

队伍编号	MCB2400166
赛道	A

# 基于机器学习方法的台风分类、路径预测及登陆对降水影响研究

## 摘要

台风作为世界上最严重的自然灾害之一，其强大的破坏性和复杂性对人类特别是居住在沿海地区的生产生活以及生命财产安全都产生非常大的影响，因此对台风的分类与预测进行研究就显得尤为重要。本文基于历史台风数据，针对台风不同特征分类、路径预测以及登陆后的降水量和风速变化进行了详细分析，并结合多种基于机器学习方法的模型进行求解。

本文首先对历史台风数据进行数据预处理，由于在原始数据集中存在一些不规范性例如气压、台风强度和台风等级存在空白值、移动方向和移动速度部分数据缺失。本文分别对这些问题进行处理，首先是将空白值进行删除以及对时间日期进行格式转换，然后通过对每条台风的经纬度进行分析获取每条台风的移动方向并且通过采用 Haversine 公式求得每条台风的移动速度，最后通过数值化操作将台风强度这些文本数据编码成数字以便用于后续模型的使用。

**针对问题一：**本文首先通过对历史台风数据进行可视化分析得到台风各项特征的分布情况以及与气压、季风之间的关系，然后本文首先根据季节分割数据，创建夏季和秋季的台风数据子集。接下来使用 Kmeans 算法，将台风等级、风速、气压、移动方向、移动速度以及经纬度作为聚类特征来分别求得夏季台风和秋季台风的聚类关系并且绘制成图以此来初步判断夏季台风和秋季台风的特点与区别，然后本文通过分别建立随机森林、支持向量机、LSTM 和图神经网络四种不同的分类模型，用 Precision 和 F1 作为评估指标找到最优的模型并且将这些模型应用于后续对所收集到的 2024 年的台风数据集进行分类得到最终的分类结果。

**针对问题二：**本文首先使用和问题一一样的缺失值处理方法，对原始数据进行缺失值处理、时间格式转换、台风强度、移动方向数值化映射、数据编码。在处理好缺失值根据数据特征构建特征工程，其中我们提取了原始数据的时间特征、周期性特征、历史特征计算、距离特征、台风持续特征、季节性特征等特征，为后续的模型训练提供信息，提高模型的预测精度。因为台风数据具有时间序列和结构化的特征，所以我们最后分别使用了 LSTM、随机森林、XGBoost、LightGBM 和 CatBoost 模型五种不同的模型，利用历史台风数据训练模型，以台风强度，风速，气压，台风等级等多个特征作为输入，经度和维度作为输出，从而预测台风的路径，并且根据误差总和归一化误差的方式来分配权重，误差越小，权重越大。最后基于权重使用加权投票的模型来预测新数据的台风路径，并运用 Dynamic Time Warping(DTW) 动态时间规整算法与台风实际行进路线进行对比

**针对问题三：**本文的目标是研究台风登陆后降水量与风速的变化模式。通过结合 Holland 公式和一个经拟合确定的参数模型，我们分析了降水量如何受台风中心距离、风速以及气压等因素的影响。基于对贝碧嘉台风案例的研究，我们发现降水量呈现出随距离及风速变化的非线性趋势。本文的创新之处在于将 Holland 模型与一个经过拟合优化的降水公式结合起来，从而能够更全面地描述台风登陆的影响，并提高了降水量分布预测的准确性。综上，本研究采用数据驱动与物理模型相结合的方法，系统地分析了台风的分类、路径预测及其登陆后的影响。这为应对台风灾害提供了科学依据和支持。这些模型的应用不仅加深了我们对台风行为的理解，还为台风的预测和防范工作提供了强有力的工具。

**关键词：**台风分类、Kmeans、随机森林、LSTM、路径和降水量预测、XGBoost、Holland

## 目录

一、 问题背景与重述 .....	1
1.1 问题背景 .....	1
1.2 问题重述 .....	1
二、 问题分析 .....	1
三、 模型假设 .....	2
四、 符号说明 .....	3
五、 问题一模型建立与求解 .....	3
5.1 数据预处理 .....	3
5.1.1 缺失值处理 .....	3
5.1.2 数值化编码 .....	4
5.2 数据可视化 .....	4
5.3 分类评价模型建立 .....	7
5.3.1 基于 K-means 算法进行分类 .....	7
5.3.2 多种模型的建立 .....	8
六、 问题二模型建立与求解 .....	11
6.1 数据预处理 .....	11
6.1.1 数据描述性分析 .....	11
6.1.2 特征工程 .....	13
6.2 基本预测模型 .....	14
6.3 集合模型路径预测 .....	19
6.4 预测结果分析 .....	20
6.4.1 台风贝碧嘉的数据预处理 .....	20
6.4.2 台风贝碧嘉数据预测和分析 .....	20
6.5 运用动态时间规整算法与实际路线做对比 .....	22
七、 问题三模型的建立与求解 .....	23
7.1 数据预处理 .....	23
7.2 建立模型 .....	23
7.2.1 风速与距离的关系模型 .....	24
7.2.2 降水量与距离的关系模型 .....	24
7.3 可视化分析 .....	24
7.4 建立综合的预测模型 .....	26
7.4.1 风速与台风中心路径 .....	26
7.4.2 使用拟合的降水量公式计算降水量 .....	27
7.4.3 可视化分析 .....	27
八、 模型总结 .....	28
8.1 模型优点 .....	28
8.2 模型的缺点 .....	28
九、 参考文献 .....	29
十、 附录 .....	30
10.1 问题一代码 .....	30
10.2 问题二代码 .....	55
10.3 问题三代码 .....	79

## 一、问题背景与重述

### 1.1 问题背景

台风作为一种热带气旋，与人类的生活及生产活动密切相关，同时也是重要的降雨系统。尽管其带来降水对农业等方面有益，但台风同样会造成诸如狂风、暴雨、风暴潮、泥石流等一系列自然灾害，并且常常伴随有生态破坏以及疫病流行等问题，这些灾害特征表现为突发性强、破坏力大，因此被认为是全球最严重的自然灾害之一。

关于台风的形成原因，科学界目前的认识是，其起源于热带大气中的某些扰动。具体来说，在热带海域，由于太阳直射导致海水升温，从而增加了水的蒸发率，使得空气中的湿度增加。温暖且湿润的空气因温度升高而膨胀，密度随之降低，进而变得更为轻盈，易于上升，并在这一过程中引发对流现象。与此同时，周围的冷空气会流入补充上升的暖湿空气，这一过程不断循环，最终导致一个由温暖、轻盈、低密度空气组成的气旋体——即低气压系统的形成。

台风的运动轨迹极为复杂，其移动不仅受内部动力学机制影响，如北半球台风特有的反时针旋转带来的偏向效应，还受到外部环境如副热带高压等天气系统的作用。副热带高压的位置及其强度变化能够显著改变台风的行进方向，使其向西或向北甚至东北方向移动。此外，台风间的相互作用也可能导致它们的路径发生变化，包括停滞或打转等情况。由于这些复杂的交互作用，预测台风的精确路径仍然是气象学的一大挑战。

