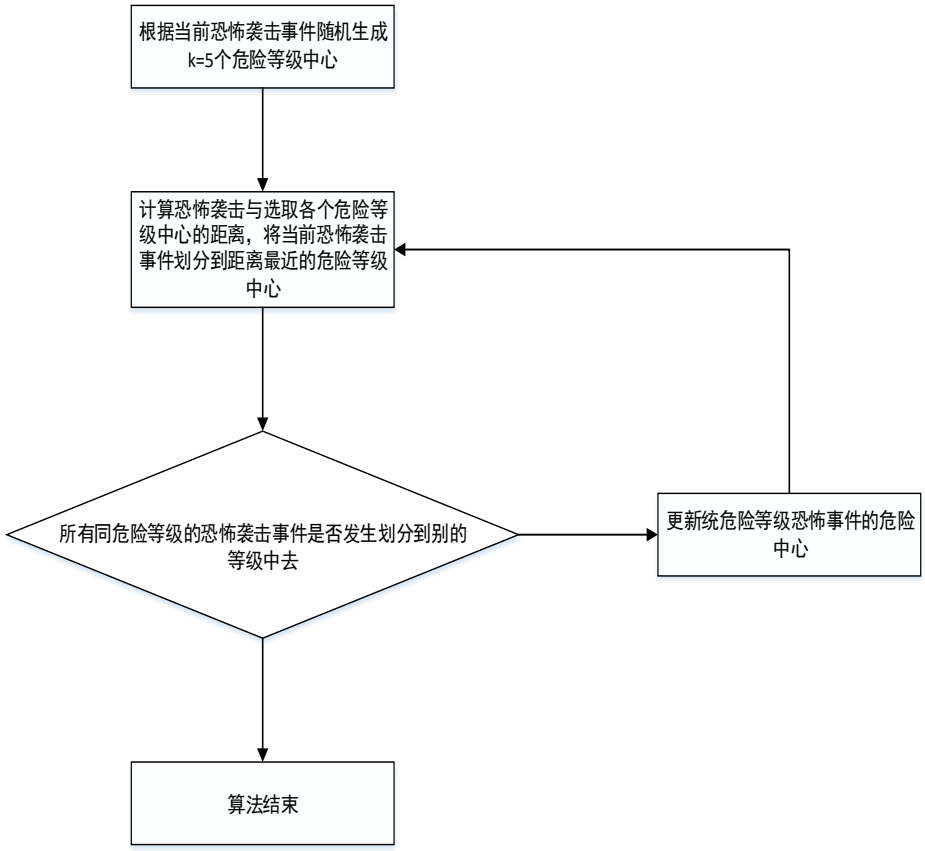

图 4-2 K-Means 算法流程

k的选择设置为 5，因为依照题目需要根据危害程度分为 5 级。

算法以距离作为危害等级相似性度量的标准，采用欧氏距离来计算恐怖事件间的距离。下面给出欧式距离的计算公式：

$$\mathrm{dist}(x_i, y_j) = \sqrt{\sum_{d=1}^{D}(x_{i,d} - x_{j,d})^2} \qquad (4.1.2)$$

其中，D表示恐怖事件的危害程度的度量属性，这里选择了 10 个属性，$D = 10$。

算法聚类过程中，每次迭代，对应的危险等级中心需要重新计算（更新）：对应该危险等级中所有恐怖事件对应属性的均值，即为更新后该危险等级中心。定义了 5 个危险等级，这里的k位于[1-5]，第k个危险等级中心为$\mathrm{Center_k}$，则中心更新方式如下：

$$\mathrm{Center}_k = \frac{1}{|C_k|}\sum_{x_i \in C_k} x_i \qquad (4.1.3)$$

其中，C_k表示第k个危险等级中心，$|C_k|$表示第 k 个危险等级所有恐怖事件的个数，这里的求和是指该等级C_k中所有元素在每列属性上的和，这里选取了 10 个

关键属性，因此C_k也是一个含有 10 个属性的向量，表示为：

$$\mathrm{Center}_k = (\mathrm{Center}_{k,1}, \mathrm{Center}_{k,2} \ldots, \mathrm{Center}_{k,10})$$

K-Means 算法需要不断地迭代来重新划分中心，并更新危险等级中心，迭代终止的条件采用误差平方和准则函数，函数模型如下：

$$\Delta J = \sum_{k=1}^{K}\sum_{x_i \in C_k} dist(x_i, Center_k) \qquad (4.1.4)$$

表 4.2 K-Means 算法步骤

算法 1：K-Means 算法
输入：清洗过的数据集 1. 随机选取五个危险等级中心，根据公式（4.1.2）计算$\mathrm{dist}(x_i, Center_k)$； 2. 将$x_i$划分至离其最近的危险等级中心所在的危险等级； 3. 依据公式（4.1.3）计算该等级的危险等级中心； 4. 计算两次迭代的差值ΔJ； 5. 直到$\Delta J < \delta$，δ为设定的算法终止阈值。 **输出：**5 个聚类，使每一个类到其所在聚类中心的欧氏距离最小

（2）AFW-Kmeans 算法

AFW-Kmeans[1]是对距离聚类中的一种方法，通过计算距离的大小衡量恐怖事件的相似性程度，从而实现相似数据的合并。

k的选择为 5，因为需要根据危害程度分为 5 级。

设当前迭代后的对象划分成 5 个危害等级，每个聚类中的恐怖事件个数为$n_1, n_2, \ldots, n_5$，则所有k个在第j维上的类内距离之和为d_n，m_{kj}为聚类k在第j维属性上的均值：

$$d_n = \sum_{k=1}^{5} \sum_{i=1}^{n_k} (x_{ij-m_{kj}})^2 \tag{4.1.5}$$

所有k个聚类在第j维恐怖事件的属性上的类间距离之和为d_n，m_{kj}为聚类k在第j维属性上的均值：

$$d_w = \sum_{k=1}^{5} (m_{kj} - m_j)^2 \tag{4.1.6}$$

依据当前结果，计算属性恐怖事件的属性d对聚类的贡献度：

$$c_d = \frac{d_w}{d_n} \tag{4.1.7}$$

w_d表示在恐怖事件属性d上的权重，特征权重是根据各恐怖事件属性的贡献度计算得出的，

$$w_{d=} c_d / \sum_{d=1}^{10} c_d \quad , w_d \in [0,1], \sum_{d=1}^{10} w_d = 1 \tag{4.1.8}$$

算法以距离作为危害等级相似性度量的标准，采用加上权重的欧氏距离来计算恐怖事件间的距离。下面给出欧式距离的计算公式：

$$\mathrm{dist}(x_i, y_j) = \sqrt{\sum_{d=1}^{10} w_d (x_{i,d} - x_{j,d})^2} \tag{4.1.9}$$

选取初始聚类中心C_i，mean 为恐怖事件在各属性上的均值，v作为均方差，$2v/k - 1$与$2v/k$均作为偏移因子：

$$\begin{cases} \left\{mean \pm \frac{2v}{k-1} \times j, j = 1, \ldots, \frac{k}{2}\right\} \cup \{mean\}, \text{k为奇数} \\ \left\{mean \pm \frac{2v}{k} \times j, j = 1, \ldots, \frac{k}{2}\right\}, \text{k为偶数} \end{cases} \quad (4.1.10)$$

算法需要不断地迭代来重新划分危险等级中心，并更新新的危险等级中心，迭代终止的条件采用误差平方和准则函数，函数模型如下：

$$\Delta J = \sum_{k=1}^{5} \sum_{x_i \in C_k} dist(x_i, C_k) \quad (4.1.11)$$

表 4.3 AFW-Kmeans 算法步骤

算法 2：AFW-Kmeans 算法

输入：清洗过的数据集，聚类个数$k = 5$

1. 依据公式（4.1.10）构造初始危险等级中心C；
2. 特征权重初始化$w_d = \frac{1}{m}, d = 1,2,\ldots,m$；
3. 依据公式（4.1.4）计算每一个恐怖袭击事件到危险等级中心的加权欧式距离，按距离最小原则为每个恐怖袭击事件分配聚类号，重新计算危险等级中心；
4. 判断每一危险等级中是否都包含恐怖袭击事件，某一危险等级中心无恐怖事件说明本次聚类非常密集，偏移因子过大，将偏移因子减半，重新选取危险等级中心；
5. 根据迭代结果，按照恐怖事件类内紧密，类间远离的原则调整每个恐怖事件属性的特征权重；
6. 计算两次迭代的差值ΔJ，直到$\Delta J < \delta$，δ为设定的算法终止阈值。

输出：k个聚类，使每一个类到其所在危险等级中心的加权欧氏距离最小

4.1.3 模型求解及结果分析

针对于上述两种模型的算法步骤及实现原理，通过 Python 编程实现上述两种关于恐怖事件危害程度的分级，通过观察危险等级中心的值选取其中重要的伤亡人数、死亡人数、财产损失的加权求和进行排序，即可得到危险等级中心的危险等级。使用两种算法的聚类结果依据公式（4.1.8）计算危险等级 1 的事件距离第一类聚类中心的距离，然后排序选出其中的前 10，如下表所示：

表 4.4 全球 10 大恐怖事件

K-Means	AFW-Kmeans
200109110005	200109110005
200409010002	200409010002
199808070002	199808070002

200507070002	200507070002
200403110001	200403110001
201607140001	201607140001
199909090002	199909090002
200907050020	200907050020
201403010012	201412160002
200210120004	201304150002

对比两种算法对于题目给出的事件的危险级别的预测结果：

表 4.5 典型事件的危害级别（题目中为表 1）

事件编号	K-Means 危害级别	AFW-Kmeans 危害级别
200108110012	1	1
200511180002	1	2
200901170021	1	1
201402110015	3	3
201405010071	1	1
201411070002	1	2
201412160041	4	4
201508010015	1	1
201705080012	5	5

我们参照网上给出的全球二十大恐怖事件，通过观察我们的模型预测的十大恐怖事件是否在这二十大恐怖事件中来评价模型建立好坏的依据，结果显示，AFW-Kmeans 产生的危害级别对于十大恐怖事件预测较为准确。

K-Means 算法虽然具有简单的算法逻辑、计算速度相较与 AFW-Kmeans 算法较快，但是在进行危险等级划分时认为所有输入特征重要性一样，对于聚类中心会有所偏移，进而影响到危险等级最高的恐怖事件划分以及对于后来模型预测典型事件的危险等级的计算。危险等级的划分结果与真实的二十大恐怖事件对比发现，基本上十大恐怖事件都在我们所划分的第一等级中。在事件预测时，我们的 AFW-Kmeans 算法由于给等级划分加入了权重并且对危险等级中心的选取上加入了偏移因子所以具有更好的效果，使得危险等级的划分更加具有区分度，各个危险等级分层效果更加明显，我们分析发现产生这种现象的原因是：

1．在计算恐怖袭击事件与已知危险等级中心的距离上加权，可以自动调整我们的权值 w，从而可以考虑当前恐怖袭击事件中某一属性特征的重要性，例如伤亡人数、死亡人数、财产损失明显时比较重要的评估标准，因为加入了权值计算，使得我们的同等级的危险事件具有更好的聚合效果。

2．对于危险等级标准主要依据与聚类中心的距离进行计算，AFW-Kmeans 使用（4.1.9）的计算公式，给危险等级中心计算时加入了一些偏移因子，使得我们选取的危险等级中心性能更加具有代表性，便于后续的危险等级划分任务的进行。

AFW-Kmeans 相对于 K-Means 提出了一种简单，易于使用的危险等级中心的选取方法，该方法能真实描述危险等级中心在各属性上的分布情况，减少算法迭代

次数,提高算法的稳定性;在恐怖袭击事件对象相似性度量中引入自适应特征权重,根据每次迭代的结果,在每个恐怖袭击事件属性上计算不同危险等级之间距离之和与同等级恐怖袭击事件距离之和的比值,对欧氏空间进行一定程度的归约,消除恐怖袭击事件相关程度不大的属性对聚类的影响,以便更真实地反映同等级恐怖袭击事件的相似程度，从而使得判断未知恐怖袭击事件等级更加有效。

4.2 任务二模型的建立及求解

由于数据集中部分数据的缺失，有多起恐怖袭击事件尚未确定作案者。任务二要求确定 2015 年、2016 年度发生的、尚未由组织或个人宣称负责的恐怖事件，按照一定规律找出相似事件的类别，将其归为一类。在此基础上，选出组织或个人的危害程度排名前 5 的完成表 2 任务。本任务首先对未知事件通过随机森林算法建模进行分类任务，通过数据分析得出排名前 5 的组织，标记为 1-5 号，之后通过 k-近邻算法建模找到与未知恐怖事件有嫌疑的 5 个嫌疑人。

4.2.1 随机森林

随机森林首先由多棵分类回归树（Classification And Regression Tree,CART）构成，采用自助采样法(boostrap sample)得到差异化训练样本数据集，对于单颗决策树，在选择特征构建时采用随机子空间划分(random subspace method)策略，也就是说随机选择部分属性构成单棵决策树，并从中挑选最佳属性进行属性分裂。

因此，结合本任务的具体要求随机森林的训练过程可以总结如下。

表 4.6 利用随机森林进行未知组织的预测的算法步骤

算法 3：Random Forests 预测分类算法

输入： 1.训练集$\boldsymbol{D}$（1998 年至 2017 年有组织名称的样本）

2.待测样本为 2015 年至 2016 年未标有组织的数据

输出： 最终的强分类器$\mathbf{f(x)}$

1) 对于 $\boldsymbol{t = 1,2,\cdots,T}$ （弱分类器迭代次数实验设置为 100）；

a) 对训练集进行第 t 次随机采样，共采集 m 次，得到包含 m 个样本的采样集$\boldsymbol{D_t}$；

b) 用采样集$\boldsymbol{D_t}$训练第$\boldsymbol{t}$个决策树模型$\boldsymbol{G_t(x)}$，在训练决策树模型的节点的时候，在节点上所有的样本特征中选择一部分样本特征，在这些随机选择的部分样本特征中选择一个最优的特征来做决策树的左右子树划分；

2) $\boldsymbol{T}$个弱学习器投出最多票数的类别或者类别之一为最终类别。

4.2.2 多层感知机

多层感知机（MLP）是一种前向结构的人工神经网络，映射一组输入向量到一组输出向量。MLP 可以被看做是一个有向图，由多个节点层组成，每一层全连接到下一层。除了输入节点，每个节点都是一个带有非线性激活函数的神经元。实验中使用反向传播算法训练 MLP，预测全球所有地区的恐怖袭击风险。

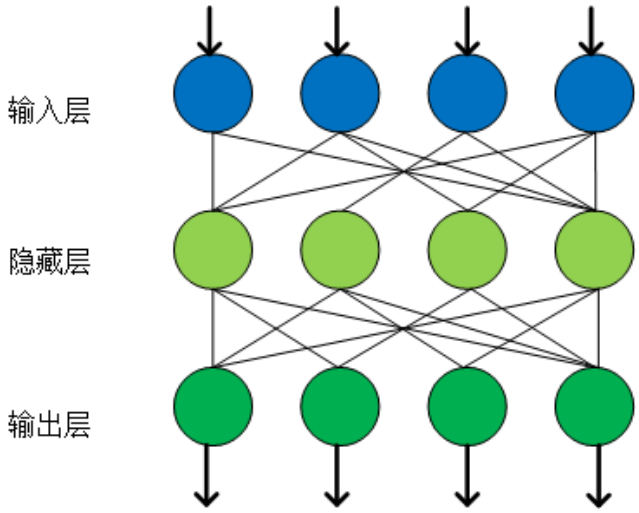

图 4-3 多层感知机结构示意图

在本任务中，将 1998 年-2017 年的恐怖袭击事件进行有组织或个人宣称的恐怖袭击事件和未有组织或个人宣传的恐怖袭击事件进行划分，gname 字段不为空的样本作为训练集，将数据集中的 gname 字段作为训练标签。2015、2016 年没有组织或个人宣称的事件作为测试集。统计得出有名称的恐怖袭击事件有个 49852 条样本，未有组织宣称的事件有 59049 条。设置隐层神经元个数设置为 64，进行多层感知机模型分类任务的构建。

4.2.3 k 近邻算法

k 近邻（k-Nearest Neighbor，简称 kNN）是一种基于某种距离度量对训练样本进行分类的算法。由于按照距离对样本进行度量，所以可以找出本任务预测恐怖袭事件的嫌疑人。具体解题思路是：在表 2 给定的 10 个测试样本中，根据度量距离找出在特征空间中的 5 个最相似样本，然后根据找到的这 5 个最邻近样本的信息进行分类预测，在分类预测过程中，通常遵循“投票法则”，即在这 10 个样本中出现最多的类别作为预测结果。通过 sk-learn 中的 predict_proba 函数可以输出所有类别的预测结果，我们按照降序顺序完成对表 2 的填写。

在解题过程中，使用的距离为欧式距离，公式如下：

$$d(x,y) = \sqrt{\sum_{k=1}^{n}(x_k - y_k{}^2)} \qquad (4.2.1)$$

表 4.7 KNN 预测典型事件的嫌疑度的算法步骤

算法 4：kNN 预测嫌疑度算法
输入： 10 个待预测样本数据
1.计算 10 个待预测样本与各个训练数据（1998 年至 2017 年的）之间的距离； 2.按照距离的递增关系进行排序； 3 选取距离最小的 5 个点； 4 确定前 5 个点所在类别的出现频率； 5.返回前 5 个点中出现频率最高的类别作为测试数据的预测分类。若此结果与

代号 1-5 嫌疑人有重合，则该嫌疑人在表 2 中标为 1。
输出：10 个样本所属的类别以及对于所有类别的预测概率结果，根据概率值对表 2 进行填写。

4.2.4 模型求解及结果分析

利用 Python 进行建模编程，综合对比多层感知机和随机森林的实验结果，结果发现多层感知机的实验效果并没有随机森林的效果好。可能由于训练样本数量太小，模型不能很好的学习获得参数；隐藏层只有一层，结构过于简单，而此样本又具有较多的特征。使用随机森林得到的实验结果位列前五名分别是 Muslim extremists、Gunmen、Militants、Separatists、Maoists。实验结果如图所示。