近些年来国内外学者运用多种方法对台风进行分类和预测，以提高预警能力和防灾减灾水平。在国外，研究主要集中于气象模型和机器学习技术的结合。例如，Kossin 等（2014）<sup>[1]</sup>提出了一种基于统计模型的方法，利用历史台风数据对台风路径进行预测，显著提高了预测精度。Chavas 和 Emanuel（2010）<sup>[2]</sup>通过数值模型对台风强度变化进行模拟，探讨了气温和海洋条件对台风强度的影响。近年来，深度学习技术也逐渐被应用于台风预测，如 Yuan 等（2020）<sup>[3]</sup>利用卷积神经网络（CNN）对台风路径进行预测，取得了良好的效果。在国内，研究者多采用随机森林、支持向量机等机器学习算法进行台风分类。例如，李华（2021）<sup>[4]</sup>结合气象要素，通过随机森林模型对台风进行分类，结果显示模型的准确率和召回率均优于传统方法。此外，利用大数据和云计算技术，许多研究者开始探索实时台风监测和预测系统的构建。

### 1.2 问题重述

**问题 1：**分析台风特征参数（强度、等级、风速等）与气温、气压、季风的关系，依据气温、气压、季风等因素的影响，建立台风的分类评价模型，明确类别划分的标准或评价算法，进行不同特征的台风类别划分。夏台风生成于 6-8 月，秋台风 9-11 月，根据所建立的分类评价模型，以 2024 年我国 7 月和 9 月为例，给出台风类别及途经省份的列表，并分析夏台风与秋台风的区别。

**问题 2：**除温度、气压、季风外，台风的路径还受地球自转偏向力等多重因素影响。根据气温、气压、洋流、风场等多种因素，建立台风路径预测的模型。并且，以 2024 年 9 月 13 日-1 日每日 14 点的第 13 号台风贝碧嘉为例，预测行进路线，运用 Dynamic Time Warping(DTW)动态时间规整算法与台风实际行进路线进行对比。

**问题 3：**台风登陆后，风速和雨量将逐渐减弱，且某地区受台风影响而产生的风速和降水量，随着其与台风中心距离的增加而减小。请建立台风在登陆后的行进过程中降水量及风速的关系，及降水量与距台风中心距离的关系。并且，以 2024 年 9 月 16 日-18 日的第 13 号台风贝碧嘉为例，根据所建立的模型，预测贝碧嘉行进途中的中心风力及降水量，进行分析。

## 二、问题分析

为了解决上述提出的研究问题，本文采用以下求解思路进行数据处理与模型建立：

首先，在本题目所给出的涵盖从 1945 年到 2023 年所有台风信息及其相关特征数据集中。我们需要对这些数据进行必要的处理，包括缺失值处理、时间格式转换、数据编码和描述性分析。

1、缺失值处理：针对数据集中台风名称和台风强度数据项里面存在的大量缺失值，采用直接删除含有缺失值的行，以提高数据的完整性和一致性。

2、时间处理：将数据中的时间格式转换为%Y/%M/%D %H:%M 的标准格式，以便计算机能够识别和进一步分析。特别是针对六小时间隔的数据，确保时间序列的连续性和一致性。

3、数据编码：对于台风数据中使用汉字或字母进行编码的内容，需要转换为数值形式，以便在模型中应用。例如，台风等级和移动方向这些信息都是汉字形式需要数值化表示。

4、数据描述性分析：通过可视化手段对台风的路径和特征参数进行描述性分析，为后续模型的建立提供数据支持。

#### 问题 1：台风特征参数与气象因素的关系及分类评价模型

关于台风的强度、等级、风速、气压以及经纬度、季风等特征参数，本文首先对这些特征参数的分布进行可视化分析例如台风月份分布、不同台风的运行轨迹线路等等，然后对这些特征参数进行相关性分析，利用斯皮尔曼相关系数找出与台风强度和等级高度相关的气象因素，并使用热力图进行可视化展示，明确不同特征之间的关系。接着关于探索夏季风与秋季风的特点和区别时，本文根据季节分割数据，创建夏季和秋季的台风数据子集然后使用 K-means 算法，将台风等级、风速、气压、移动方向、移动速度以及经纬度作为聚类特征来分别求得夏季台风和秋季台风的聚类关系并且绘制成图，通过图例来更好的分析之间的区别。最后本文分别使用随机森林、支持向量机、LSTM 和图神经网络四种不同的分类模型，利用历史台风数据训练模型，以气温、气压、季风等作为输入，台风等级或类型作为输出，从而实现对台风类别的划分。并且通过 Precision 和 F1 作为评估指标找到最优的模型用于后续操作。

问题 2：台风路径预测问题中，路径的影响因素非常复杂，涉及温度、气压、风场等多方面数据，具有高度非线性。这些因素相互作用，使得台风的路径具有高度的不确定性。因此，建立一个能够准确预测台风路径的模型，对于提高预警能力和防灾减灾水平具有重要意义。为了解决台风路径预测的问题，本文首先使用和问题一一样的缺失值处理方法，对原始数据进行缺失值处理、时间格式转换、台风强度、移动方向数值化映射、数据编码。在处理好缺失值根据数据特征构建特征工程，其中我们提取了原始数据的时间特征、周期性特征、历史特征计算、距离特征、台风持续特征、季节性特征等特征，为后续的模型训练提供信息，提高模型的预测精度。因为台风数据具有时间序列和结构化的特征，所以我们最后分别使用了 LSTM、随机森林、XGBoost、LightGBM 和 CatBoost 模型五种不同的模型，利用历史台风数据训练模型，以台风强度，风速，气压，台风等级等多个特征作为输入，经度和维度作为输出，从而预测台风的路径，并且根据误差总和归一化误差的方式来分配权重，误差越小，权重越大。最后基于权重使用加权投票的模型来预测新数据的台风路径，并运用 Dynamic Time Warping(DTW) 动态时间规整算法与台风实际行进路线进行对比

问题 3：台风登陆后，风速和雨量将逐渐减弱，且某地区受台风影响而产生的风速和降水量，随着其与台风中心距离的增加而减小。请建立台风在登陆后的行进过程中降水量及风速的关系，及降水量与距台风中心距离的关系。并且，以 2024 年 9 月 16 日-18 日的第 13 号台风贝碧嘉为例，根据所建立的模型，预测贝碧嘉行进途中的中心风力及降水量，进行分析。

### 三、模型假设

为了有效建立台风分类模型、路径预测模型以及登陆后降水与风速关系的模型，本研究提出以下模型假设：

假设 1：台风的强度、等级、风速等特征参数与气温、气压、季风等气象因素之间存在较强的线性或非线性相关性。

假设 2：不同季节（如夏季台风和秋季台风）形成的台风，其特征参数存在显著差异，且可以通过这些差异来进行有效分类。

假设 3：台风路径的预测台风的路径受气温、气压、洋流、风场等多种因素的共同作用影响，

这些影响在一定的时间尺度上具有可预测性。

假设 4：台风登陆后的降水与风速关系台风登陆后，风速和降水量与距离台风中心的距离之间存在一定的函数关系，并且这种关系可以通过物理机制和统计建模相结合的方法描述。

## 四、符号说明

本文主要变量符号说明：

符号	说明	单位
$\Delta \text{lng}$	同一台风下上一纬度值和这一纬度值的差值	°
$\Delta \text{lng}$	同一台风下上一经度值和这一经度值的差值	°
R	地球半径	KM
$V(r)$	距台风中心为 $r$ 的位置的风速	m/s
$r$	距离台风中心的距离	km
P	气压	hPa

注：未申明的变量以其在符号出现处的具体说明为准

## 五、问题一模型建立与求解

### 5.1 数据预处理

#### 5.1.1 缺失值处理

在原始数据集中，气压、台风等级、台风强度、移动方向和移动速度这些台风重要特征因素都存在大量的缺失值，对于这些缺失值本文选择直接删除和通过公式计算来进行填充这两种方法以此来保证数据的完整性。具体而言本文将气压，台风强度和台风等级这三列中的缺失值我们选择直接删除，而对于移动方向和移动速度中的缺失值我们选择通过结合经纬度数据通过数学计算的方法进行求取并补充，具体计算过程为：

首先计算每条台风的移动方向，计算过程可表示为：

$$\text{移动方向} = \text{atan2}(\Delta \text{lng}, \Delta \text{lat})$$

求取移动方向后，接下来首先通过运用 Haversine 公式来计算每条台风的移动距离，其具体计算过程可表示为：

$$\begin{aligned}\alpha &= \sin^2\left(\frac{\Delta \text{lng}}{2}\right) + \cos(\text{lng1}) \cdot \cos(\text{lng2}) \cdot \sin^2\left(\frac{\Delta \text{lat}}{2}\right) \\ C &= 2 \cdot \alpha \tan2(\sqrt{\alpha}, \sqrt{1 - \alpha}) \\ S &= R \cdot C\end{aligned}$$

然后再将所求得的移动距离除以当前台风时间与上一台风时间的差值便可得到最终的移动速度，其计算过程可表示为：

$$\Delta v = \frac{S}{\Delta t}$$

经过上述处理后，移动方向和移动速度值如表 1 所示

表 5.1 计算后的移动方向和移动速度数据

台风编号	移动方向	移动速度
194901	104.0362435	22.90224254
194902	79.70972338	28.258523
194908	83.84647009	29.82658092

### 5.1.2 数值化编码

为了在后续模型训练中能够有效利用台风强度的分类信息，本文对台风强度的数据进行了必要的数值化编码转换。由于原始数据中使用汉字或者字母表示台风的强度类别，如“超强台风（Super TY）”、“强热带风暴（STS）”等，为了便于处理并应用于后续模型建立，本文通过数字 1-6 将这些分类转换为数值编码。具体编码方案如下：

1. 超强台风(Super TY) : 1
2. 强热带风暴(STS) : 2
3. 强台风(STY) : 3
4. 热带低压(TD) : 4
5. 热带风暴(TS) : 5
6. 台风(TY) : 6

经过数值化编码后，台风强度值如表 2 所示：

表 5.2 数值化编码的台风强度部分数据

台风编号	原台风强度	台风强度
194901	热带风暴(TS)	5
194908	热带低压(TD)	4
202316	热带低压(TD)	4

## 5.2 数据可视化

在完成数据预处理之后，由于问题一探讨的是台风特征参数（强度、等级、风速等）与气温、气压、季风之间的关系，因此本文首先将从 1945-2023 年台风各特征参数的分布情况通过图示展示出来：

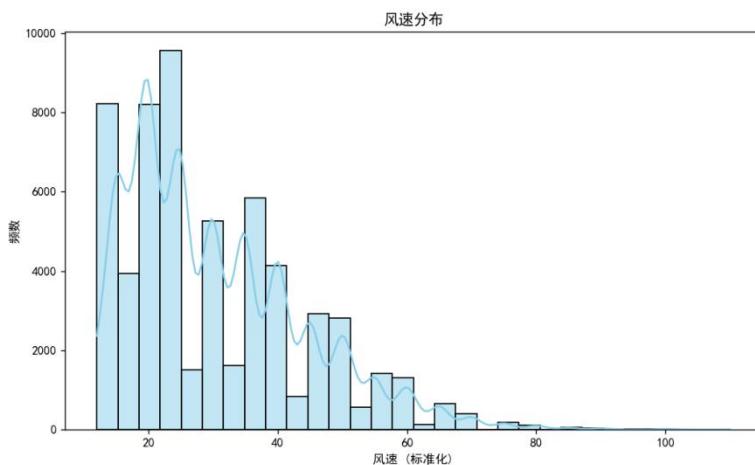


图 5.1 1945-2023 台风风速分布图

首先是关于台风风速分布情况的柱状图，从图中可以看到大多数情况下，低至中等程度的风速(10-30)占据了主导地位；然而当风速变得更高时，则呈现出明显的下降趋势——这意味着极端天气条件下的大风事件相对来说比较少见。但是也不能忽视在风速到达 60 的时候图中也形成了一个小高峰，表明在 1945-2023 这期间也出现了一定量的极端台风天气例如 2023 年的超强台风布拉万。不过

总的来说绝大多数时候人们所经历的都是较为温和的台风气候状况。

接下来本文根据台风发生时所处月份建立了台风月份分布直方图如图 2 所示

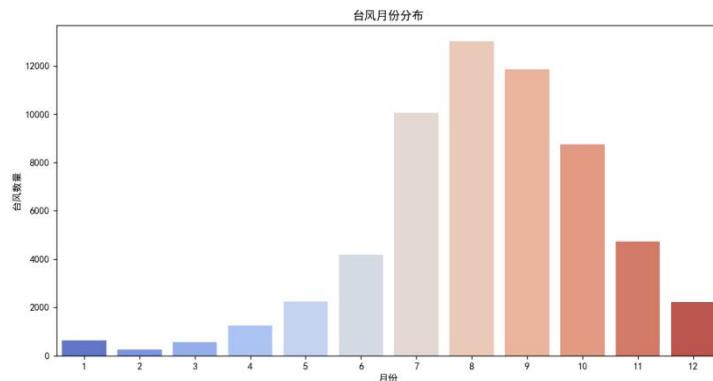


图 5.2 台风月份分布图

从图中可以清楚的看出从 1945 到 2023 年期间台风在春冬季节发生的频率非常小，但是随着时间的不断推移当逐渐进行夏季的时候台风活动的频率急速上升并且在 8 月份到达了顶峰，因此可以得出夏季风的一个特点是台风频发，虽然随着进入秋季台风数量明显减少但是也依然处于比较高的频率，对于人类的生产生活同样也会产生很大的影响。

同时为了更好的分析研究台风的活动轨迹，本文从数据集中随机选取了 20 条台风并且通过它们之间经纬度的变化情况结合世界地图绘制成台风活动轨迹图如图 3 所示：

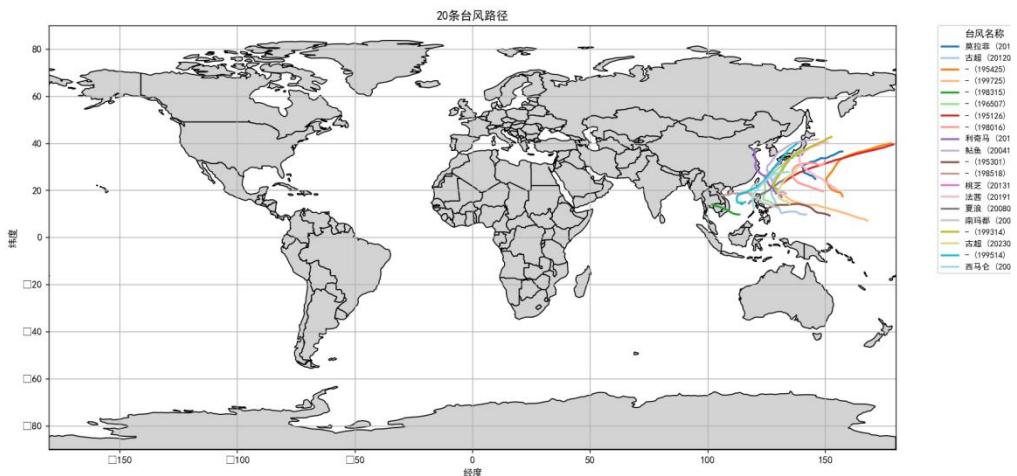


图 5.3 台风移动路径轨迹图

通过观察这些路径可以看出，大部分台风路径集中在东亚和东南亚的海域，尤其是南海、东海和日本附近海域。台风路径主要呈现从东南向西北的移动趋势，且在接近东亚沿海时普遍偏向北方或西北方，这与台风在生成后通常向偏北方向移动的模式相吻合。这一趋势与太平洋副热带高压、季风系统等气象因素密切相关。此外，部分台风路径向东延伸，显示出少数台风可能向偏东的太平洋深处移动，甚至接近北美西海岸。这样的东向移动较为罕见，通常在特殊的气象条件下发生，如受到强烈偏西风的影响。这些路径也揭示了台风路径的不确定性，即使在同一地区生成的台风，其路径和移动方式也可能出现显著差异。