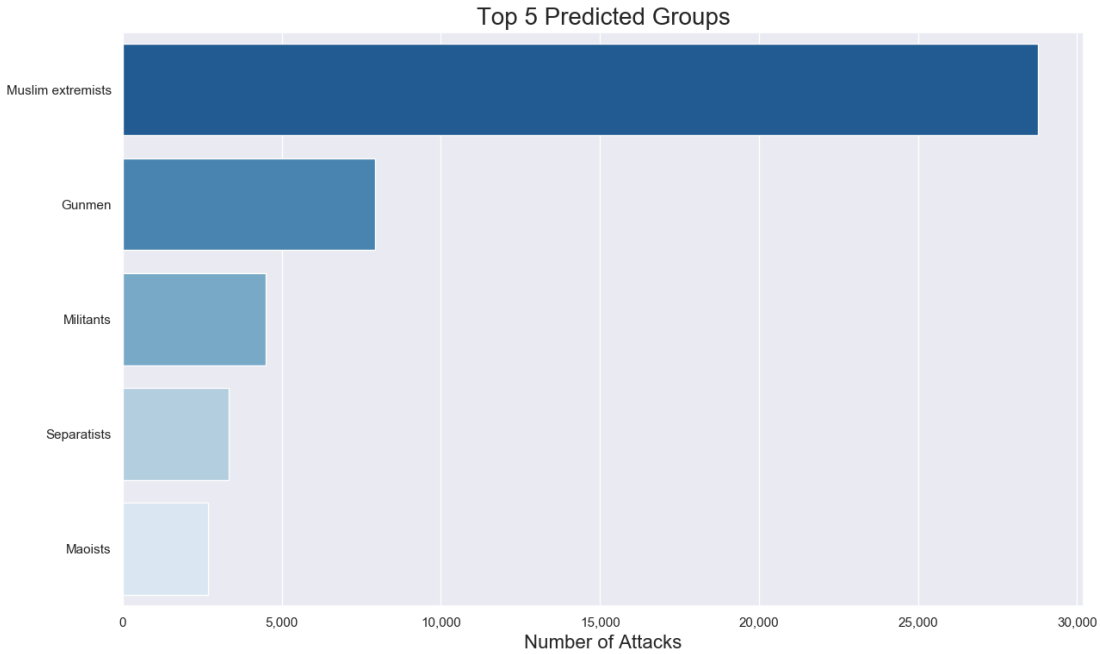

图 4-4 组织或个人危害性前 5 名

随机森林算法作为一个可以高度并行化的算法，在针对 11 万条数据处理的时候显示了卓越的性能。针对此问题，随机森林算法的主要优点有：1）训练可以高度并行化，对于大数据量的样本训练速度有优势。2）由于可以随机选择决策树节点划分特征，在针对样本特征维度很高的时候，仍然能高效的训练模型。3）在训练后，可以给出各个特征对于输出的重要性，在数据预处理部分，我们利用了此优点，挑选出了重要特征作为不同问题的参考。4）由于采用了随机采样，训练出的模型的方差小，泛化能力强。5）相对于 Boosting 系列的 Adaboost 和 GBDT，随机森林算法实现比较简单。6）对部分特征缺失不敏感，GTD 数据集中有大量的数据缺失。但是在训练过程中也存在一些缺点： 2) 取值划分比较多的特征会影响随机森林算法产生的决策，从而影响拟合的模型的效果。

通过实验结果将 1 号嫌疑人标记为 Musilim，2 号嫌疑人标记为 Gunmen，3 号嫌疑人标记为 Militans，4 号嫌疑人标记为 Separatists,5 号嫌疑人标记 Maoists。

再利用 kNN 算法进行建模求解，通过概率值的降序排列得到题目中的表 2（表 4.8）。

表 4.8 恐怖分子关于典型事件的嫌疑度

	1 号嫌疑人	2 号嫌疑人	3 号嫌疑人	4 号嫌疑人	5 号嫌疑人
201701090031	2	1	3	5	4
201702210037	1	2	3	5	4
201703120023	2	3	1	5	4
201705050009	3	1	2	4	5
201705050010	3	1	2	3	5
201707010028	1	3	2	5	4
201707020006	2	1	3	5	4
201708110018	1	2	3	5	4
201711010006	2	1	3	5	4
201712010003	2	1	3	5	4

4.3 任务三模型的建立及求解

任务三要求研究近三年（2015 年--2017 年）来恐怖袭击事件发生的主要原因、时空特性、蔓延特性、级别分布等规律，进而得到 2018 年全球或某些重点地区的反恐态势。

针对该任务，我们分别建立了时间序列模型，依据附件 1 和结合因特网上的有关信息进行数据分析。

4.3.1 时间序列模型的建立

针对该问题，有多种方法可以达到时间序列预测的结果。我们采用了时间序列预测框架 Prophet[2]。对由数据分析结果显示的重点地区进行了下一年的恐怖袭击态势的预测。针对恐怖袭击发生的规律从主要原因、蔓延特性、级别分布等方面做了数据分析可视化，从而实现了此任务的目标，并在结果分析中对反恐斗争提出了见解和建议。

（1） 时间序列模型

在针对于该问题的解决，应用了 Facebook 开源的时间序列预测算法。Prophet 是一种基于加性模型预测时间序列数据的程序，其中非线性趋势能够拟合每一年，每一周和每一日。它对于缺失数据和趋势的变化是非常有效的，通常很好地处理异常值，并且适用于具有强烈季节性影响的历史数据的时间序列。因为从数据分析中得出易发生恐怖袭击的国家大多数都是信奉伊斯兰教的国家，教徒们通常会根据节日做相应的祷告活动，我们大胆推断发生恐怖袭击事件的日期与他们节假日之间有着一定的关联，结果显示也证实了我们猜想的正确性。Prophet 是一个可加回归模型，也就是把模型分为趋势模型、周期模型、节假日以及随机噪声的叠加。它由四个组成部分：

1. 一个分段的线性或逻辑增长曲线趋势。Prophet 通过提取数据中的转变点，自动检测趋势变化。
2. 一个按年的周期组建，使得傅里叶级数建模而成。
3. 一个按周的周期组建，使得虚拟变量建模而成。
4. 用户设置重要节日表。

时间序列模型可以分解为三个主要组成部分：趋势、季节性和节假日。它们

按如下公式组合：

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t \quad (4.3.1)$$

其中，$g(t)$用于拟合时间序列中的分段线性增长或逻辑增长等非周期变化。s(t)用于周期变化（如：每周/每年的季节性）。$h(t)$代表了非规律性的节假日效应，这个参数是由用户造成的。ϵ_t误差项用来反映未在模型中体现的异常变动。

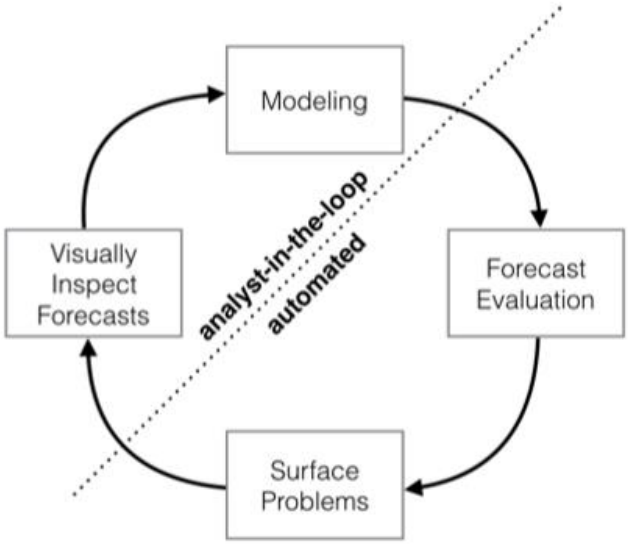

图 4-5 时间序列预测框架

4.3.1.1 季节性趋势

由于时间序列中有可能包含多种周期类型的季节性趋势，因此，傅里叶级数可以用来近似表达这个周期属性，公式如下：

$$s(t) = \sum_{n=1}^{N} \left(a_n cos\left(\frac{2\pi nt}{P}\right) + b_n \sin\left(\frac{2\pi nt}{P}\right) \right) \quad (4.3.2)$$

其中，P表示某个固定的周期（例如用“天”做单位统计的数据中，年数据的P设为 365.25，周数据的P设为 7）。$2N$则表示我们希望在模型中使用的某种周期的个数，较大的N值可以拟合出更复杂的季节性函数，但这也会带来更多的

过拟合问题。按照经验值，年周期的N通常取 10，周周期的N通常取 3。当将$s(t)$中的所有季节性时间序列模型组合成一个向量$X(t)$，那么最终的季节性模型为：

$$s(t) = X(t)\beta \tag{4.3.3}$$

其中，$\beta \sim Normal(0, \sigma)$，以此提高季节性模型的平滑性。

4.3.1.2 节假日模型

很多实际经验告诉我们，节假日或者是一些大事件都会对时间序列造成很大影响，而且这些时间点往往不存在周期性。对这些点的分析是极其必要的，甚至有时候它的重要度远远超过了平常点。

鉴于每个节假日（或者某个已知的大事件）的日期与影响程度存在差异，节假日模型将不同节假日在不同时间点下的影响视作独立的模型。同时为每个模型设置了时间窗口，这主要是考虑到节假日的影响有窗口期（例如中秋节的前几天与后几天），模型将同一个窗口期中的影响设置为相同的值。例如，i表示节假日，D_i表示窗口期中包含的时间t，则节假日模型$h(t)$可表示为：

$$h(t) = \sum_{i=1}^{L} k_i 1(t \in D_L) \tag{4.3.4}$$

其中，k_i表示窗口期中的节假日对预测值的影响。同季节性趋势的模型，这里可以定义为：

$$Z(t) = [1(t \in D_1), \ldots, 1(t \in D_t)] \tag{4.3.5}$$

那么，$h(t) = Z(t)\kappa$，其中，$\kappa \in Normal(0, v^2)$

4.3.2 模型求解及结果分析

通过对时空特性进行建模。分析得到了下一年（2018 年）全球或某些重点地区的反恐态势。通过数据分析得出 1998 年至 2017 年排名前 20 易受到恐怖袭击的国家，如图所示。

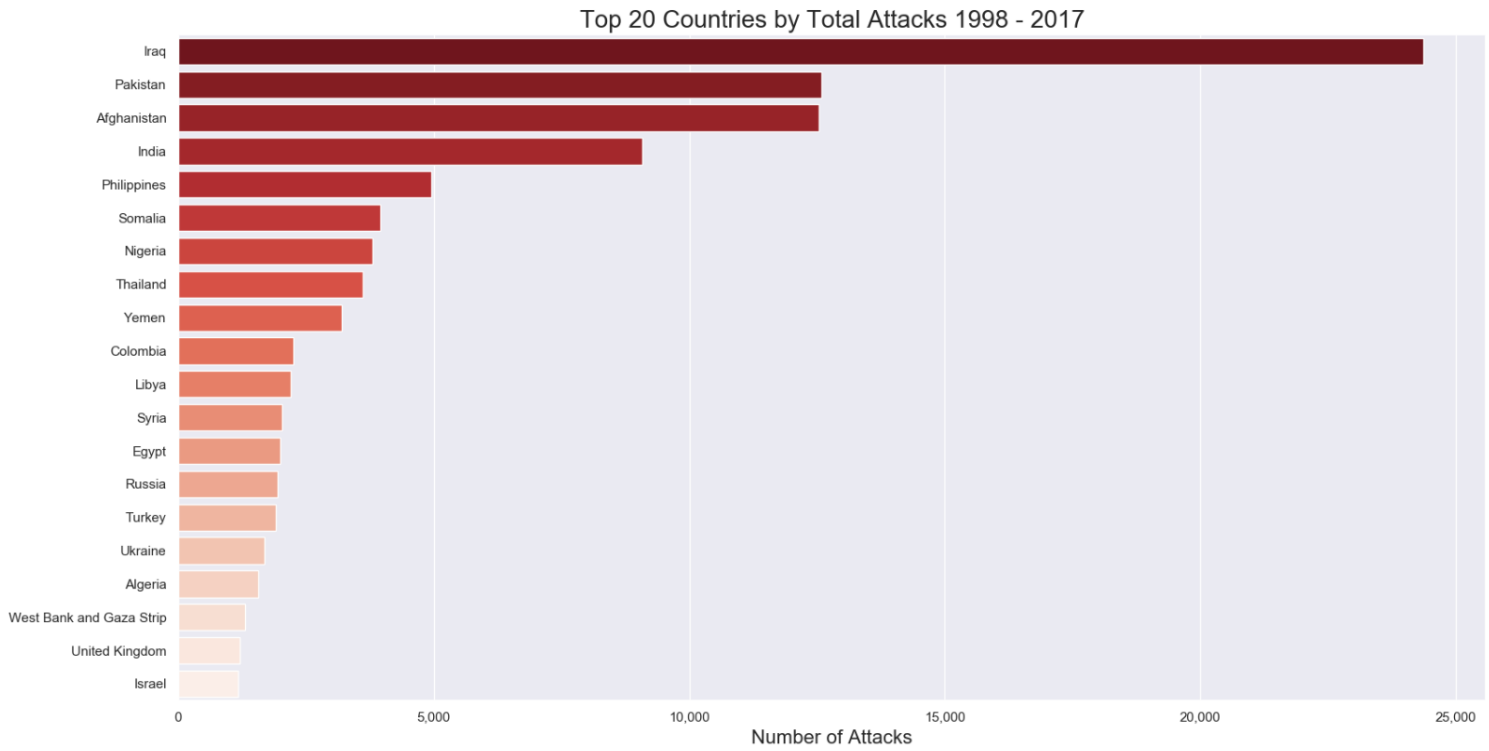

图 4-6 易遭受恐怖袭击国家排名

前五名的是 Iraq、Pakistan、Afghanistan、India、Philippines。所以，在解决研判下一年某些重点地区的反恐态势问题中，我们挑选了排名前三的国家，分别是伊拉克共和国、巴基斯坦伊斯兰共和国、阿富汗伊斯兰共和国进行时序建模，从而预测下一年（2018 年）的反恐态势。（由于图片像素问题，导致论文中图片里的字体不是很容易识别，已将所有图片上传至附件。）

针对于 Prophet 适用的场景和本任务的描述，相似的情况有： 1、事先知道的以不定期的间隔发生的重要节假日；2、缺失的历史数据或较大的异常数据的数量在合理范围内；3、有历史趋势的变化；4、对于数据中蕴含的非线性增长的趋势都有一个自然极限或饱和状态。Prophet 是一种类似 generalized additive model(GAM)的模型，不同于以往的时间序列预测模型（例如 ARIMA），Prophet 模型将预测问题视作曲线拟合问题。时间序列中无需有一个固定的周期，也不需要在拟合前对缺失值进行填补，这是传统的（例如 ARIMA）模型所办不到的。