在分析完台风各个特征因素分布情况后，本文接下来对这些特征参数进行相关性分析，考虑到台风特征数据（如台风强度、风速、气压等）不符合正态分布，因此本文使用 Spearman<sup>[5]</sup> 相关系数来分析变量之间的关系。Spearman 相关系数是一种非参数的相关性度量方法，用于衡量两个变量之间的单调关系，适合处理非正态分布的数据。

Spearman 相关系数的具体计算公式为：

$$\rho = 1 - \frac{6 \sum_{i=1}^n d_i^2}{n(n^2 - 1)}$$

其大致计算过程可以描述为存在两个变量  $X$  和  $Y$ , 它们分别包含大量观测值  $n$ , 首先分别将它们的观测值进行排序并为每个值分配排名(如果存在相同的值(即平级情况), 则为这些相同的值分配其平均排名。例如, 假设变量  $X$  的观测值中有两个相等的值分别在第 3 位和第 4 位, 那么这两个值的排名都为  $(3+4)/2=3.5$ ), 然后接下来就分别计算  $X, Y$  中两者的每个观测值的排位差:

$$d_i = rank(X_i) - rank(Y_i)$$

求得每个观测值的排位差  $d_i$  后再对其进行平方求和:

$$\sum_{i=1}^n d_i^2$$

最后在带入 Spearman 相关系数公式后便可以求得最终的相关系数  $\rho$ 。

其中  $\rho$  的取值范围为  $[-1, 1]$ :

- ① 当  $\rho=1$  时, 表示两个变量完全正相关, 且排名完全一致。
- ② 当  $\rho=-1$  时, 表示两个变量完全负相关, 且排名完全相反。
- ③ 当  $\rho=0$  时, 表示两个变量之间没有单调关系。

在本文研究中, 我们将台风强度、台风等级、风速、气压、移动方向和移动速度这些特征因素作为变量带入 Spearman 相关系数来求得每个变量之间的相关性情况, 其计算结果如下表 5.3 所示:

表 5.3 Spearman 相关系数结果

	台风强度	台风等级	风速	气压	移动方向	移动速度
台风强度	1.0	-0.2	-0.2	0.19	-0.0046	-0.02
台风等级	-0.2	1.0	1.0	-0.94	-0.019	0.013
风速	-0.2	0.996	1.0	-0.94	-0.019	0.016
气压	0.19	-0.94	-0.94	1.0	0.05	-0.012
移动方向	-0.0046	-0.019	-0.019	0.05	1.0	-0.08
移动速度	-0.02	0.013	0.016	-0.012	-0.08	1.0

为了更直观地展示各个变量之间的相关性, 本文绘制了 Spearman 相关性矩阵的热图。在热图中, 颜色的深浅代表相关性的强弱。颜色越接近深红色, 代表变量之间的正相关性越强, 颜色越接近深蓝色, 代表变量之间的负相关性越强, 其热力图如图 4 所示:

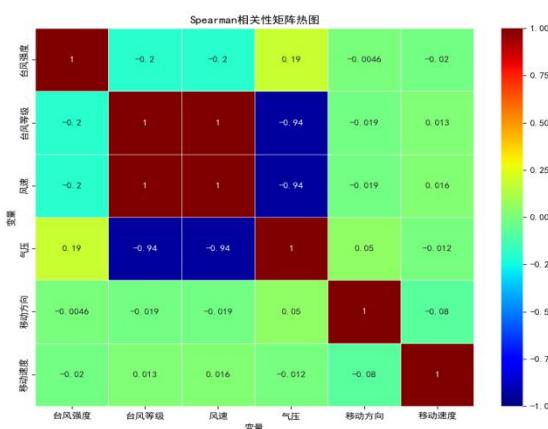


图 5.4 Spearman 相关性矩阵的热图

从图中可以看到, 风速与台风等级的 Spearman 相关系数达到了 1, 这意味着风速越高, 台风等级也越高。这一结果与台风的物理特性相符: 风速是决定台风等级的关键因素, 台风的破坏力主要由风速决定, 因此两者间表现出极强的正相关性。此外, 气压与风速之间的 Spearman 相关系数

为 -0.94，呈高度负相关，这与气象学中的常识一致——气压越低，风速往往越大。同样地，气压与台风等级之间的相关系数也为 -0.94，表明低气压通常对应更高的台风等级，这一结果与台风形成的物理原理相符。在分析移动速度时，发现其与其他变量的相关性较弱。移动速度与气压的 Spearman 相关系数为 -0.012，表明气压对移动速度的影响微弱。此外，移动速度与台风等级的相关系数为 0.013，与风速的相关系数为 0.016，均显示出移动速度的变化无法单独通过风速或台风等级来解释。移动速度可能更多地受到大气环流系统的影响，如副热带高压的强弱和洋流的变化等外部因素。

综合来看，Spearman 相关性分析结果揭示了台风特征参数与气象因素之间的重要关系。风速、台风等级和气压之间存在显著的相关性，这些变量是台风强度和破坏力的主要决定因素。而移动方向和移动速度与其他变量的相关性较弱，表明这些因素受到更复杂的外部条件影响。对各变量关系的分析为后续的台风分类和路径预测提供了关键的特征选择依据，有助于更深入理解台风的形成和演变机制。

## 5.3 分类评价模型建立

在完成数据预处理和可视化操作后，为了对不同特征的台风类别进行划分和分析夏季风与秋季风的差别，本文开始通过基于机器学习和深度学习的方法来建立多种分类评价模型并且后续通过加入评估指标 Precision 和 F1 来找到最优模型并且用于最后 2024 年的台风的分类预测任务。

### 5.3.1 基于 K-means 算法进行分类

本文首先探索夏季风与秋季风的特点和区别，根据数据集中的月份数据来定义季节函数，为每个台风记录添加了季节特征，分为夏季和秋季。接着通过选取特征（如台风等级、风速、气压等）并进行标准化，以便进行聚类分析。使用 K-means 算法将夏季和秋季的台风数据分别聚类为三个类别来进行计算以此分别求得夏季台风和秋季台风的聚类关系并且绘制成图，通过图例来更好的分析之间的区别。其分类算法如下所示：

算法 1	K-means 分类算法
1	// 定义季节函数
2	get_season(m) =
3	IF m ∈ {6, 7, 8} THEN 'Summer'
4	ELSE IF m ∈ {9, 10, 11} THEN 'Autumn'
5	ELSE 'Other'
6	// 添加季节特征
7	D['Season'] ← apply(get_season, D['月'])
8	// 分割数据集按季节
9	D_summer ← D[D['Season'] = 'Summer']
10	D_autumn ← D[D['Season'] = 'Autumn']
11	// 选择用于 K-means 分类的特征
12	F_summer ← D_summer[['台风等级', '风速', '气压', '移动方向', '移动速度', '经度', '纬度']]
13	F_autumn ← D_autumn[['台风等级', '风速', '气压', '移动方向', '移动速度', '经度', '纬度']]
14	// 标准化特征
15	F*_summer ← StandardScaler().fit_transform(F_summer)
16	F*_autumn ← StandardScaler().fit_transform(F_autumn)
17	// K-means 聚类
18	C_summer ← KMeans(n_clusters=3, random_state=42).fit_predict(F*_summer)
19	C_autumn ← KMeans(n_clusters=3, random_state=42).fit_predict(F*_autumn)
20	// 添加聚类结果到数据集
21	D_summer['Cluster'] ← C_summer

经过分类后得到的夏季风和秋季风分类散点图如下所示：

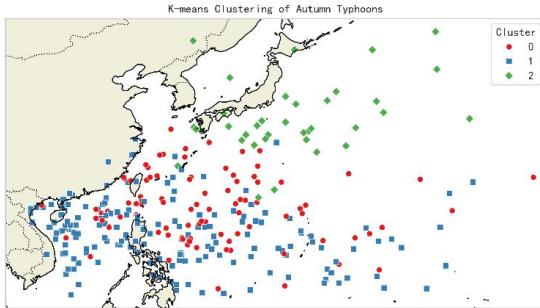


图 5.5 秋季风分类散点图

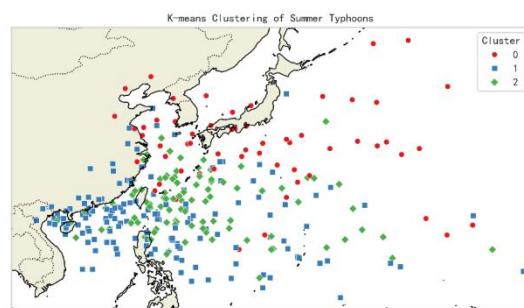


图 5.6 夏季风分类散点图

图中红色点代表的是低强度的台风，1 代表的是中强度的台风，2 代表的是高强度的台风，从两张图对比可以看出夏季台风一般都在海洋中形成高强度台风，因为海温较高，有利于热带气旋的生成和发展。然后不断向西北方向移动当慢慢接近大陆沿岸时风速就会逐渐减小变为低强度的台风，而秋季台风刚好相反，在海洋里面形成的时候往往风速很小但是随着不断迁移靠近内陆后风速越变越大。

### 5.3.2 多种模型的建立

在完成对夏季风和秋季风的区别和特点进行探索后，本文接下来将探讨如何建立有效的分类评价模型，首先我们选择经过处理后的 1945-2023 台风数据集作为模型训练和验证的数据集并且按照 7:3 的比例将数据集拆分成训练集和测试集。同时为了使模型能够对台风进行有效分类，我们选择了台风等级、移动方向、风速、气压和移动速度作为分类模型的输入特征。

本文主要使用了随机森林、支持向量机、LSTM 和图神经网络四种基线模型来进行模型训练并且通过使用两个评估值 Precision 和 F1 来进行评估并找到最优模型。

- **随机森林 (Random Forest)** <sup>[6]</sup>:一种集成学习算法，主要用于分类和回归任务。它通过构建多个决策树，并将它们的结果结合起来以提高预测的准确性和控制过拟合。
- **支持向量机 (Support Vector Machine, SVM)** <sup>[7]</sup>:一种强大的监督学习算法，主要用于分类和回归分析。SVM 的核心思想是找到一个最佳超平面，以最大化类间的间隔，从而实现对数据的有效分类。
- **长短期记忆网络 (Long Short-Term Memory, LSTM)** <sup>[8]</sup>:一种特殊类型的递归神经网络 (RNN)，专门设计用于处理和预测时间序列数据。LSTM 的关键优势在于其能够有效地捕捉长期依赖关系，从而克服了传统 RNN 在长序列训练中遇到的梯度消失问题。

本文所采用的 LSTM 模型采用顺序结构，包含多个 LSTM 层，用于捕捉时间序列数据中的长期依赖关系。每个 LSTM 层之后都添加了 Dropout 层以防止过拟合，并使用 BatchNormalization 层来加速训练和提高模型稳定性。模型的输出层使用 Softmax 激活函数，其具体算法如下表所示：

算法 2	LSTM 算法
1	model = Sequential()
2	model.add(LSTM(100, return_sequences=True, input_shape=(X_train.shape[1], 1)))
3	model.add(Dropout(0.2))
4	model.add(BatchNormalization())
5	model.add(LSTM(50, return_sequences=True))

---

```

6   model.add(Dropout(0.2))
7   model.add(BatchNormalization())
8   model.add(LSTM(25))
9   model.add(Dropout(0.2))
10  model.add(Dense(20, activation='relu'))
11  model.add(Dense(y_categorical.shape[1], activation='softmax'))

```

---

- 图神经网络 (Graph Neural Network, GNN)<sup>[9]</sup>: 一种针对图结构数据进行学习的深度学习模型。GNN 能够有效地处理非欧几里得数据，适用于社交网络、化学分子、知识图谱等多种应用场景。

本文所设计的图神经网络模型包含多个图卷积层 (GCNConv) 和图注意力层 (GATConv)，用于处理图结构数据。模型首先通过一个 GCN 层提取节点特征，然后通过一个带有多个头的 GAT 层进一步增强特征表示能力，接着再次经过一个 GCN 层整合信息。每一层之后都应用了 ReLU 激活函数和 Dropout 层来增加非线性和防止过拟合。最后，通过一个全连接层 (Linear 层) 将提取到的特征映射到分类标签空间，输出每个节点属于不同类别的概率分布。其具体算法如下所示：

---

算法 3	图神经网络算法
------	---------

```

1  class GNNModel(nn.Module):
2      def __init__(self):
3          super(GNNModel, self).__init__()
4          self.conv1 = GCNConv(X.shape[1], 64)
5          self.conv2 = GATConv(64, 64, heads=4)
6          self.conv3 = GCNConv(64 * 4, 128)
7          self.dropout = nn.Dropout(0.3)
8          self.fc = nn.Linear(128, len(np.unique(y_encoded)))
9      def forward(self, data):
10         x, edge_index = data.x, data.edge_index
11         x = self.conv1(x, edge_index)
12         x = torch.relu(x)
13         x = self.dropout(x)
14         x = self.conv2(x, edge_index)
15         x = torch.relu(x)
16         x = self.dropout(x)
17         x = self.conv3(x, edge_index)
18         x = torch.relu(x)
19         x = self.dropout(x)
20         x = self.fc(x)
21         return x

```

---

本文分别将这四种模型在经过处理后的 1945-2023 台风数据集上进行训练后得到的评估值如下表 5.4 所示：

表 5.4 实验结果

模型	Precision	F1
----	-----------	----

随机森林	0.56	0.56
支持向量机	0.46	0.38
LSTM	0.51	0.49
图神经网络	0.4768	0.3842

从表中结果可以看出随机森林模型的精确率和 F1 值均为 0.56，表现最佳，表明其在台风分类任务中具有良好的识别能力。相比之下，支持向量机的精确率为 0.46，F1 值仅为 0.38，表明其在处理该数据集时的效果相对较差。LSTM 模型取得了 0.51 的精确率和 0.49 的 F1 值，显示出一定的预测能力。图神经网络的表现最差，精确率为 0.4768，F1 值为 0.3842 可能是因为数据关系不明显导致较差的效果。这些结果表明，随机森林在台风分类中表现突出。