以下是对伊拉克、巴基斯坦、阿富汗的恐怖袭击事件和节假日之间创建时间序列模型求解结果。

- **伊拉克**

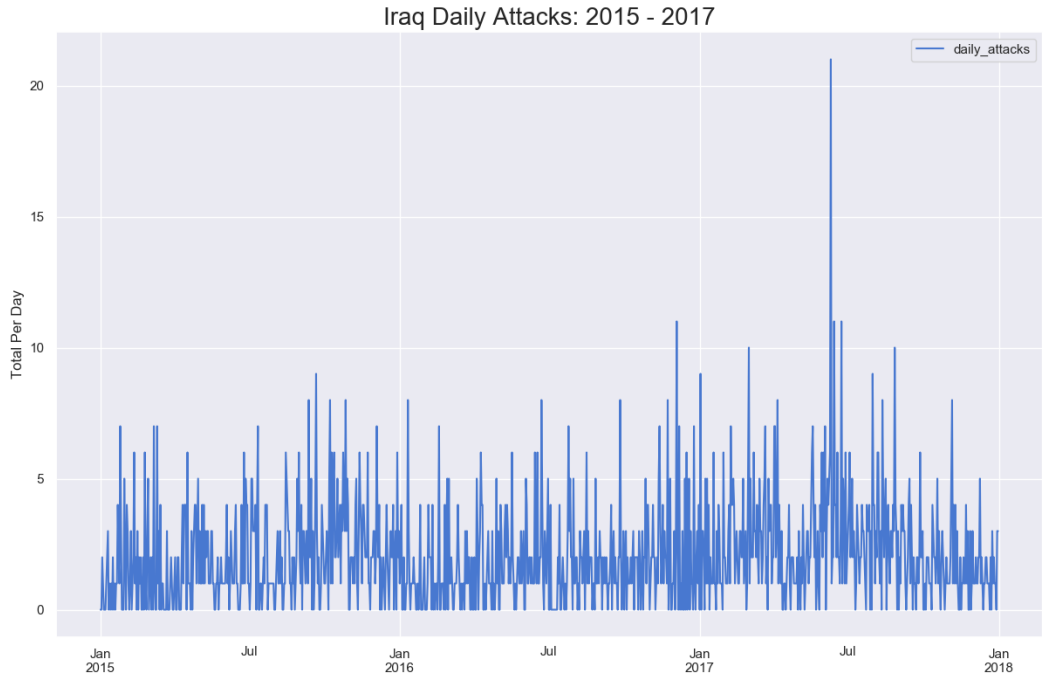

图 4-7 伊拉克每日所遭受的恐怖袭击数量

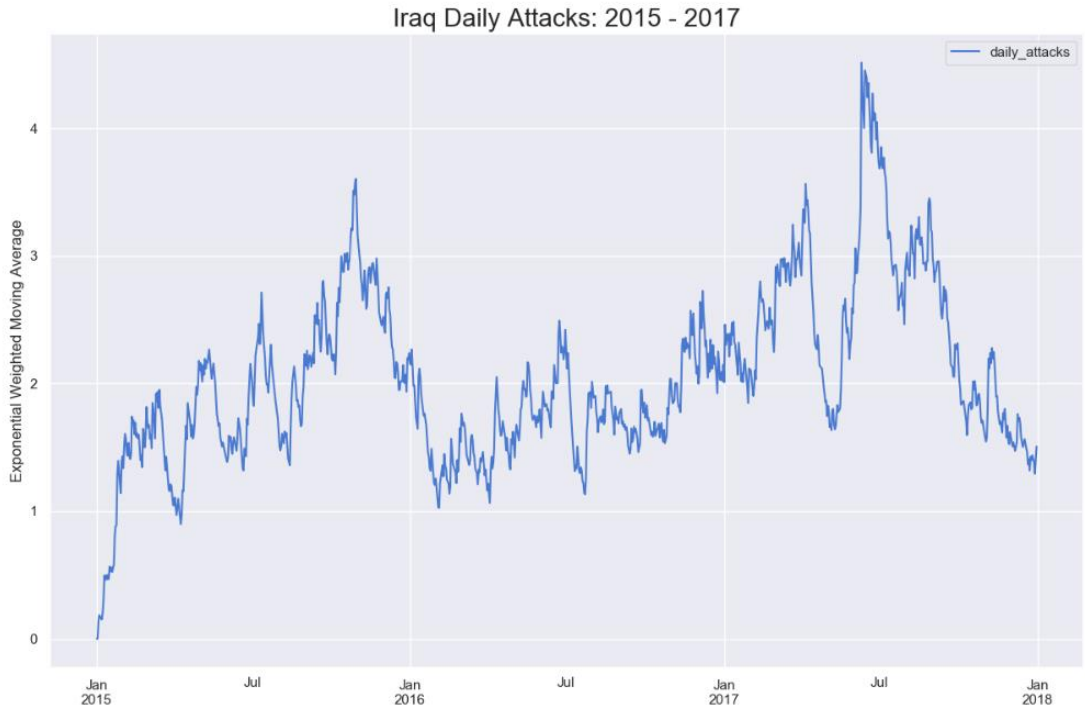

图 4-8 伊拉克每日所遭受的恐怖袭击数量——平滑后

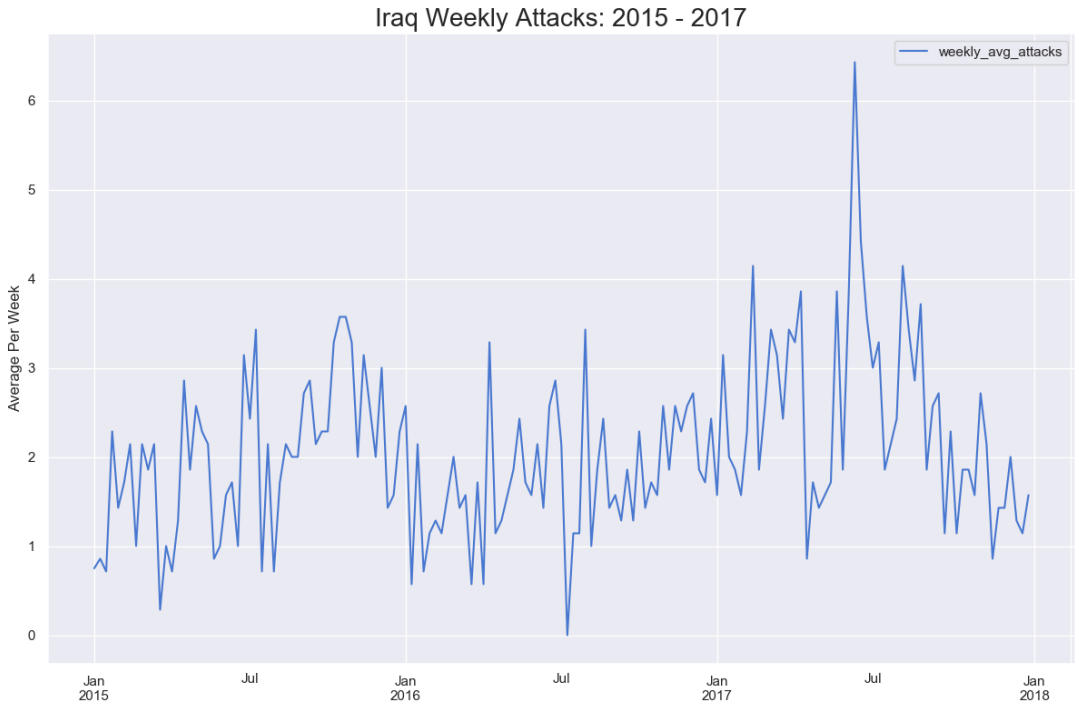

图 4-9 伊拉克每周所遭受的恐怖袭击数量

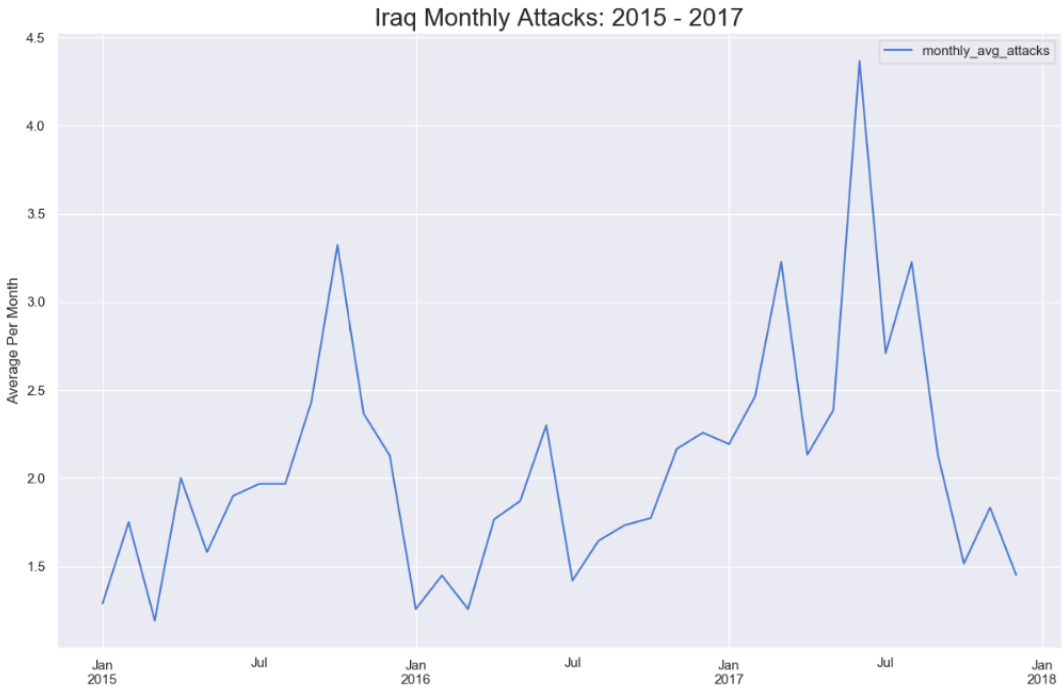

图 4-10 伊拉克每月所遭受的恐怖袭击数量

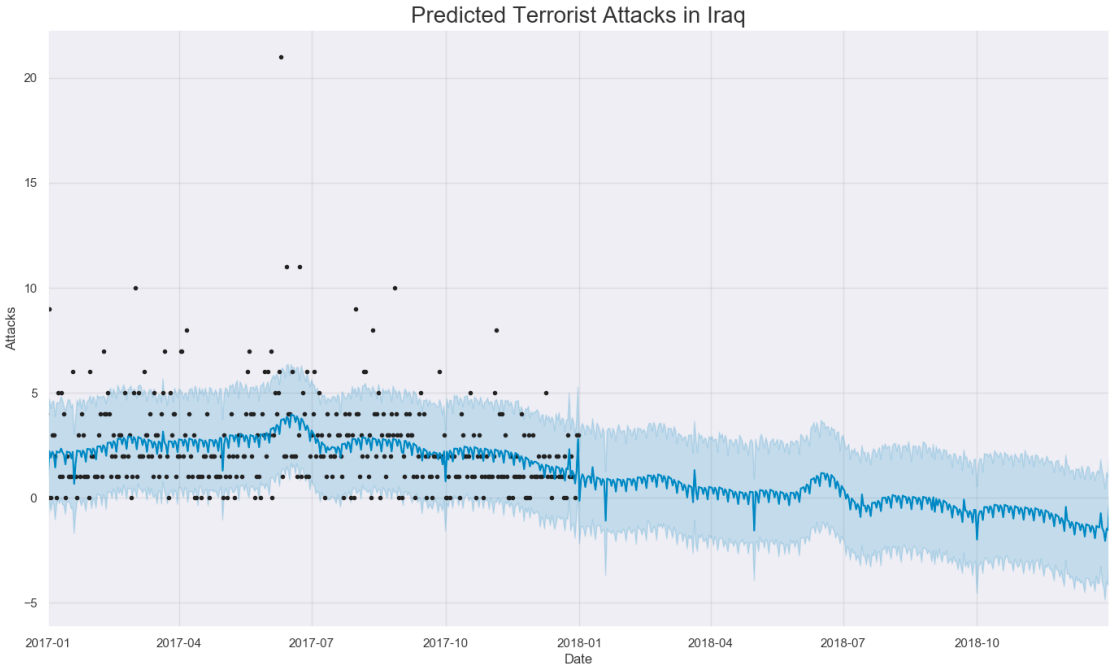

图 4-11 预测 2018 年伊拉克地区恐怖袭击趋势

其中蓝色线为预测结果，黑色点为原数据点。将 2018 年伊拉克发生恐怖袭击的次数的预测图进行放大，可以看到在 2018 年 3、6 月份会有明显上升，而在 5、10 月份之后会有明显的下降，模型得出预测趋势大致与接下来趋势分析中的季节性趋势一致。

对伊拉克地区近三年来发生的恐怖袭击事件进行了季节性分析和趋势分析。

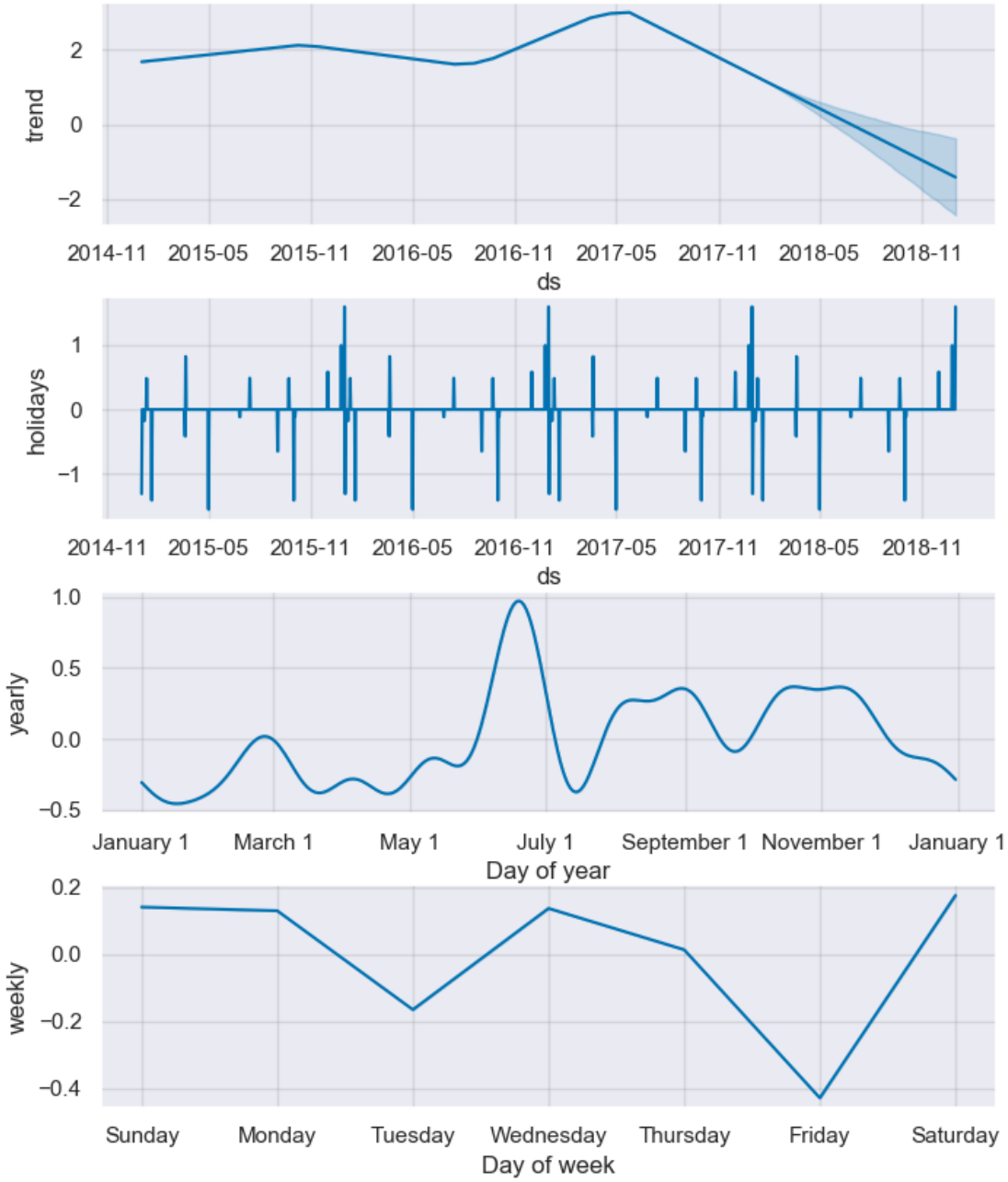

图 4-12 伊拉克恐怖袭击趋势图

由于 Prophet 使用加性模型，y 轴代表相对于趋势的绝对变化。从表中我们得知，6 月的飙升发生在斋月结束之前，12 月的跳跃发生在默罕默德诞辰之后。9 月份的下降发生在古尔邦节和伊斯兰新年附近。与此同时，周五发生的攻击次数较少，与穆斯林星期五祈祷相对应。总体遭到恐怖袭击趋势呈下降状态，分析得出的预测趋势与上述趋势分析中显示的季节性趋势一致。