同时本文还分别将四种模型训练所得到的数据通过混淆矩阵以可视化的方式展现出来：

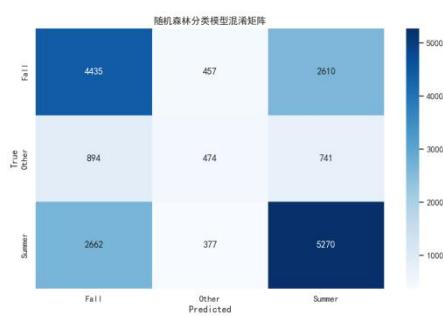


图 5.7 随机森林混淆矩阵

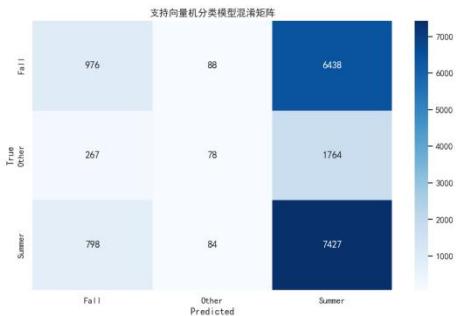


图 5.8 支持向量机混淆矩阵

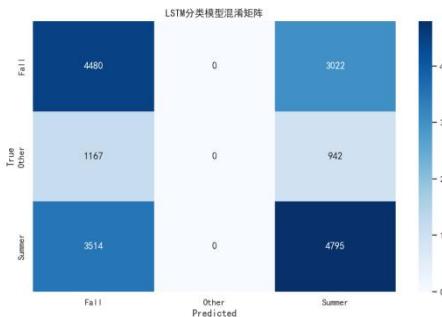


图 5.9 LSTM 混淆矩阵

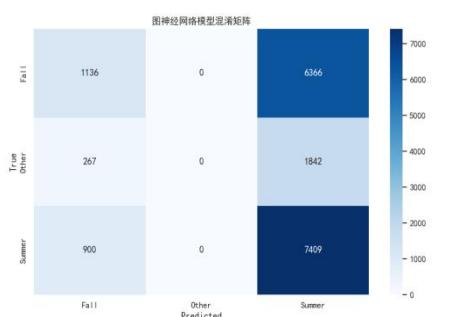


图 5.10 图神经网络随机矩阵

从四张混淆矩阵图可以看出不同模型在台风数据集上的表现差异。首先，随机森林模型在处理多分类问题时，对于“Summer” 的预测较为准确，但对于其他两个则存在较高的误分类率。其次，支持向量机（SVM）模型虽然在某些类别上表现出色，但在处理“Other” 类别时却显示出较大的不确定性，经常将其误分为“Summer”。接着，LSTM 模型在识别“Summer” 类别时几乎无误，但对于“Other” 类别的识别则完全失败，显示出该模型在这个特定类别的弱点。最后，神经网络模型在预测“Summer” 类别时也有较好的表现，但同样在“Fall” 类别上存在不足，显示出模型可能需要在非夏季类别的特征学习上进一步提升。

最后本文对所收集到的 2024 年台风数据进行分类预测，2024 台风数据集在经过和前面 1945-2023 台风数据集一样的数据预处理后，首先将每一条台风的真实行驶路径通过经纬度坐标的不同变化以可视化的形式在地图里面进行显示出来，其过程如图 9 所示：

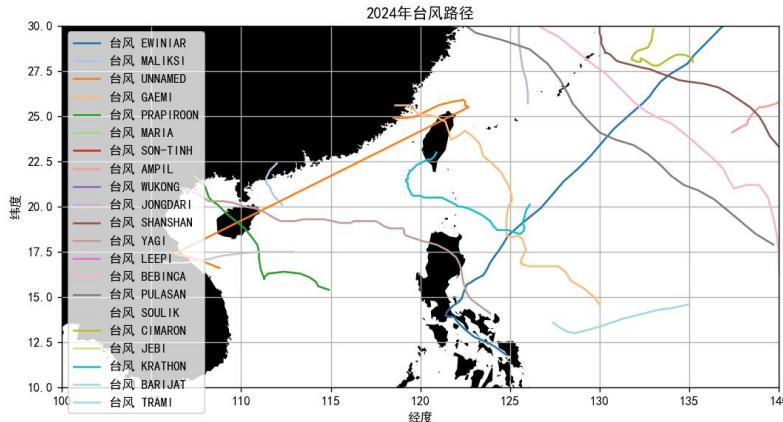


图 5.11 2024 台风行驶路径可视化展示

从图中可以看出大部分台风主要在西北太平洋地区生成，路径多集中于菲律宾以东海域、南海以及东海附近，说明这些区域是台风活动的高频区域。最后本文将数据集通过模型训练并且按照夏季风和秋季风划分得到最终的台风分类表 5-5)，总共分为 3 类分别为 0.1.2 表示 3 种不同的台风。

表 5.5 2024 台风分类表

月份	台风名称	台风编号	台风类别	途径省份
夏台风	GAEMI	2024202N14130	1	福建省
	PRAPIROON	2024202N15116	2	海南省, 广西省
秋季风	YAGI	2024246N14125	1	广东省, 海南省
	BEBINCA	2024254N11147	0	浙江省, 江苏省
	PULASAN	2024261N18141	2	浙江省, 江苏省

## 六、问题二模型建立与求解

### 6.1 数据预处理

本文首先使用和问题一一样的缺失值处理方法，对原始数据进行缺失值处理、时间格式转换、台风强度、移动方向数值化映射、数据编码。其中比如将数据中的时间格式转换为%Y/%M/%D %H:%M 的标准格式，以便计算机能够识别和进一步分析。特别是针对六小时间隔的数据，确保时间序列的连续性和一致性。

对于台风数据中使用汉字或字母进行编码的内容，需要转换为数值形式，以便在模型中应用。例如，台风等级和移动方向这些信息都是汉字形式需要数值化表示。

#### 6.1.1 数据描述性分析

我们针对于风的路径和特征参数进行描述性分析，过滤出当前时间在起始时间和结束时间之间的数据，创建地图，并添加海岸线、边界、陆地和海洋等特征，绘制台风路径，每个台风的路径用不同的颜色表示，路径上的点用相同颜色的圆点表示。设置了地图的范围为全球，以便更好地显示所有台风路径。



图 6.1

其中我们可以发现不同台风的路径形状各异，有的呈直线型，有的呈曲线型，有的呈螺旋型。这表明台风路径受到多种因素的影响，如大气环流、海洋温度、地形等。而且台风路径的方向也各不相同，有的向西北方向移动，有的向东北方向移动，有的向西南方向移动。这表明台风路径的方向受到多种因素的影响。

而为了探究台风强度和风速对经纬度分布的影响，我们通过绘制散点图，展示不同强度台风的经纬度分布。

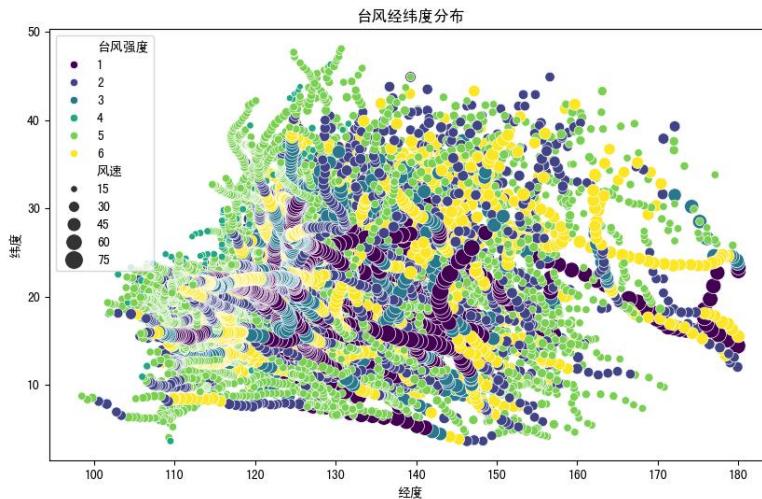


图 6.2

如图 6.2 所示，展示了台风的路径分布。以经度和纬度为坐标，点的颜色和大小分别表示台风强度和风速。

我们可以观察到台风主要集中在热带和亚热带地区，风速较高的台风往往也伴随着较强的强度，且在分布上表现出一定的路径规律性。强度较高的台风主要集中在北纬 0 到 30 度之间，尤其是东经 120 到 150 度之间。强度较低的台风分布较为广泛，但数量较少。

这种分布规律性也为我们的路径预测模型提供了方向，说明路径预测可以利用地理空间特征，同时考虑强度和风速的影响。

而为了探究强度和风速的影响，我们使用箱线图观察台风强度与风速之间的关系。

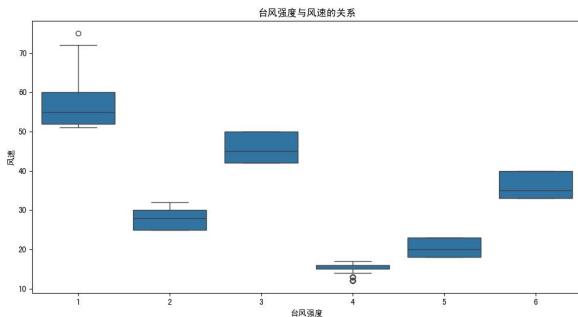


图 6.3

图 6.3 展示了不同强度等级的台风风速分布情况。我们可以看到，台风强度与风速呈现正相关关系，随着台风强度的增加，风速的中位数和四分位数范围也相应增加。这表明台风强度越高，风速越大。

较高强度的台风（例如强度为 1 和 3）往往伴随着较大的风速分布，而强度较低的台风（例如强度为 4 和 5）风速较低且分布较窄。

而且随着台风强度的增加，风速的离散程度也增加。这表明强度较高的台风，其风速变化范围较大。所以我们在预测模型中，考虑将强度作为一个显著特征，结合风速对台风路径和持续时间进行预测。

同时我们分析了每年台风的数量，通过提取年份和月份，统计每年台风数量，分析其变化趋势。

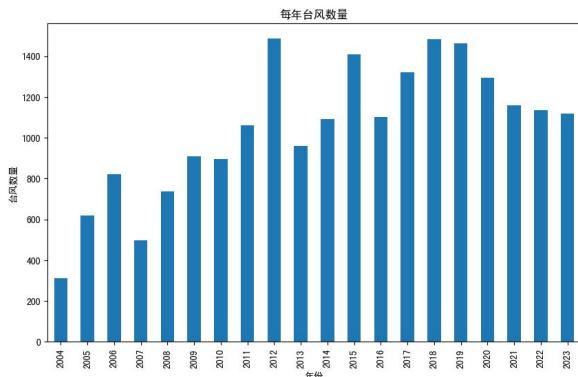


图 6.4 2004 年至 2023 年间每年台风的数量趋势

图 6.4 仅展示 2004 年至 2023 年间每年台风的数量趋势。我们可以看出，台风数量在 2004 年至 2023 年间呈现出波动变化的趋势。从 2004 年到 2018 年间，台风数量呈现逐渐上升的趋势，尤其是在 2012 年和 2018 年达到了高峰，而近几年台风数量有所下降。这一趋势可能反映了年际间气候变化对台风活动的影响，而且台风活动具有一定的周期性和不稳定性。

对于预测模型而言，年份、季节性变化可以作为特征变量，帮助模型更好地适应不同年份或月份的台风活动频率变化。

## 6.1.2 特征工程

针对上述的数据探索，我们可以发现台风数据有着季节性、年份、周期性等特征，他会受到地理空间、台风强度、风速等因素的影响。为了提高台风路径预测的准确性，我们对台风数据进行了详细的特征工程处理，以提取有用的特征，为后续的模型训练提供高质量的数据支持。

(一) 构建时间特征：为了捕捉台风的时间特性，我们计算了台风的已持续时间、预计总持续时间和预计剩余时间。公式如下所示

$$a) \text{ 台风已持续时间}_\text{小时} = \frac{(\text{当前台风时间} - \text{台风起始时间}) * 24 * 60 * 60}{3600}$$

$$b) \text{ 台风预计总持续时间}_\text{小时} = \frac{(\text{台风结束时间} - \text{台风起始时间}) * 24 * 60 * 60}{3600}$$

$$c) \text{ 台风预计剩余时间_小时} = \frac{(\text{台风结束时间}-\text{当前台风时间})*24*60*60}{3600}$$

(二) 构建周期性特征：为了捕捉时间序列的周期性变化，我们对周期性特征（如小时、日、月、季度）进行正弦和余弦变换，以捕捉其周期性变化。对连续特征进行标准化处理，以确保不同特征的尺度一致。公式如下所示

$$a) \text{ Hour}=\text{当前台风时间} \bmod 24$$

$$b) \text{ Day}=\text{当前台风时间}-\text{当前台风时间}\div 100\times 100$$

$$c) \text{ Quarter}=\Gamma \left[ \frac{\text{当前台风时间}\div 100 \bmod 100}{3} \right]$$

$$d) \text{ Hour\_sin} = \sin\left(\frac{2\pi \times \text{Hour}}{24}\right), \text{ Hour\_cos} = \cos\left(\frac{2\pi \times \text{Hour}}{24}\right), \text{ Day\_sin} = \sin\left(\frac{2\pi \times \text{Day}}{31}\right), \text{ Day\_cos} = \cos\left(\frac{2\pi \times \text{Day}}{31}\right), \text{ Month\_sin} = \sin\left(\frac{2\pi \times \text{Month}}{12}\right), \text{ Month\_cos} = \cos\left(\frac{2\pi \times \text{Month}}{12}\right), \text{ Quarter\_sin} = \sin\left(\frac{2\pi \times \text{Quarter}}{4}\right), \text{ Quarter\_cos} = \cos\left(\frac{2\pi \times \text{Quarter}}{4}\right)$$

(三) 构建历史特征：为了捕捉台风的历史特性，我们计算了过去 24 小时的平均风速和平均强度。

$$a) \text{ 24 小时平均风速} = \frac{1}{24} \sum_{i=0}^{23} \text{风速}_{t-i}$$

$$b) \text{ 24 小时平均强度} = \frac{1}{24} \sum_{i=0}^{23} \text{台风强度}_{t-i}$$

(四) 构建距离特征：为了捕捉台风的移动特性，我们计算了经度和纬度的变化量。

$$a) \text{ 经度差} = \text{经度}_t - \text{经度}_{t-1}$$

$$b) \text{ 纬度差} = \text{纬度}_t - \text{纬度}_{t-1}$$

通过上述特征工程处理，我们成功地提取了台风数据中的多种特征，包括时间特征、周期性特征、历史特征和距离特征。这些特征将为后续的模型训练提供丰富的信息，有助于提高台风路径预测的准确性。