- **巴基斯坦**

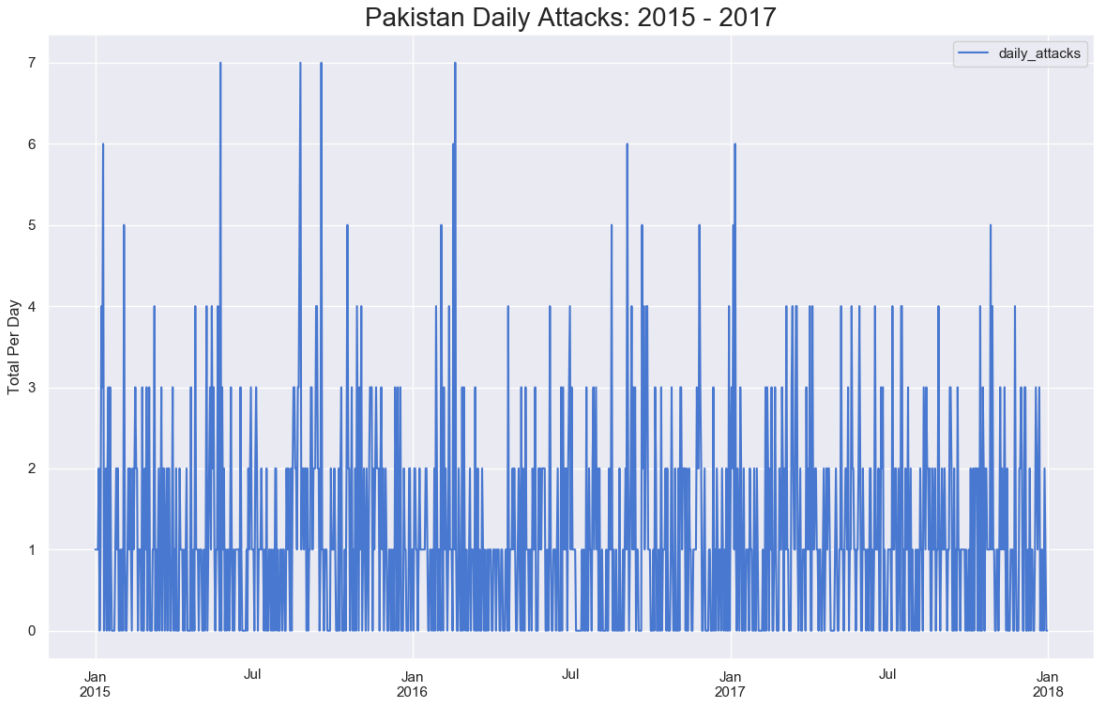

图 4-13 巴基斯坦每日所遭受的恐怖袭击数量

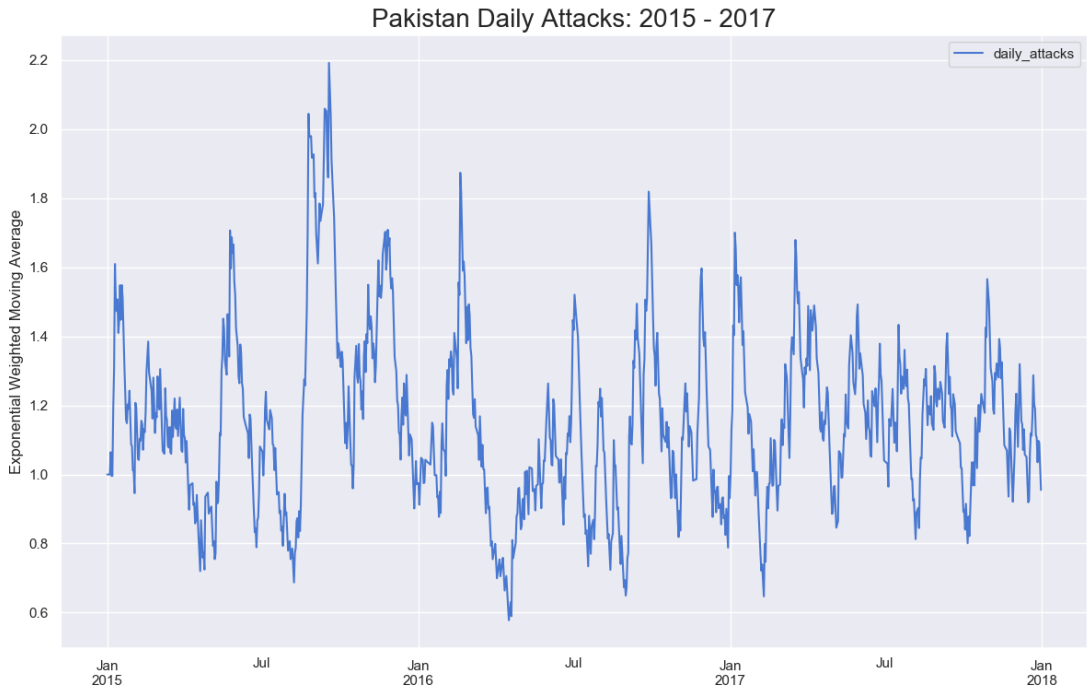

图 4-14 巴基斯坦每日所遭受的恐怖袭击数量——平滑后

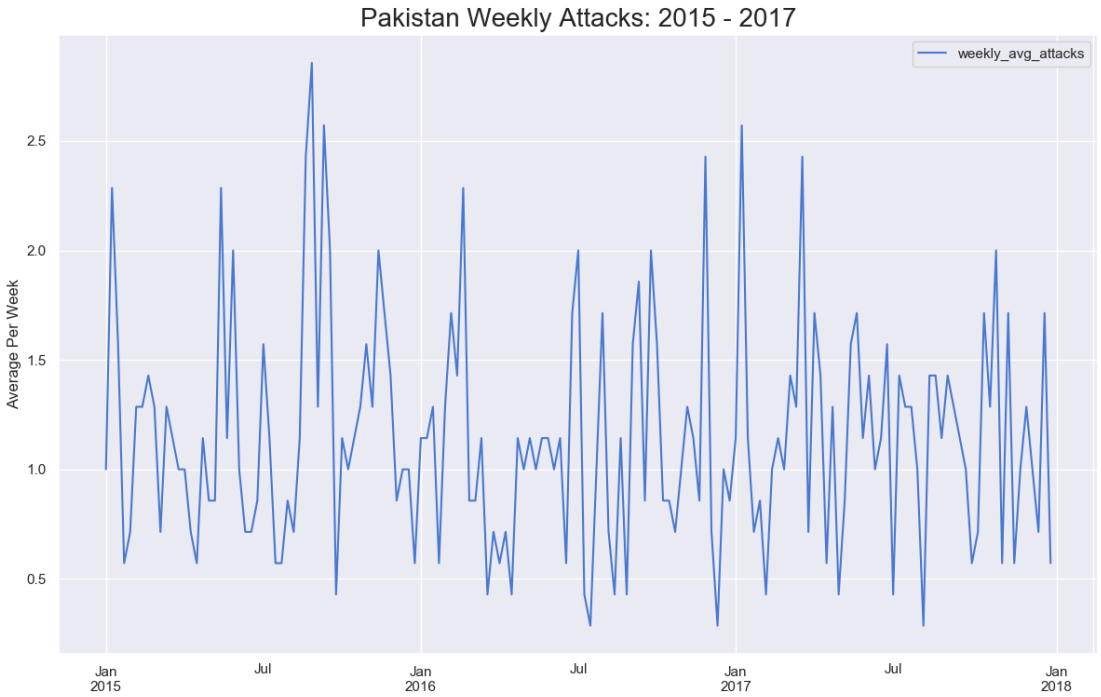

图 4-15 巴基斯坦每周所遭受的恐怖袭击数量

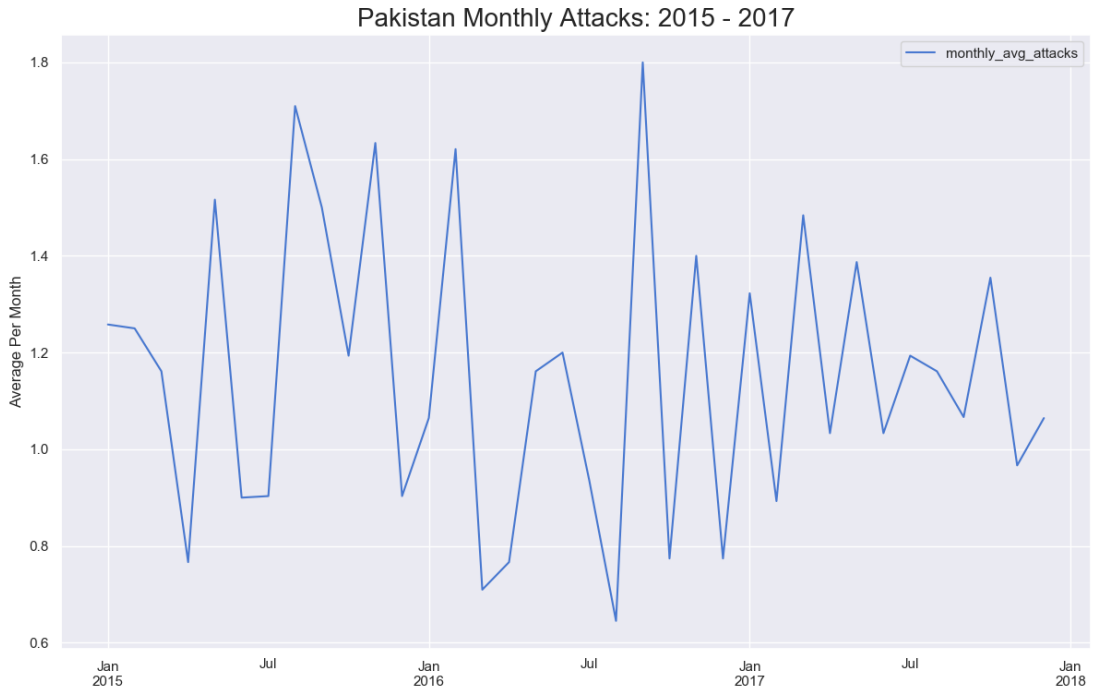

图 4-16 巴基斯坦每月所遭受的恐怖袭击数量

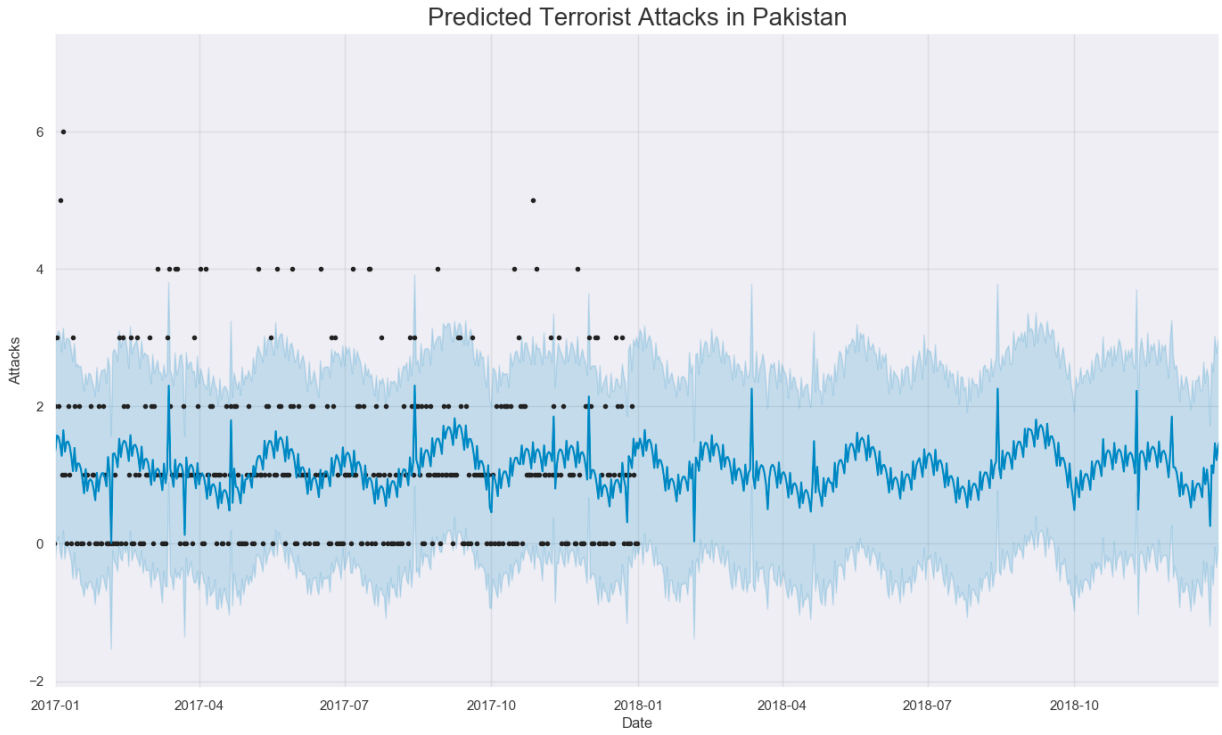

图 4-17 2018 年巴基斯坦地区恐怖袭击趋势

将 2018 年巴基斯坦发生恐怖袭击的次数的预测图进行放大，可以看到在 2018 年 6、9 月份会有明显上升，而在 3、8、10 月份会有明显的下降，模型得出预测趋势大致与接下来趋势分析中的季节性趋势一致。

对巴基斯坦近三年来发生的恐怖袭击事件进行了季节性分析和趋势分析。

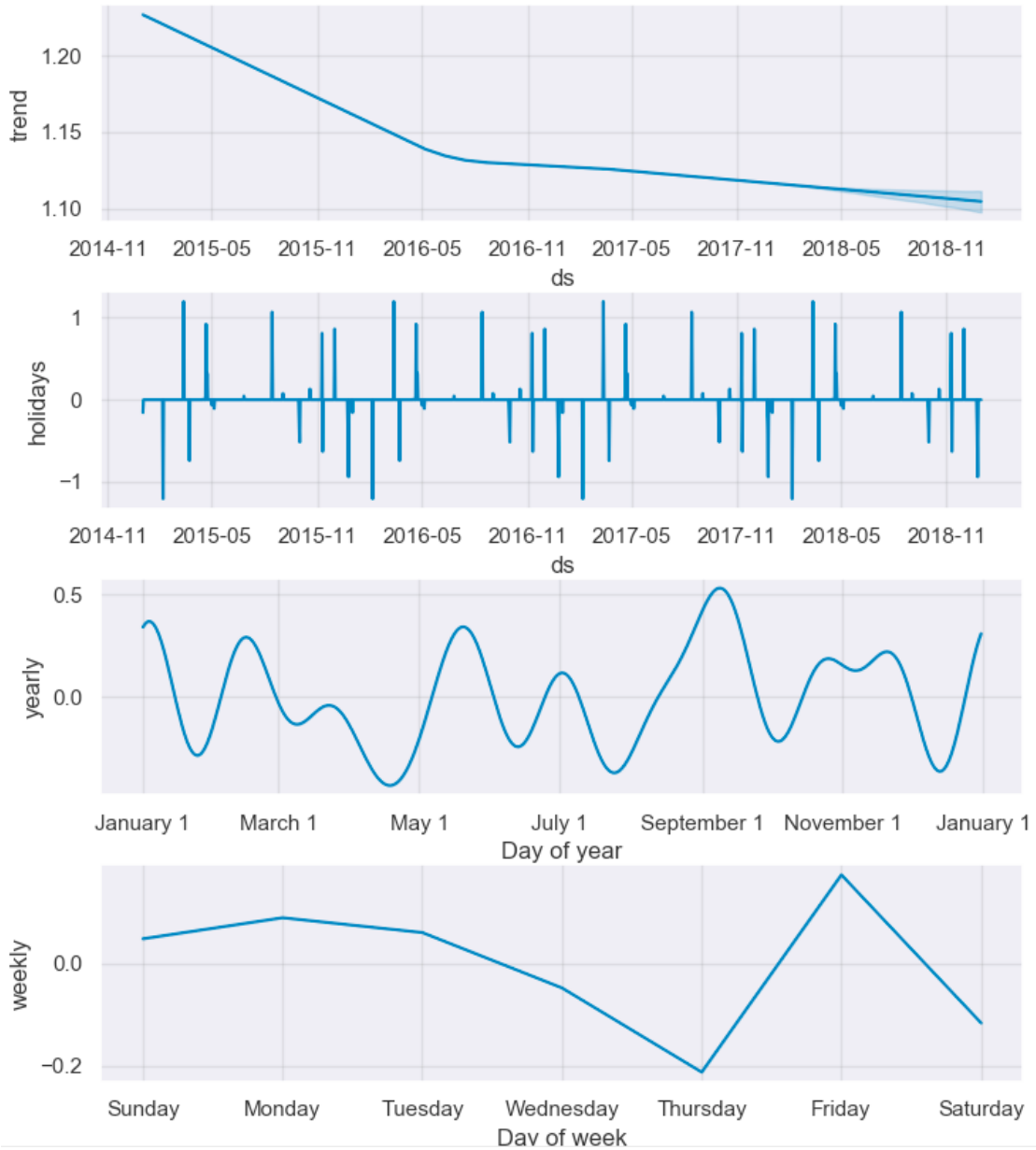

图 4-18 巴基斯坦恐怖袭击趋势图

从表中我们得知，巴基斯坦每年发生袭击事件的高峰期是在 6 月份的开斋节附近和 9 月的开斋节附近，10 月份排灯节和新年前后，3 月份巴基斯坦日、8 月份独立日和 10 月份阿舒拉节前后恐怖袭击事件会有大幅度下降，总体遭到恐怖袭击趋势呈下降状态，表中也可以得出在巴基斯坦，周五发生的攻击次数最多。

- **阿富汗**

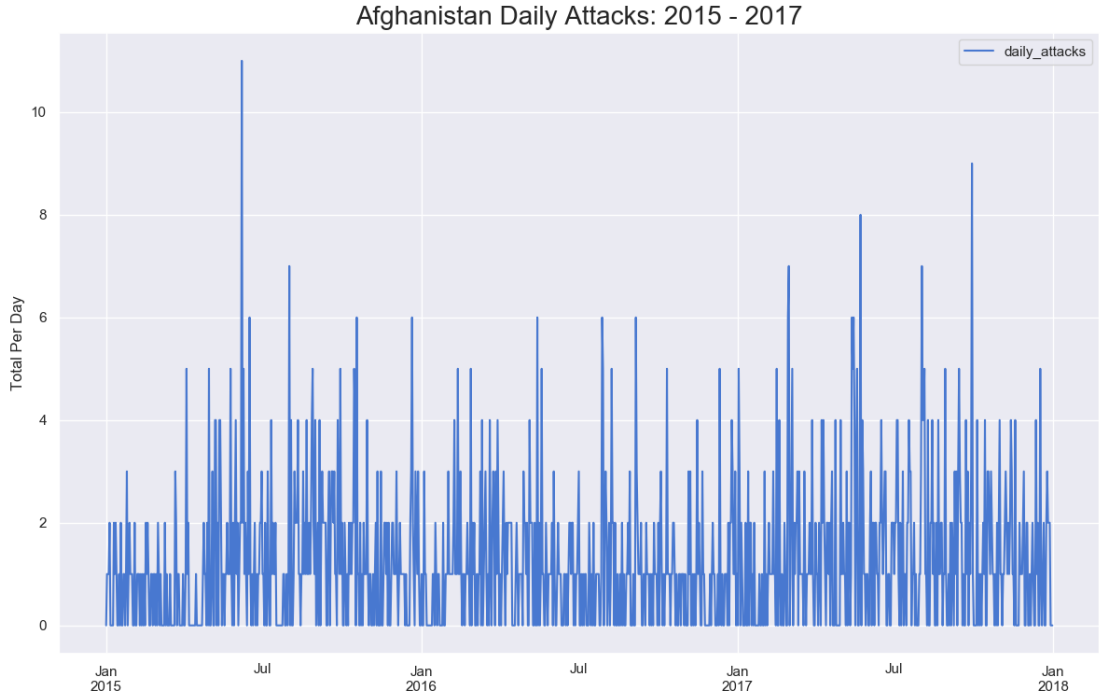

图 4-19 阿富汗每日所遭受的恐怖袭击数量

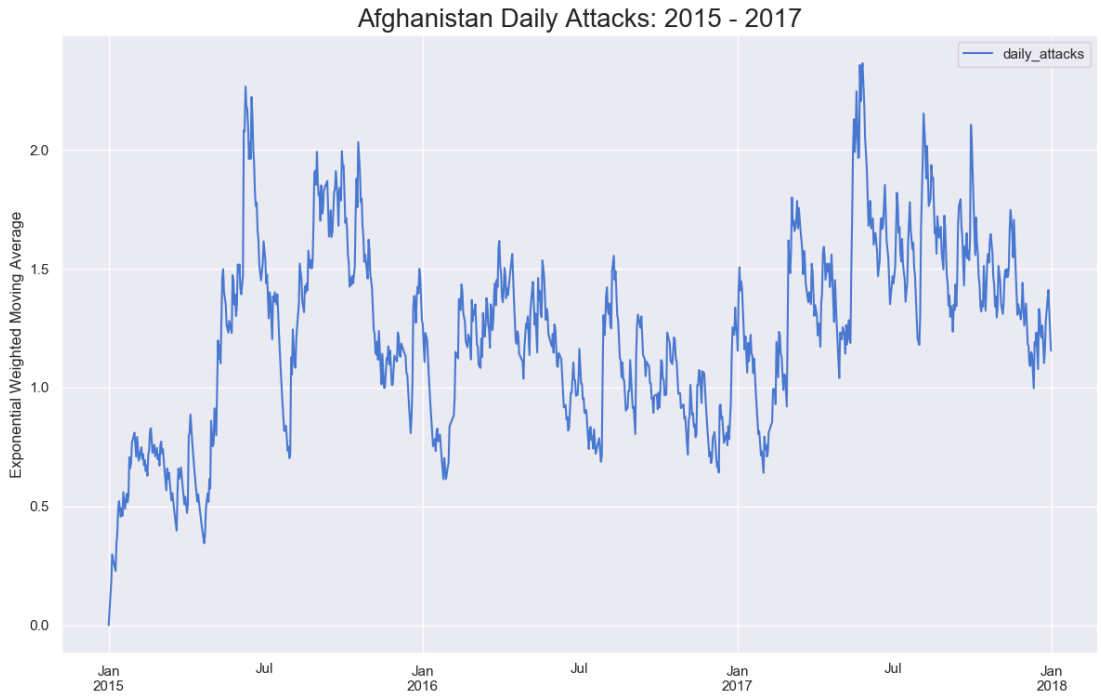

图 4-20 阿富汗每日所遭受的恐怖袭击数量——平滑后

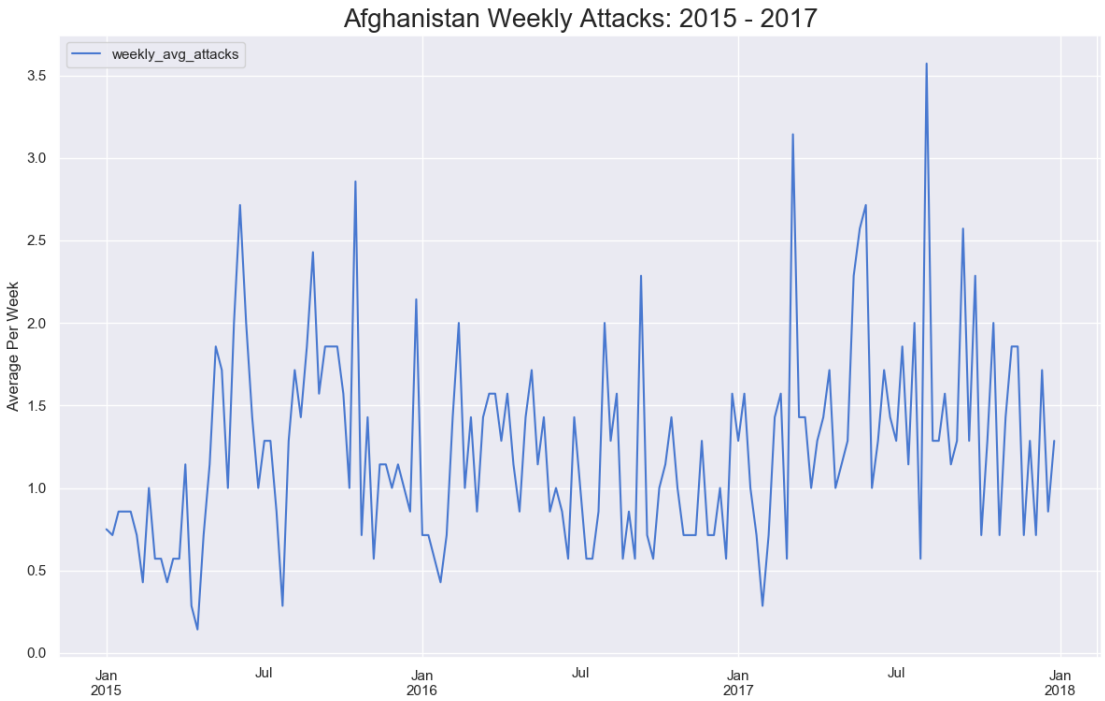

图 4-21 阿富汗每周所遭受的恐怖袭击数量

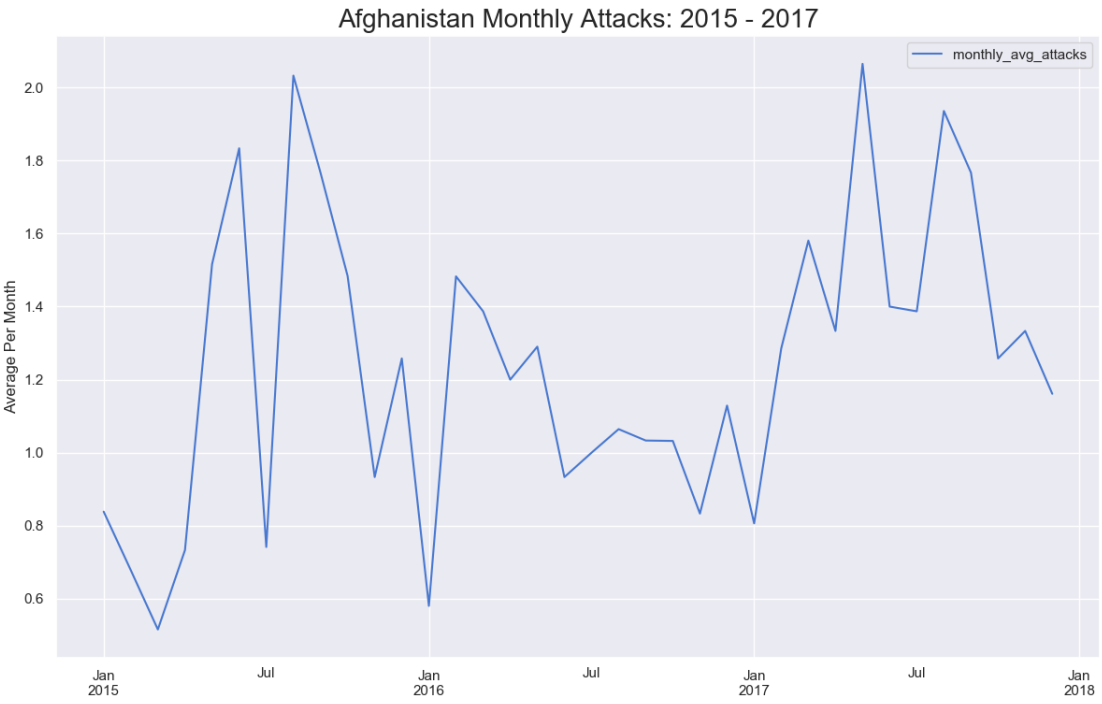

图 4-22 阿富汗每月所遭受的恐怖袭击数量

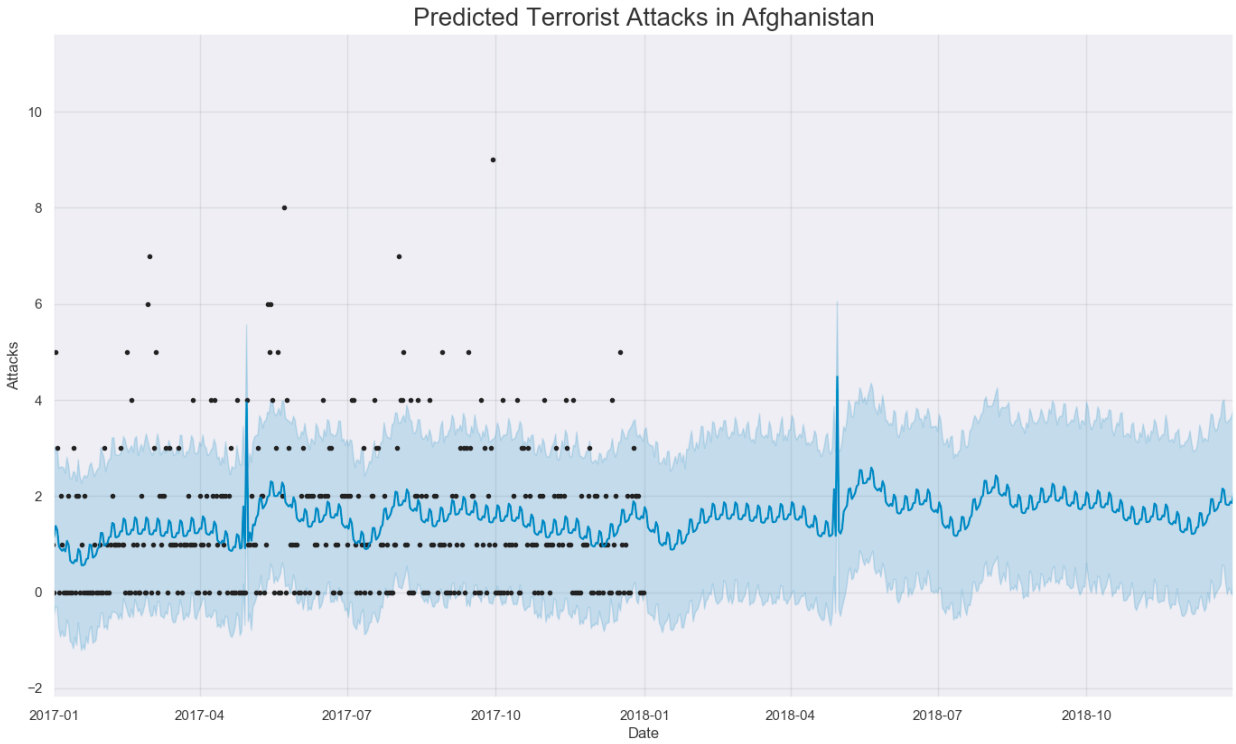

图 4-23 2018 年阿富汗地区恐怖袭击趋势

将 2018 年阿富汗发生恐怖袭击次数的预测图放大，可以看到预测图中新年前后发生的恐怖袭击事件次数较少，在 5、6 月份会有上升，在 7 月份会有下落并在 8 月回升持续至新年。

对阿富汗地区近三年来发生的恐怖袭击事件进行了季节性分析和趋势分析。

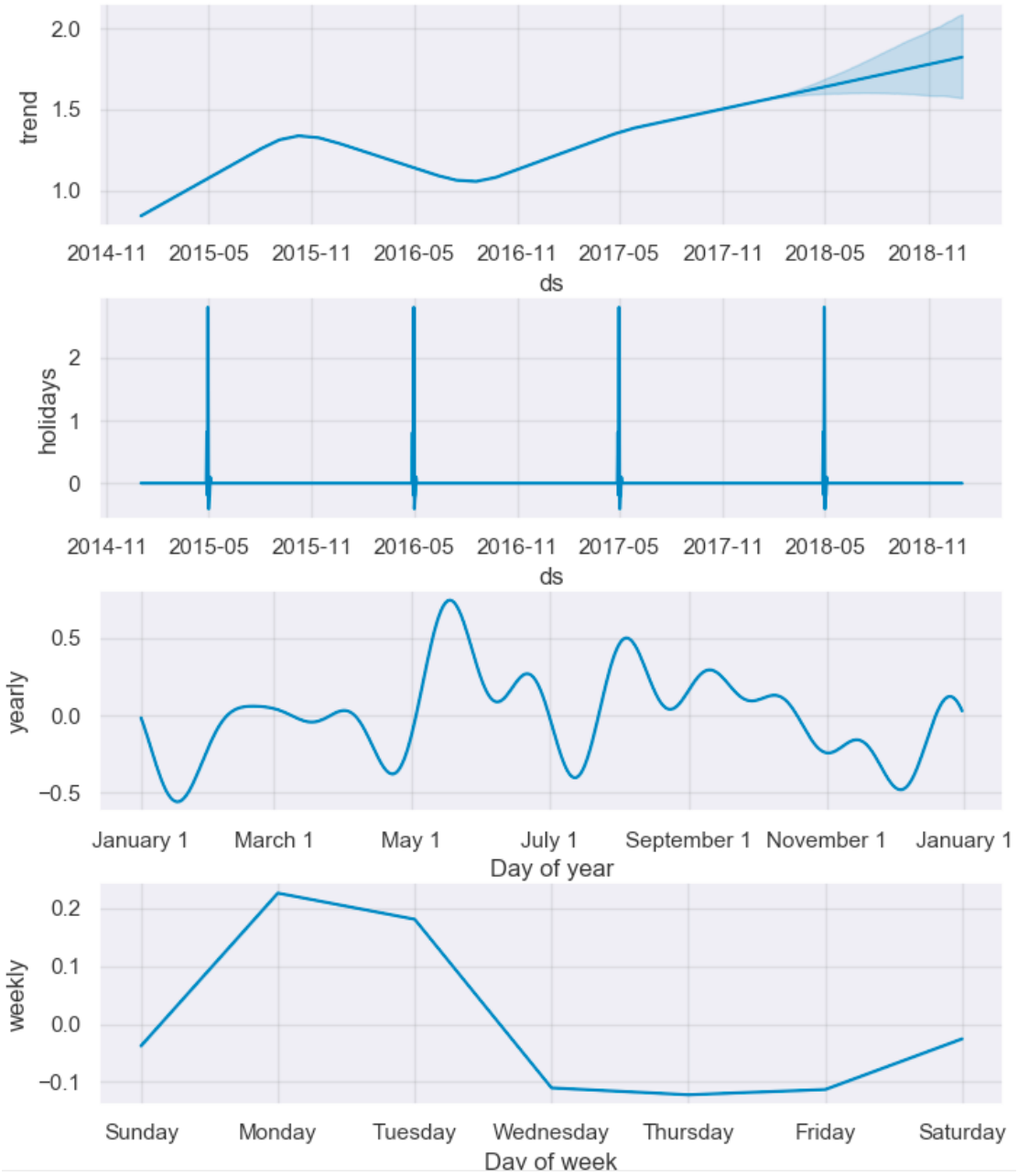

图 4-24 阿富汗恐怖袭击趋势图

从表中我们得知，阿富汗每年的恐怖袭击高峰期发生在 4 月阿富汗胜利日和 8 月独立日前后，在新年前后下降，周三至周日发生的恐怖袭击事件较少。总体遭到恐怖袭击趋势呈上升状态。

综上所述，恐怖袭击主要发生重点地区的节假日前后，例如开斋节、宰牲节等。从每周情况看，星期五是特殊的一天，在阿富汗和伊拉克这一天的恐怖袭击次数均为最少，而在巴基斯坦星期五最多，结合实际，巴基斯坦是穆斯林世界中伊斯兰教最盛行的国家之一，但是由于星期五是穆斯林集体礼拜日，穆斯林会在这天下午去清真寺集体祈祷。这也是星期五恐怖袭击次数较多的主要原因之一。在分析了阿富汗、伊拉克和巴基斯坦这些重点地区的恐怖袭击情况后，可以看出近年来恐怖袭击在伊拉克、巴基斯坦地区呈下降趋势，结合时事，这与美俄等大

国正在伊拉克和叙利亚进行围剿“伊斯兰国”极端组织战争有关，“伊斯兰国”极端组织逃亡邻国和阿富汗，导致阿富汗地区恐怖袭击呈上升趋势。通过数据分析，恐怖主义具有手段的多样化以及根源多样性等特点，恐怖袭击热点与燃点主要集中在中东地区、中亚的恐怖动荡弧，这片地区的恐怖组织互相勾结，共同作案，已成为威胁国家安全，社会稳定的重要因素，因此反恐应成为许多国家部门施政的重中之重。从伊拉克、巴基斯坦的分析结果并结合当今时事可以看出，美俄大国在这两个恐怖袭击事件高发国家加大打击力度，迫使恐怖分子转移基地，改变了伊拉克与巴基斯坦的反恐态势。因此军事打击是国际反恐重要且主要的手段。

但从国际形势上来看，反恐形势反而愈演愈烈，这与只注重军事打击有关。只考虑军事打击却忽略了恐怖主义产生的根源，并不能使人们正视恐怖头目非政治领袖的事实，反而会使更多的年轻人被煽动而加入恐怖组织。应当积极在国际外交中批驳恐怖主义的协力谬论，让反恐文化深入每个人心中，同时不断提高公民防恐、反恐和自我保护意识才能从根源上打击恐怖主义势力。

4.3.3 有关其他因素的建模与分析

通过数据分析工具，我们可以看到 1998 年至 2017 年全球恐怖袭击次数有所下降，致命性也越来越低。

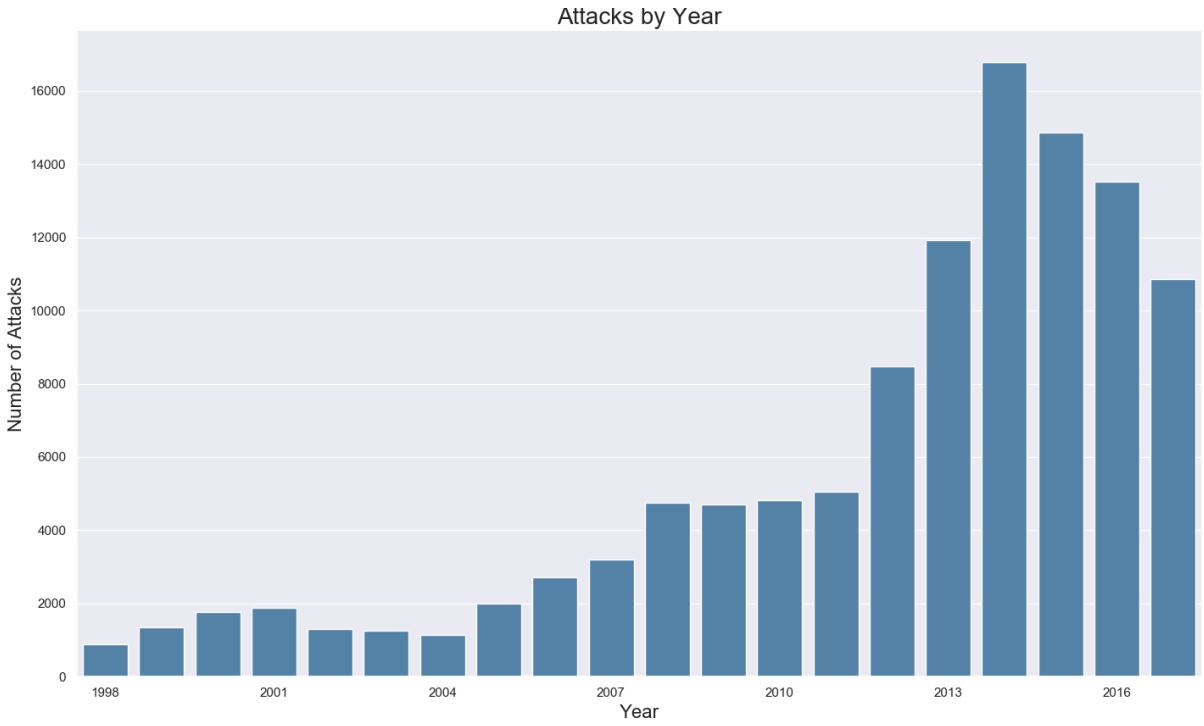

图 4-25 全球范围内恐怖袭击次数

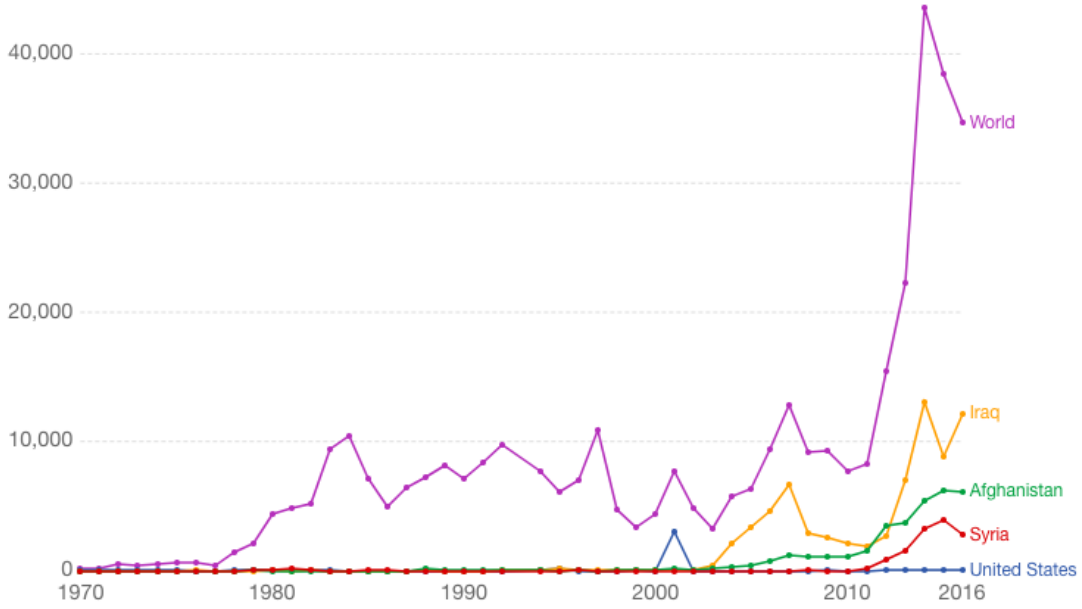

图 4-26 各地区的恐怖袭击死亡人数

全球各地的袭击事件从 2014 年约 17,000 起降至 2017 年约 11,000 起。致命受害者人数在同一时期减少了近一半。但如图所示，去年的袭击总数仍然远高于 20 世纪 90 年代（1998 年）。结合任务二，尽管伊斯兰极端主义仍然是 2017 年最活跃的恐怖主义组织，但它的攻击次数减少了 10%，与 2016 年相比，死亡人数减少了 40%。

在下面的图表中，我们看到了按地区汇总的恐怖袭击造成的死亡人数。在这段时间里，一些地区存在特定时期的相对较高的死亡人数。在过去十年中，中东和非洲的死亡人数大幅增加。2016 年，大约 75%的与恐怖主义相关的死亡事故发生在中东和非洲，在南亚和东南亚的比例不到四分之一，在欧洲约为 1%，在美洲则不到 0.5%。

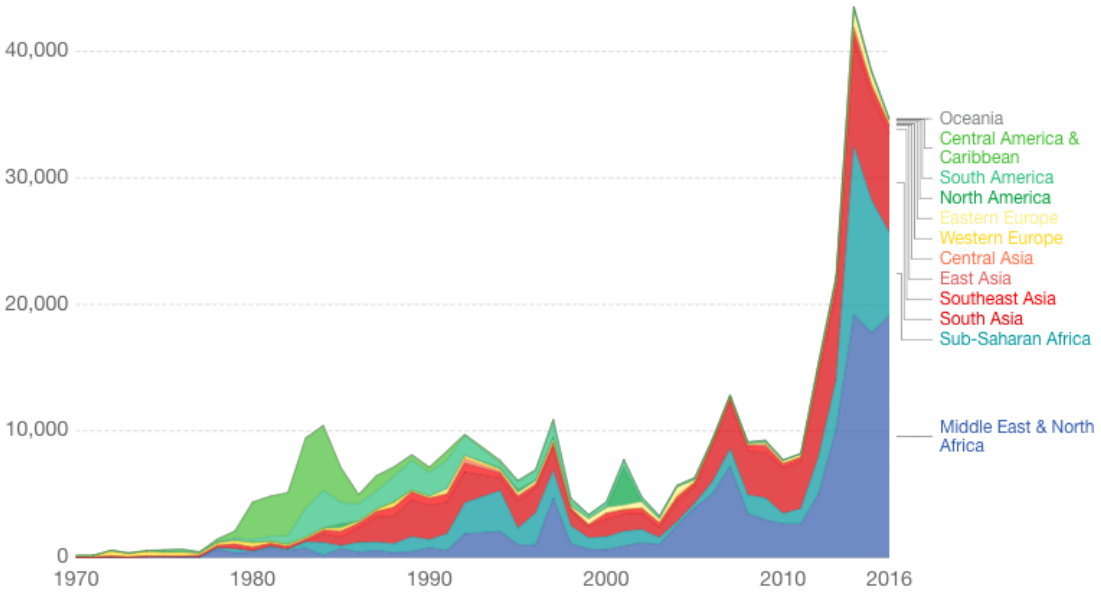

图 4-27 各地区的恐怖袭击死亡人数

经验分布函数是对样本中生成点的累积分布函数的估计，它以概率 1 收敛到该基础分布，用样本来推断总体的状态。从图中我们可以得出在一定范围内死亡人数占某确定人群中的比。

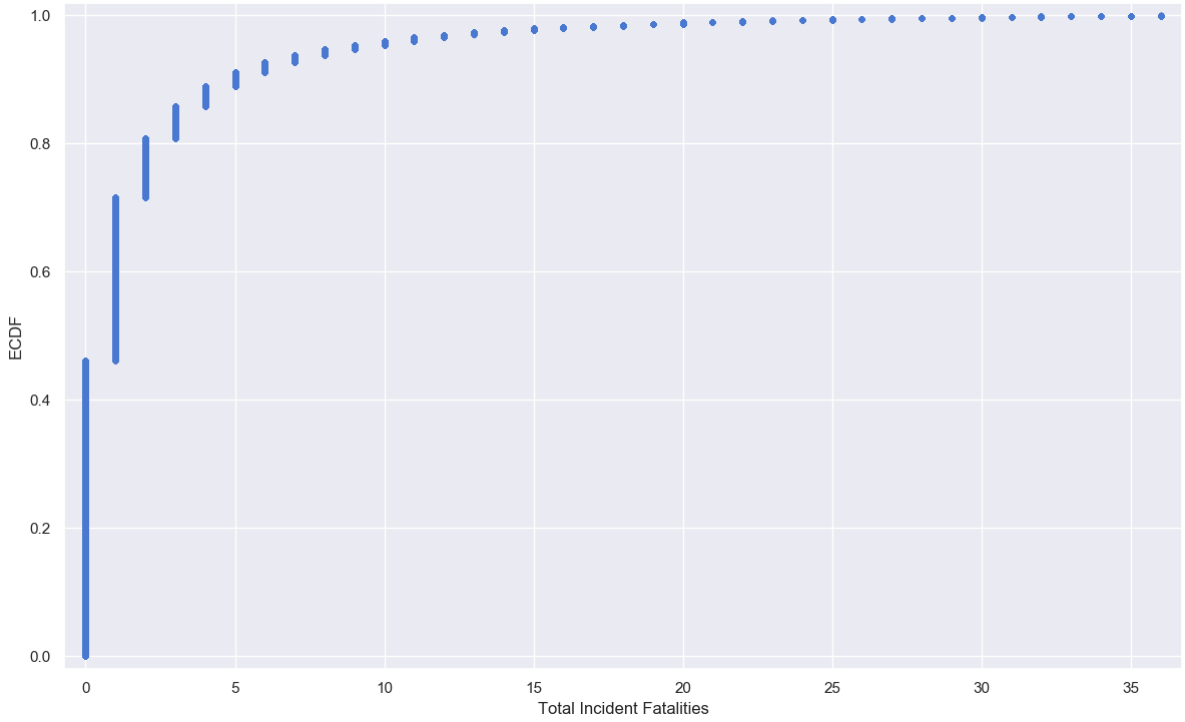

图 4-28 死亡经验积累分布图

结合图4-28和图4-29分析，2001年9月11日的发生的事件被称为911事件，标志着世界历史的转折点和“反恐战争”的开始。据估计，911造成3000人死亡，使其成为人类历史上最致命的恐怖事件。随后，2001年反恐战争入侵阿富汗，2003年入侵伊拉克。从地理区域分布图可以得知911事件之前的恐怖主义集中在拉丁美洲和亚洲，但在911事件之后转移到了中东地区，超过四分之一的恐怖主义袭击事件发生在伊拉克。由于激进的伊斯兰意识形态和宗派暴力，911事件后的恐怖主义集中在主要是穆斯林国家。

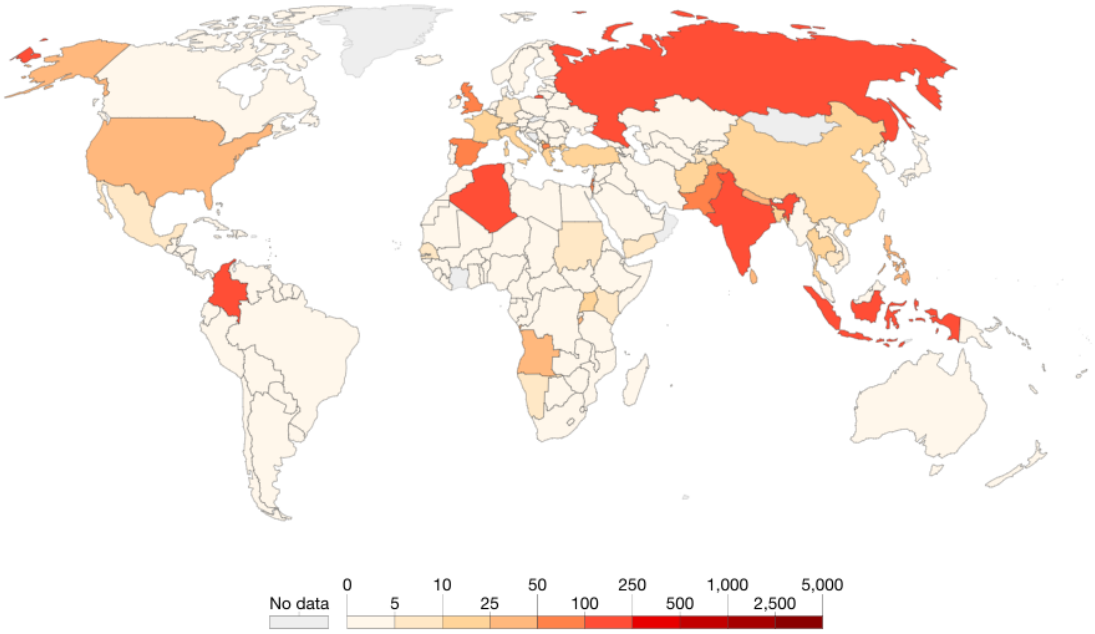

图 4-29 2001 年恐怖袭击事件

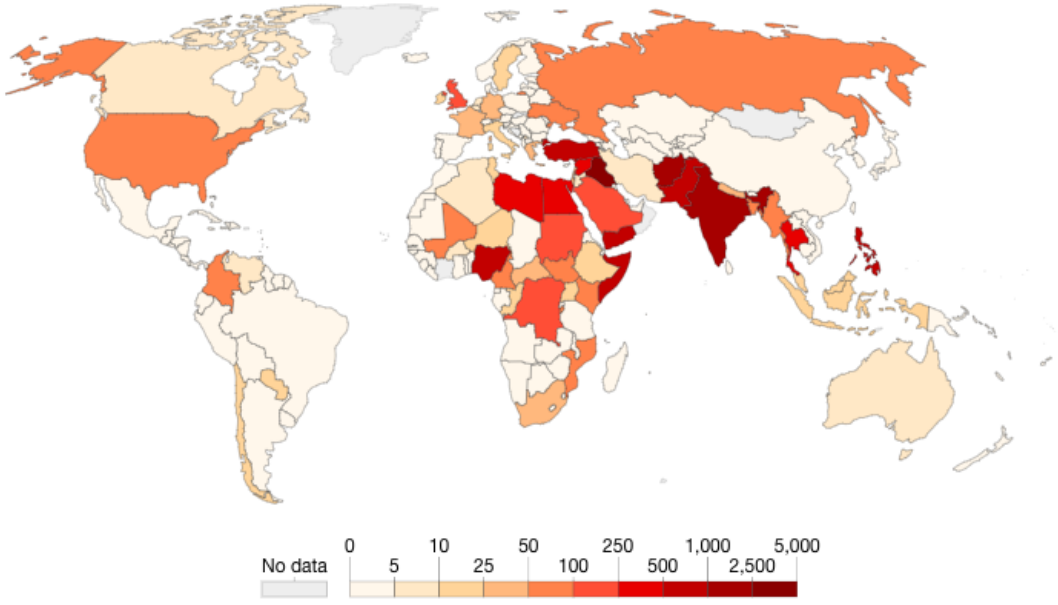

图 4-30 2016 年恐怖袭击事件

使用机器学习模型成功预测恐怖袭击的地区主要依赖于大量的高维数据，包括地理，自然和社会因素，但要将这些因素的数据集预处理量化为数值属性。下表结果表明，恐怖袭击的高风险区域主要集中在南亚，中东，中非，北非，南美洲西北部，南美洲南部，南欧和西欧。

表 4.9 地理区域上分布的恐怖袭击

Middle East & North Africa	41240
South Asia	37419

Sub-Saharan Africa	13069
Southeast Asia	9638
Eastern Europe	4242
Western Europe	3568
South America	2668
North America	737
Central Asia	273
East Asia	211
Central America & Caribbean	97
Australasia & Oceania	85

通过表格分析得知，例如印度和巴基斯坦、以色列和巴勒斯坦等国家由于边界领土等原因发生冲突性袭击。

表 4.10 各地区的意识形态攻击

INT_IDEO_txt / INT_IDEO_txt	NO	UKNOWN	YES	All
Sub-Saharan Africa	5717	4551	2801	13069
Eastern Europe	508	2765	969	4242
Western Europe	782	2200	586	3568
Middle East & North Africa	7305	26213	7722	41240
South America	1953	599	116	2668
South Asia	14179	20777	2463	37419
Southeast Asia	3675	5671	292	9638
North America	65	524	148	737
East Asia	47	154	10	211
Central Asia	28	232	13	273
Australasia & Oceania	4	71	10	85
Central America & Caribbean	11	84	2	97
All	34274	63841	15132	113247

通过表格分析得知，大部分的恐怖袭击事件以意识形态领域的冲突为主，比如伊斯兰国家与西方国家的冲突，如基地组织对世界各国的袭击，他们在政治、经济、文化、社会等领域具有不同的价值观。

结合互联网上的资料显示，尽管恐怖袭击在全球范围内有所改善，但一些特定国家的恐怖主义比以前更多。在美国，致命袭击的数量几乎增加了两倍。在 21 世纪，互联网已经成为恐怖分子交流的核心工具，它使恐怖主义组织能够毫无阻碍地、成功地传达他们的信息，并以世界为目标招募新成员。这些发展将会造成严重的后果，因为监管部分对其仍然知之甚少，互联网已经大大提高了恐怖分子进行交流的能力。

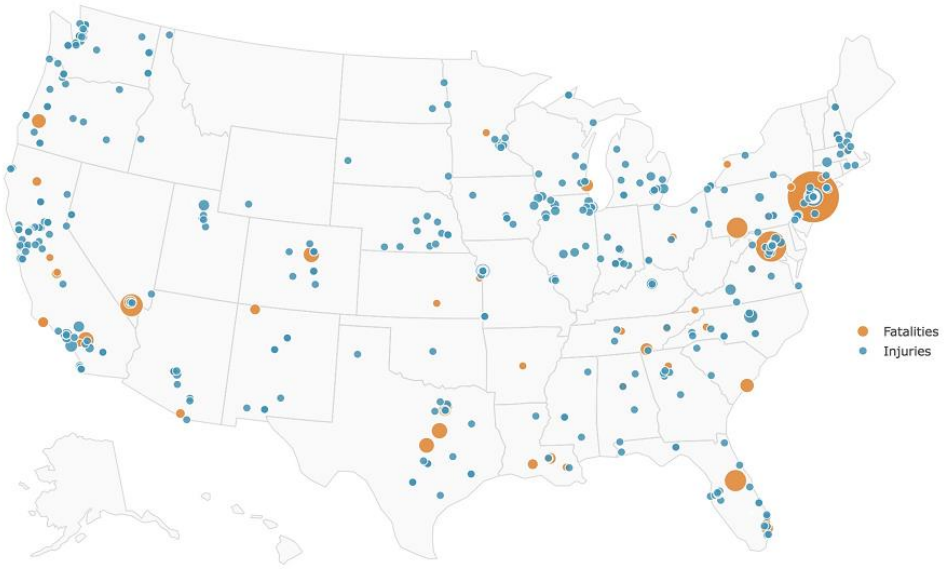

图 4-31 美国地区恐怖袭击数量

4.4 任务四模型的建立及求解

任务四要求利用附件 1 数据进一步使用。首先我们对数据集进行了可视化操作，并且针对该问题做了以下两点考虑使数据集充分利用。

通过对数据分析显示，如图所示，近 20 年来遭受恐怖袭击的国家每年所遭受袭击数量，其中 2014 年恐怖袭击次数达到了峰值。

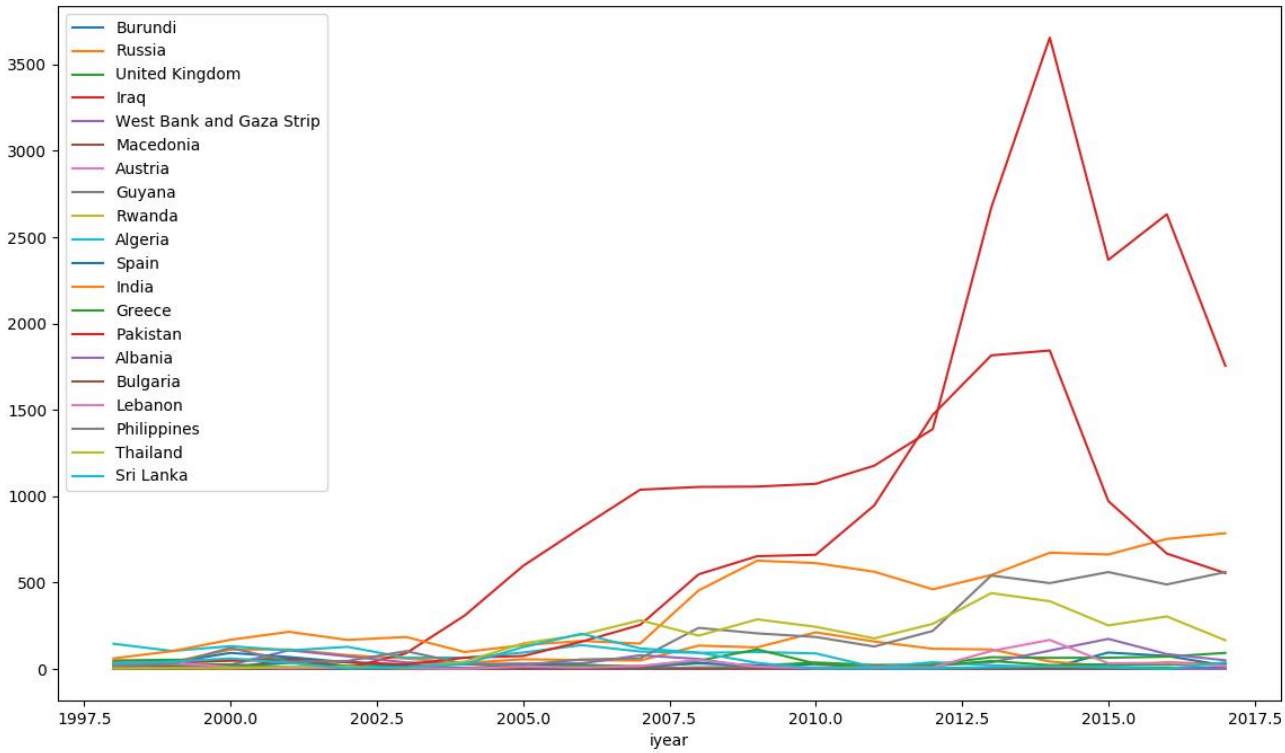

图 4-32 近 20 年遭受恐怖袭击的国家

（1） 利用数据集中所显示的恐怖袭击中使用的武器，按照其类型进行分类，此问题提出的依据如图所示。

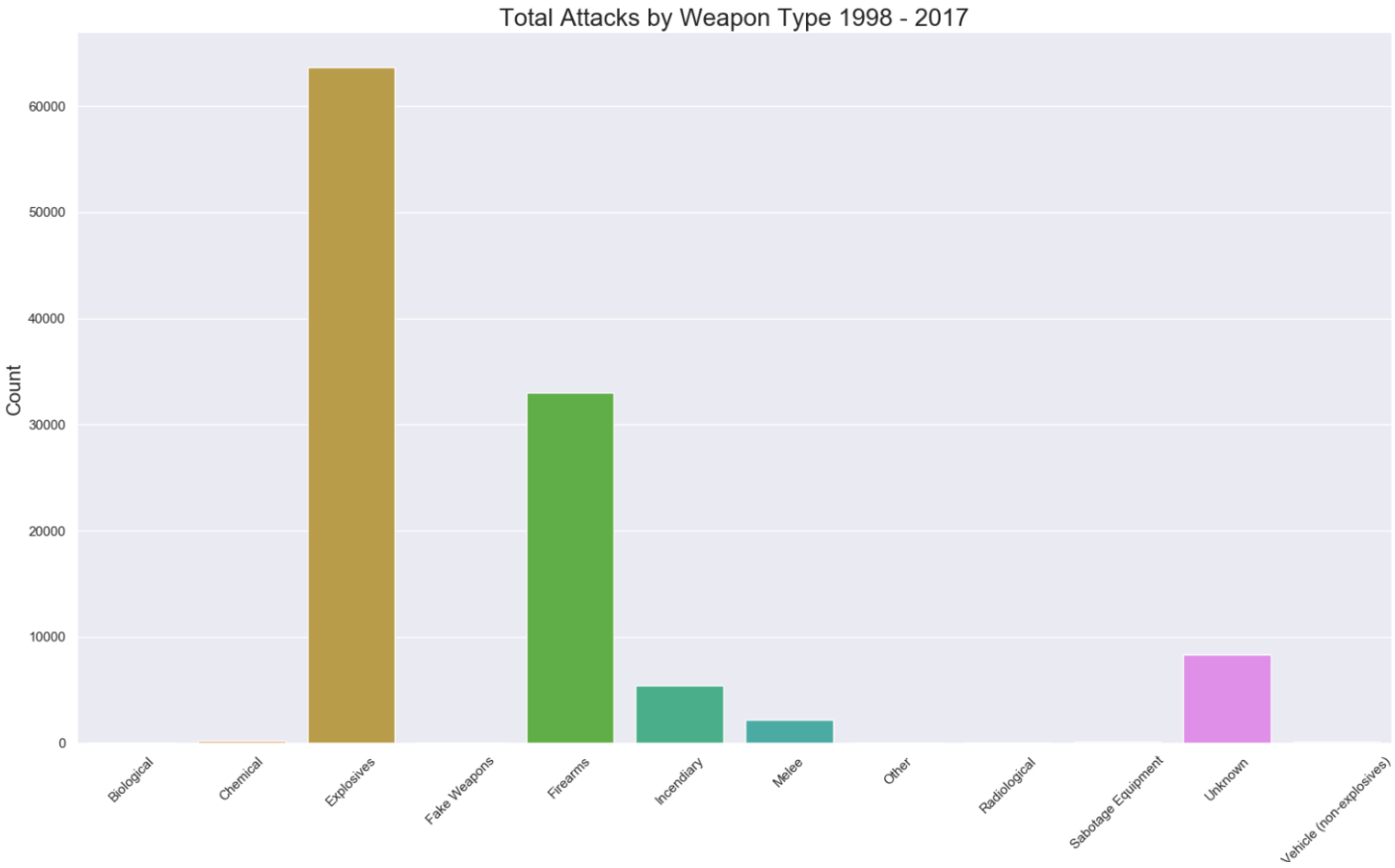

图 4-33 武器类型统计图

图片显示爆炸袭击，枪炮，以及不知具体武器的攻击类型占了前三位。所以根据显示结果，利用附件 1 的数据建立了武器分类模型，以便预测出来武器攻击类型以做好未来针对该武器类型的防范。

（2） 自然语言处理是人工智能最早的研究领域之一，通过对附件 1 数据分析得知有 summary 和 country_txt 两个特征。利用 Seq2Seq 模型对这两个属性进行数学建模。我们所要解决的问题是根据 summary 内容预测出 country_txt 中的内容。在数据分析领域，数据集严重影响着分析结果，每一条数据都很珍贵，如果 country_txt 内容丢失，我们完全可以从 summary 中预测出结果，而不必删除缺失值的数据。

4.4.1 支持向量机分类模型的建立

（1）支持向量机

在武器分类问题中，选定的特征为：'country_txt'，'region_txt'，'attacktype1_txt'，'nkill'，'nwound'。预测标签为'weapontype1'。给定训练样本集$\boldsymbol{D} = \{(x_i, y_i), i = 1,2,\cdots,N\}, x \in R^d, y \in \{\pm 1\}$，$x_i$为第 i 个特征向量，$y_i$为$x_i$的类标记。

选取径向基核函数$K(x,z)$和惩罚参数C，其中核函数定义[3]为：设χ是输入空间（欧氏空间R^n的子集或离散集合），又设$\mathcal{H}$为特征空间（希尔伯特空间），

如果存在一个从χ到$\mathcal{H}$的映射

$$\phi(x): \chi \to \mathcal{H} \tag{4.4.1}$$

使得对所有$x, z \in \chi$，函数$K(x,z)$满足条件

$$K(x,z) = \phi(x) \cdot \phi(z) \tag{4.4.2}$$

则称$K(x,z)$为核函数，$\phi(x)$为映射函数，式中$\phi(x) \cdot \phi(z)$为$\phi(x)$和$\phi(z)$的内积。

构造并求解最优化问题

$$\min(\alpha) \ \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} a_i a_j y_i y_j K(x_i \cdot x_j) - \sum_{j=1}^{N} \alpha_i \tag{4.4.3}$$

$$s.t. \sum_{i=1}^{N} \alpha_i y_j = 0 \quad 0 \le \alpha_i \le C, i = 1,2,\cdots,N \tag{4.4.4}$$

解得最优解$\alpha^* = (\alpha_1^*, \alpha_2^*, \cdots, \alpha_N^*)^T$。

选择α^*的一个正分量，计算$0 < \alpha_j^* < C$，计算

$$b^* = y_j - \sum_{i=1}^{N} \alpha_i^* y_i K(x_i, x_j) \tag{4.4.5}$$

构造决策函数：

$$f(x) = sign\left(\sum_{i=1}^{N} \alpha_i^* y_i K(\mathrm{x} \cdot x_i) + b^*\right) \tag{4.4.6}$$

基于以上算法过程，对具有上述 5 个属性的样本x进行武器预测，并挑选 20%的验证集 y_test 与预测结果 y_pred 进行模型评估。

4.4.2 模型求解与结果分析

针对上述模型的算法步骤及实现原理，通过 Python 编程实现上述模型的算法建立。利用 SVM 分类器对武器类型进行分类，实验准确率为 0.8895805739514349。其算法产生的混淆矩阵如下图所示。

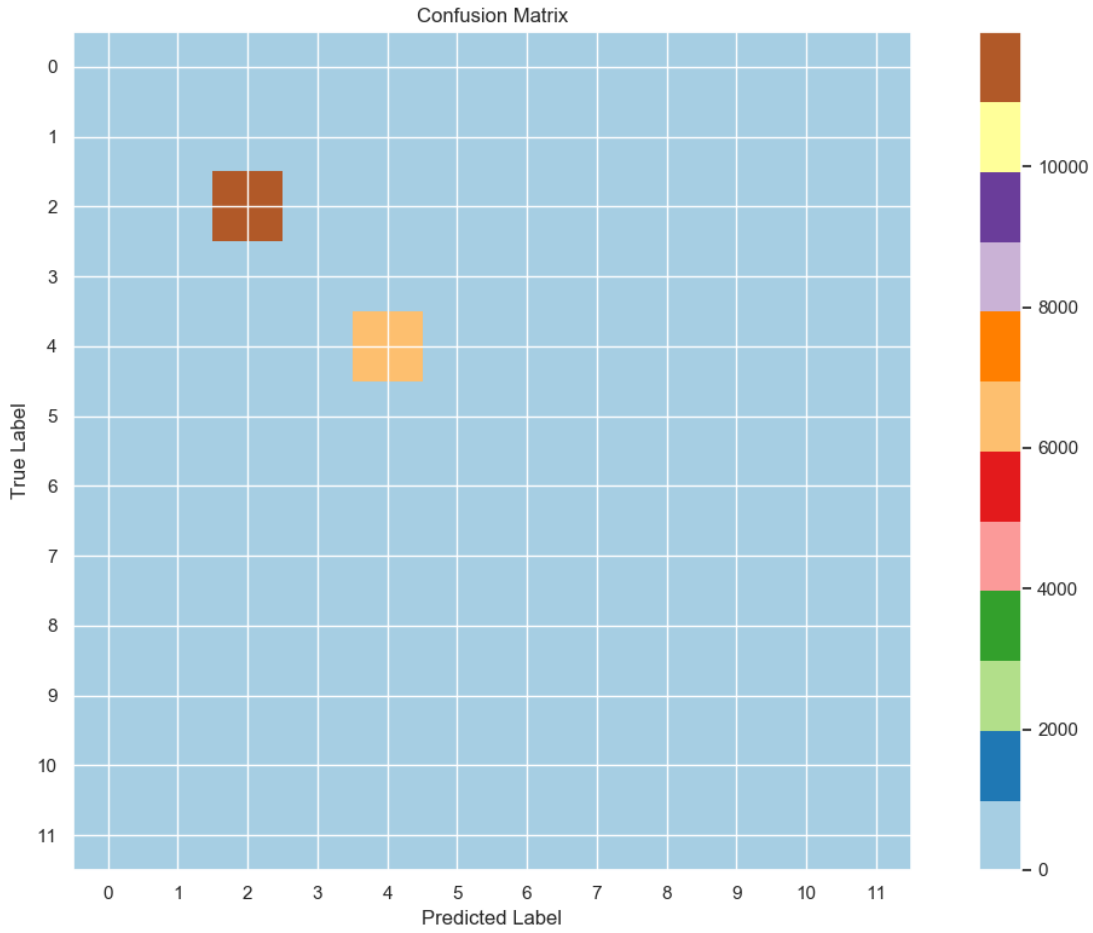

图 4-34 SVM 分类器产生的混淆矩阵

混淆矩阵在图像精度评价中，主要用于比较分类结果和实际测得值，可以把分类结果的精度显示在一个混淆矩阵里面。混淆矩阵图说明我们所构建的 SVM 分类器能够准确的对武器类别进行有监督的预测。

4.4.3 Seq2Seq 模型的建立

（1） LSTM 网络

LSTM（Long Short-Term Memory）结构[4]是一种特殊的循环体结构，可以学习长期依赖信息，其结构如图所示。

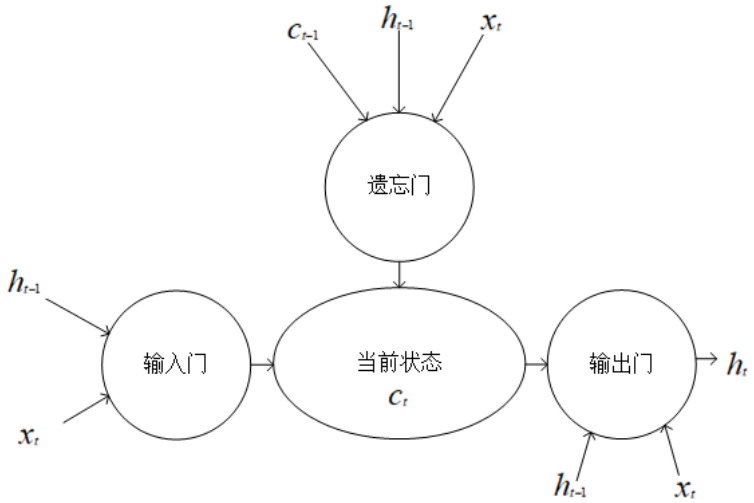

图 4-35 LSTM 结构示意图

LSTM 有通过精心设计的称作为“门”的结构来去除或者增加信息到细胞状态的能力。门是一种让信息选择式通过的方法。具体 LSTM 每个“门”的公式定义如下。

$$z = tanh(W_z[h_{t-1}, x_t])$$（输入值）

$$i = sigmoid(W_i[h_{t-1}, x_t])$$（输入门）

$$f = sigmoid(W_f[h_{t-1}, x_t])$$（遗忘门）

$$o = sigmoid(W_o[h_{t-1}, x_t])$$（输出门）

$$c_t = f \cdot c_{t-1} + i \cdot z$$（新状态）

$$h_t = o \cdot tanhc_t$$（输出）

让文本内容通过 LSTM 网络学习后，获得每个神经元输出的隐层向量，利用该隐层向量对文本内容进行预测分析。

（2）Seq2Seq 模型

Seq2Seq 模型的基本思想非常简单，通过使用一个循环神经网络读取输入句子，将整个句子的信息压缩到一个固定维度的编码中；再使用另一个循环神经网络读取这个编码，将其解压为目标语言的一个句子。这两个循环神经网络分别称为编码器（Encoder）和解码器(Decoder)，这个模型也称为 Encoder-Decoder 模型。

编码器部分的结构与语言模型几乎完全相同：输入为单词的词向量，输出为 softmax 层产生的单词概率。

解码器与编码器一样拥有词向量层和循环神经网络，在训练过程中，编码器顺序读入每个单词的词向量，然后将最终的隐藏状态复制到解码器作为初始状态。解码器的第一个输入是特殊的<Go>字符，每一步预测的单词是训练数据的目标句子，预测序列的最后一个单词是与语言模型相同的<EOS>(End-Of-Sentence)

字符。在解码过程中，每一步预测的单词中概率最大的单词被选为这一步的输出，并复制到下一步的输入中。

针对我们要解决的问题，结合 Seq2Seq 模型给出下列示意图。

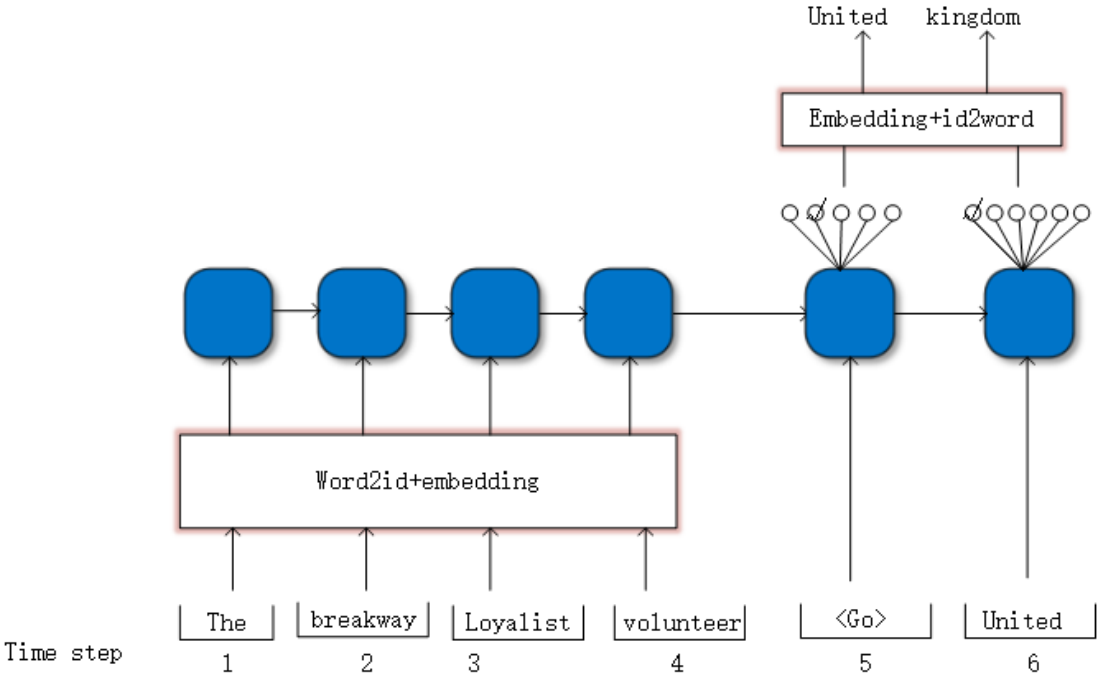

图 4-36 Seq2Seq 模型示意图

上述示意图简要的描述了 Seq2Seq 模型生成 country_txt 的步骤。首先在 Encoder 端，将文本处理为数字并且进行 Word Embedding（当前单词对应的单词向量）之后输入到长短期记忆网络（LSTM）层，Encoder 端通过学习输入数据，将其编码成一个固定大小的状态 S，继而将 S 传给 Decoder 端，Decoder 端再通过对状态向量 S 的学习来进行输出。

表 4.10 Seq2Seq 模型预测 country_txt 文本内容的算法步骤

算法 6：Seq2Seq 模型进行国家预测
输入部分： Inputs：输入数据（summary 的文本内容）的张量，长度不同的序列用<PAD>进行填充。 Targets：作为 Decoder 端训练部分输入（country_txt 的文本内容）的张量，每个序列的开始增加<Go>标记，结尾增加<EOS>标记，长度不同的填充<PAD>。 **输出部分：** Outputs：输出数据（country_txt 的文本内容）的张量，作为预测的结果。

4.4.4 模型求解与结果分析

通过深度学习中的 Tensorflow 框架编程实现上述模型的算法建立。实验准确率为 75%，这个数据结果显示，通过对文本输入基本可以实现对恐怖袭击发生地点的预测。实验只是采用了最基础的 Seq2Seq 模型，从而实验正确率还可以利

用已经公布的Glove数据集[4]进行Word Embedding从而提升结果;或者Encoder端使用双层LSTM网络进行学习;或者采用Attention机制[5],众多实验结果显示Attention机制对于提高Seq2Seq模型效果有着很显著的作用。

五. 模型分析及展望

本文研究的是对全球恐怖主义数据库（GTD）进行数据挖掘建模的过程。根据现有的样本集，采用K-Means聚类算法和AFW-Kmeans算法聚类算法、随机森林算法和k近邻分类算法、时序模型以及支持向量机分类算法和Seq2seq模型等多种方式建立数学模型，通过Python语言求解问题，最终确定了恐怖袭击事件分级、犯罪嫌疑人预测、研判未来反恐态势和利用数据集进行自然语言处理。

本模型的优点有：

1. 模型的扩展性和可移植性较好，当数据集中的字段发生了更改时，此模型仍然可以挑选出有效的特征进行数学建模。
2. 在对恐怖袭击事件分级的过程中，同时建立多种模型进行独立测试，各个模型的结果以及互联网中对真实事件的调研结果互为对照，互相验证，综合了多方面考虑，提高了分析结果的准确程度。
3. 在解决问题之前，采用数据预处理中的诸多方法，删除了不必要的数据和填补了缺失的数据等，使通过处理后的数据集产生的结果更有统计意义，与此同时，也减小了计算量。

本文在分析数据集对建立模型求解的有效特征方面做了深入的探索,但是现有的工作仍有未完善的地方，今后进一步分析和改进的内容如下，本文在任务一中对表1中事件的分级采用的是判断相似度的方式,衡量方式是借鉴其他人的思想，未来可以提出自己的，针对该问题采用多种属性以及新的加权方式对事件进行分级。但任务二中，由于多层感知机并未达到理想的实验结果，对模型还可以进一步调整。在任务三中，对下一年的反恐态势预测，仅考虑了日期与节假日。还有很多因素，比如近期的全球局势等因素并未参与考量。以及受限于计算机的计算性能，例如谱聚类等许多方法不能实现，需要考虑更加准确高效的模型来解决问题，因为时间有限，对于未完善之处将在日后的学习中继续进行。

六. 参考文献

[1]李四海，满自斌，自适应特征权重的K-means聚类算法[J]，计算机技术与发展，23（06）：101-105，2013。

[2] Sean J. Taylor, Benjamin Letham, Forecasting at Scale[J], 2017.

[3]李传科，许苗村，核函数的概念、性质及构造[J]，电脑知识与技术,11（32）：171-173，2015。

[4] Jeffrey Pennington, Richard Socher, Christopher D. Manning, GloVe: Global Vectors for Word Representation, https://nlp.stanford.edu/projects/glove/, 2014-09-18.

[5] Bahdanau D, Cho K, Bengio Y, Neural Machine Translation by Jointly Learning to Align and Translate[J], In Proc. of the International

Conference on Learning Representations (ICLR), 2014.

附录

备注：此处仅列出 K-Means 算法、随机森林算法、k-近邻算法、时间序列模型算法、支持向量机算法、Seq2Seq 模型代码，其他程序详见附件。

附录一：K-Means 算法

```
import pandas as pd
from sklearn.cluster import KMeans
import numpy as np
from openpyxl import Workbook

inputfile = './tmp/standardized.xlsx'
# centersfile = '../results/centersfile.xlsx'
labelsfile = './results/labelsfile.xlsx'

k = 5
# 读取数据进行聚类分析
data = pd.read_excel(inputfile)
# 检查数据中是否有缺失值
print(np.isnan(data).any())
# 删除有缺失值的行
data.dropna(inplace=True)
# 调用 kmeans 进行聚类分析
kmodel = KMeans(n_clusters=k)
kmodel.fit(data) # 训练模型

print(kmodel.cluster_centers_) # 查看聚类中心
#print(kmodel.cluster_centers_.shape) # (5, 10)

labels = np.reshape(kmodel.labels_,(-1,1))
# print(kmodel.labels_.shape) # (114182,)
print(labels)
```

附录二：随机森林算法

```
# 创建机器学习模型——随机森林
# Random Forest Model
# Create the model using 1000 estimators.
start = time.time()

# Create the model
rf1 = RandomForestClassifier(n_estimators = 1000, oob_score = True, n_jobs = -1,
random_state = seed)
```

```
# Fit it to the training data
rf1.fit(X_train, y_train)

end = time.time()
print("Execution Seconds: {}".format((end - start)))
print("\n")
print(rf1)
##############################################################################
##########
# 随机森林里重要的特征，前 25 个重要的特征
# 不用随机森林，则删除这里
# Feature Importance
# Display the top 25 features by importance in descending order.
# Get the modified column names with one hot encoding
column_names = list(X_train.columns.values)

# Create a descending sorted list of variables by feature importance
var_imp = sorted(zip(map(lambda x: x, rf1.feature_importances_), column_names),
             reverse = True)
print("\nFeatures Ranking - Top 25:\n")
for f in var_imp[0:25]:
    print(f)
# 利用训练好的随机森林模型验证，验证集，有组织的
# Predict labels on the test dataset
pred_labels1 = rf1.predict(X_test)

# Calculate the accuracy of the model
acc_score1 = accuracy_score(y_test, pred_labels1)
print("\nAccuracy: {}".format(acc_score1))

# Calculate the precision of the model
prec_score1 = precision_score(y_test, pred_labels1, average='weighted')
print("\nPrecision: {}".format(prec_score1))

# Calculate the recall of the model
rcll_score1 = recall_score(y_test, pred_labels1, average='weighted')
print("\nRecall: {}".format(rcll_score1))

# Calculate the F1 of the model
f1_score1 = f1_score(y_test, pred_labels1, average='weighted')
print("\nF1: {}".format(f1_score1))

# 用没有组织的数据集进行预测
```

```
# Apply Model to Uknown Groups
# Of the 106,544 observations, 56.25% have a group classification of Unknown.
# Predictor variables
X_unknown = pd.get_dummies(unknown_maj_groups.drop(['gname'], axis=1),
drop_first=True)


# Predict labels on the unknown dataset
pred_labels2 = rf1.predict(X_unknown)

# Get the list of predicted labels for unknown observations
unknown_preds = list(le.inverse_transform(pred_labels2))

# Calculate the counts for each group
unknown_counts = collections.Counter(unknown_preds)

# Top 5 Predicted Unknown Groups
# The top 5 predicted groups account for 97.59% of the unknown observations.

# Top 5 unknown
unknown_top5 = pd.DataFrame(unknown_counts.most_common()[0:25], columns=['Group',
'Attacks'])

cumsum = 0

# Display the top 25 groups with counts
for index, row in unknown_top5.iterrows():
    print("{} : {}".format(row['Group'], row['Attacks']))
    cumsum += row['Attacks']

print("\nTop 25 Account For: {}%".format((cumsum / X_unknown.shape[0])*100))

# Muslim Extremists 伊斯兰极端主义
# Almost half of the predicted unknown group observations are Muslim extremists.
# Set the color palette in reverse
colors = sns.color_palette('Blues', len(unknown_top5))
colors.reverse()
plt.figure(figsize=(14.6, 9.0))

# Plot bar chart with index as y values
ax = sns.barplot(unknown_top5.Attacks, unknown_top5.index, orient='h',
palette=colors)
ax.get_xaxis().set_major_formatter(plt.FuncFormatter(lambda x, loc:
"{:,}".format(int(x))))
```

```
# Reset the y labels
ax.set_yticklabels(unknown_top5.Group)
ax.set_xlabel(xlabel='Number of Attacks', fontsize=16)
ax.set_title(label='Top 25 Predicted Groups', fontsize=20)
plt.show();
```

附录三：K-近邻算法

```
knn = KNeighborsClassifier(n_neighbors = 5)
print("The KNN classifier parameter:\n")
print(knn)

knn.fit(X_train, y_train)

#  返回预测属于某标签的概率
# print(knn.predict_proba(x_eventid))
```

附录四：时间序列模型算法

```
idx = pd.date_range('2015-01-01', '2017-12-31')

iraq_ts = iraq_counts.set_index('incident_date')

iraq_ts = iraq_ts.reindex(idx, fill_value=0)
iraq_ts.head()

print(iraq_ts.describe())

# Daily Plot - Total Attacks
iraq_ts.plot()
plt.title('Iraq Daily Attacks: 2015 - 2017', fontsize=20);
plt.ylabel('Total Per Day')
plt.show();

# Weekly average using the resampled daily data.
# Weekly Plot - Average Attacks
weekly_summary = pd.DataFrame()
weekly_summary['weekly_avg_attacks']                                    =
iraq_ts.daily_attacks.resample('W').mean()

weekly_summary.plot()
plt.title('Iraq Weekly Attacks: 2015 - 2017', fontsize=20);
plt.ylabel('Average Per Week')
plt.show();
```

```
# Monthly Plot - Average Attacks
# Monthly average using the resampled daily data.

monthly_summary = pd.DataFrame()
monthly_summary['monthly_avg_attacks']                                    =
iraq_ts.daily_attacks.resample('M').mean()

monthly_summary.plot()
plt.title('Iraq Monthly Attacks: 2015 - 2017', fontsize=20);
plt.ylabel('Average Per Month')
plt.show();

# Exponential Weighted Moving Average
# Apply smoothing using exponential weighted moving average.
# Use a 30 day span for averaging
iraq_ewm = iraq_ts.ewm(span=30, adjust=False).mean()

iraq_ewm.head()

# Exponential Weighted Moving Average¶
# Daily attacks in Iraq, 2015 to 2017.
iraq_ewm.plot()
plt.title('Iraq Daily Attacks: 2015 - 2017', fontsize=20);
plt.ylabel('Exponential Weighted Moving Average')
plt.show();

# Facebook Prophet
# Create a modified dataset to comply with the Facebook Prophet requirements.
import fbprophet

from fbprophet import Prophet
prophet = Prophet()

iraq_fb = iraq_ts.copy()
iraq_fb['index1'] = iraq_fb.index
iraq_fb.columns = ['y', 'ds']

print(iraq_fb.head())

# Iraq Holidays
# Iraq regional and national holidays covering 2015 - 2017.
# Load the preprocessed GTD dataset
```

```
iraq_holidays = pd.read_csv('./data/iraq-holidays.csv', na_values=[''])
print(iraq_holidays.head())
prophet1 = fbprophet.Prophet(changepoint_prior_scale=0.15,
holidays=iraq_holidays)
prophet1.fit(iraq_fb)

future_data = prophet1.make_future_dataframe(periods=365, freq = 'D')
future_data.tail()
# Predict the values
forecast_data = prophet1.predict(future_data)
print(forecast_data[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail())
#forecast_data.tail()

# Plot the Predictions
prophet1.plot(forecast_data, xlabel = 'Date', ylabel = 'Attacks')
plt.title('Predicted Terrorist Attacks in Iraq', fontsize=20);
plt.show();

prophet1.plot_components(forecast_data)

# 放大去年加上一年的预测
# 预测趋势似乎与成分分析中显示的年度季节性一致。
prophet1.plot(forecast_data, xlabel = 'Date', ylabel = 'Attacks')
plt.title('Predicted Terrorist Attacks in Iraq', fontsize=20);
plt.xlim(pd.Timestamp('2017-01-01'), pd.Timestamp('2018-12-31'))
plt.show();
```

附录五：支持向量机算法

```
# svm 分类

clf = svm.SVC()
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)

from sklearn.metrics import confusion_matrix

confusion_mat = confusion_matrix(y_test,y_pred)

print(confusion_mat)

# Calculate the accuracy
score_svm = accuracy_score(y_test, y_pred)
print("\nAccuracy: {}".format(score_svm))
```

```
def plot_confusion_matrix(confusion_mat):
    plt.imshow(confusion_mat,interpolation='nearest',cmap=plt.cm.Paired)
    plt.title('Confusion Matrix')
    plt.colorbar()
    tick_marks=np.arange(4)
    plt.xticks(tick_marks,tick_marks)
    plt.yticks(tick_marks,tick_marks)
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()

plot_confusion_matrix(confusion_mat)
```

附录六：Seq2Seq 模型算法

```
# 输入层
def get_inputs():
    inputs = tf.placeholder(tf.int32, [None, None], name='inputs')
    targets = tf.placeholder(tf.int32, [None, None], name='targets')
    learning_rate = tf.placeholder(tf.float32, name='learning_rate')

    # 定义 target 序列最大长度（之后 target_sequence_length 和
source_sequence_length 会作为 feed_dict 的参数）
    target_sequence_length = tf.placeholder(tf.int32, (None,),
name='target_sequence_length')
    max_target_sequence_length = tf.reduce_max(target_sequence_length,
name='max_target_len')
    source_sequence_length = tf.placeholder(tf.int32, (None,),
name='source_sequence_length')

    return inputs, targets, learning_rate, target_sequence_length,
max_target_sequence_length, source_sequence_length


# Encoder
"""
在 Encoder 端，我们需要进行两步，第一步要对我们的输入进行 Embedding，再把 Embedding
以后的向量传给 RNN 进行处理。

在 Embedding 中，我们使用 tf.contrib.layers.embed_sequence，它会对每个 batch 执行
embedding 操作。
"""
```

```
def get_encoder_layer(input_data, rnn_size, num_layers, source_sequence_length,
source_vocab_size,
                      encoding_embedding_size):
    """
    构造 Encoder 层

    参数说明：
    - input_data: 输入 tensor
    - rnn_size: rnn 隐层结点数量
    - num_layers: 堆叠的 rnn cell 数量
    - source_sequence_length: 源数据的序列长度
    - source_vocab_size: 源数据的词典大小
    - encoding_embedding_size: embedding 的大小
    """
    #
https://www.tensorflow.org/versions/r1.4/api_docs/python/tf/contrib/layers/embe
d_sequence
    """
    embed_sequence(
    ids,
    vocab_size=None,
    embed_dim=None,
    unique=False,
    initializer=None,
    regularizer=None,
    trainable=True,
    scope=None,
    reuse=None
    )
    ids: [batch_size, doc_length] Tensor of type int32 or int64 with symbol ids.

    return : Tensor of [batch_size, doc_length, embed_dim] with embedded sequences.
    """
    encoder_embed_input = tf.contrib.layers.embed_sequence(input_data,
source_vocab_size, encoding_embedding_size)

    def get_lstm_cell(rnn_size):
        lstm_cell = tf.contrib.rnn.LSTMCell(rnn_size,
initializer=tf.random_uniform_initializer(-0.1, 0.1, seed=2))
        return lstm_cell

    cell = tf.contrib.rnn.MultiRNNCell([get_lstm_cell(rnn_size) for _ in
range(num_layers)])
```

```
    encoder_output, encoder_state = tf.nn.dynamic_rnn(cell, encoder_embed_input,
sequence_length=source_sequence_length,
                                                      dtype=tf.float32)

    return encoder_output, encoder_state

def process_decoder_input(data, vocab_to_int, batch_size):
    ending = tf.strided_slice(data, [0, 0], [batch_size, -1], [1, 1])
    decoder_input = tf.concat([tf.fill([batch_size, 1], vocab_to_int['<GO>']),
ending], 1)

    return decoder_input

def decoding_layer(target_letter_to_int, decoding_embedding_size, num_layers,
rnn_size,
                   target_sequence_length, max_target_sequence_length,
encoder_state, decoder_input):
    '''
    构造 Decoder 层

    参数：
    - target_letter_to_int: target 数据的映射表
    - decoding_embedding_size: embed 向量大小
    - num_layers: 堆叠的 RNN 单元数量
    - rnn_size: RNN 单元的隐层结点数量
    - target_sequence_length: target 数据序列长度
    - max_target_sequence_length: target 数据序列最大长度
    - encoder_state: encoder 端编码的状态向量
    - decoder_input: decoder 端输入
    '''


    # 1. Embedding
    target_vocab_size = len(target_letter_to_int)
    decoder_embeddings = tf.Variable(tf.random_uniform([target_vocab_size,
decoding_embedding_size]))
    decoder_embed_input = tf.nn.embedding_lookup(decoder_embeddings,
decoder_input)

    # 构造 Decoder 中的 RNN 单元
    def get_decoder_cell(rnn_size):
        decoder_cell = tf.contrib.rnn.LSTMCell(rnn_size,
initializer=tf.random_uniform_initializer(-0.1, 0.1, seed=2))
        return decoder_cell
```

```
    cell = tf.contrib.rnn.MultiRNNCell([get_decoder_cell(rnn_size) for _ in
range(num_layers)])

    # Output 全连接层
    # target_vocab_size 定义了输出层的大小
    output_layer = Dense(target_vocab_size,
kernel_initializer=tf.truncated_normal_initializer(mean=0.1, stddev=0.1))

    # 4. Training decoder
    with tf.variable_scope("decode"):
        training_helper =
tf.contrib.seq2seq.TrainingHelper(inputs=decoder_embed_input,

sequence_length=target_sequence_length,
                                                          time_major=False)

        training_decoder = tf.contrib.seq2seq.BasicDecoder(cell, training_helper,
encoder_state, output_layer)
        training_decoder_output, _, _ =
tf.contrib.seq2seq.dynamic_decode(training_decoder, impute_finished=True,

maximum_iterations=max_target_sequence_length)

    # 5. Predicting decoder
    # 与 training 共享参数

    with tf.variable_scope("decode", reuse=True):
        # 创建一个常量 tensor 并复制为 batch_size 的大小
        start_tokens = tf.tile(tf.constant([target_letter_to_int['<GO>']],
dtype=tf.int32), [batch_size],
                               name='start_token')
        predicting_helper =
tf.contrib.seq2seq.GreedyEmbeddingHelper(decoder_embeddings, start_tokens,

target_letter_to_int['<EOS>'])

        predicting_decoder = tf.contrib.seq2seq.BasicDecoder(cell,
                                                             predicting_helper,
                                                             encoder_state,
                                                             output_layer)
        predicting_decoder_output, _, _ =
tf.contrib.seq2seq.dynamic_decode(predicting_decoder, impute_finished=True,

maximum_iterations=max_target_sequence_length)
```

```
    return training_decoder_output, predicting_decoder_output
# 上面已经构建完成 Encoder 和 Decoder，下面将这两部分连接起来，构建 seq2seq 模型
def seq2seq_model(input_data, targets, lr, target_sequence_length,
max_target_sequence_length,
                  source_sequence_length, source_vocab_size, target_vocab_size,
encoder_embedding_size,
                  decoder_embedding_size, rnn_size, num_layers):
    _, encoder_state = get_encoder_layer(input_data,
                                         rnn_size,
                                         num_layers,
                                         source_sequence_length,
                                         source_vocab_size,
                                         encoding_embedding_size)
    decoder_input = process_decoder_input(targets, target_letter_to_int,
batch_size)
training_decoder_output, predicting_decoder_output =
decoding_layer(target_letter_to_int,
decoding_embedding_size,
num_layers,
rnn_size,
target_sequence_length,
max_target_sequence_length,
encoder_state,
decoder_input)
    return training_decoder_output, predicting_decoder_output
```