## 6.2 基本预测模型

因台风数据具有多种特征，包括时间特征、周期性特征、历史特征、距离特征等特征，本文选择了 LSTM、随机森林、XGBoost、LightGBM 和 CatBoost 这五种模型作为基本模型进行路径预测。

### (一) LSTM 模型路径预测

长短期记忆网络 LSTM 是一种适合时间序列预测的递归神经网络（RNN）模型。由于 LSTM 能够有效捕获序列中的长程依赖性<sup>[10]</sup>，这使其非常擅长处理时间序列数据，其记忆网络神经元结构如下图 4.1 所示。

台风路径预测本质上是一个时间序列预测问题，其数据中有丰富的周期性特性。使用 LSTM 能够捕捉时间序列中的长期依赖关系，这对于预测台风路径的连续变化是非常有效果的。

而且 LSTM 能够处理复杂的非线性关系，这对于台风路径预测中处理多变量和非线性特征是非常有利的，LSTM 能够有效地建模这些复杂的关系。

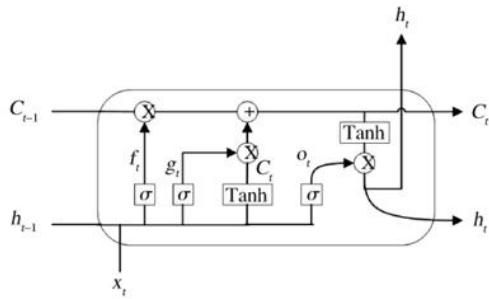


图 6.5 长短期记忆网络 LSTM 模型的结构

在 LSTM 单元中，每一个时刻  $t$  的记忆单元包含以下三个门控结构：

### (1) 遗忘门

遗忘门的作用是决定前一时间步的记忆单元信息（即上一个状态的记忆）在当前时间步要保留多少、遗忘多少。

遗忘门的计算公式如下：

$$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f)$$

其中  $f_t$  表示遗忘门的输出，取值范围在 0 和 1 之间，1 表示完全保留，0 表示完全丢弃； $\sigma$  为 sigmoid 激活函数，将输入映射到 (0,1) 区间； $W_f$  和  $b_f$  分别为遗忘门的权重矩阵和偏置项，是需要学习的参数； $h_{t-1}$  是前一个时间步的隐藏状态； $x_t$  是当前时间步的输入

遗忘门决定了哪些信息可以传递到当前时间步，从而帮助网络“遗忘”那些不再重要的历史信息。

### (2) 输入门

输入门用于控制当前时间步新信息的引入，即决定当前时间步的信息如何更新到记忆单元中。输入门包含两部分计算：输入门激活值  $i_t$  和候选记忆单元内容  $\tilde{C}_t$  的生成。

输入门的公式如下：

#### 1) 输入门激活值：

$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i)$$

#### 2) 候选记忆内容：

$$\tilde{C}_t = \sigma(W_{\tilde{C}} * [h_{t-1}, x_t] + b_{\tilde{C}})$$

其中  $i_t$  是输入门的激活值，用于控制候选记忆内容是否更新到记忆单元中； $\tilde{C}_t$  是候选记忆内容，用于决定当前时间步的新增记忆； $W_i$  和  $b_i$ 、 $W_{\tilde{C}}$  和  $b_{\tilde{C}}$  分别为输入门和候选记忆内容的权重矩阵和偏置项。

最终，输入门的激活值  $i_t$  与候选记忆内容  $\tilde{C}_t$  相乘后，更新到记忆单元  $C_t$  中：

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

$f_t * C_{t-1}$  是经过遗忘门过滤后的前一时间步的记忆， $i_t * \tilde{C}_t$  是经过输入门过滤的当前时间步的新信息。

### (3) 输出门

输出门控制从记忆单元中输出多少信息作为当前时间步的隐藏状态  $h_t$  用于下一时间步或最终的输出。

输出门的计算公式如下：

#### 1) 输出门激活值：

$$o_t = \sigma(W_o * [h_{t-1}, x_t] + b_o)$$

#### 2) 隐藏状态更新：

$$h_t = o_t * \tanh(C_t)$$

其中  $o_t$  是输出门的激活值，用于控制记忆单元中的信息如何输出到隐藏状态； $C_t$  是当前的记忆单元状态，经过  $\tanh$  激活函数处理，将信息映射到 (-1,1) 区间； $W_o$  和  $b_o$  分别为输出门的权重矩阵和偏置项。

LSTM 单元的核心是通过门控机制控制信息的流动，使得模型在序列数据中可以保持和更新重要的历史信息，达到长短期记忆的平衡。

本文使用了一个双向 LSTM 模型用于路径预测，模型输入为时间序列特征，输出为二维位置坐标。模型采用双向 LSTM 结构，通过在时间序列的正向和反向同时学习时序特征，以增强模型的预测性能。网络结构包括两个 LSTM 层以及一个全连接层，用于输出预测的坐标值。

具体预测值与真实值对比图如下图 6.6 所示，每个点表示一个预测值与真实值的对比

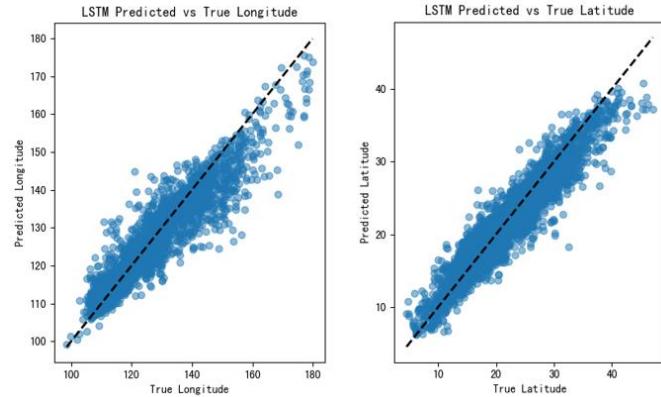


图 6.6 LSTM 预测值与真实值对比图

从图 6.6 我们可以看出散点图呈现出较为明显的线性关系，预测值与真实值之间的斜率接近 1，表明 LSTM 模型在经度预测上具有较高的准确性。而且散点的分布较为集中，表明预测值与真实值之间的误差较小。图中也存在少量异常值，这些点偏离线性关系较远，可能是由于数据噪声或模型在某些特定情况下的预测误差。

## (二) 随机森林模型路径预测

随机森林是一种基于集成学习的机器学习模型，广泛应用于分类和回归任务。它由多个决策树组成，通过综合多个树的预测结果来提高模型的稳定性和预测准确性。

随机森林的构建步骤包括随机抽样、决策树的生成、集成树的预测结果等。具体来说，随机森林通过 Bagging 实现模型集成，即每棵树都是在随机采样的数据子集上训练的，使得不同的树可以学到数据的不同特征，从而降低单棵决策树的偏差和方差。

在路径预测的回归任务中，预测值的计算公式如下：

(1) 单棵树的预测：假设随机森林由  $T$  棵树组成，对于第  $i$  棵树，输入数据  $x$  经过训练得到预测值  $y_i$

(2) 随机森林的预测结果：最终的预测结果是所有决策树输出的平均值，即：

$$\hat{y} = \frac{1}{T} \sum_{i=1}^T y_i$$

其中  $T$  为随机森林中树的数量； $y_i$  是第  $i$  棵树的预测值； $\hat{y}$  是最终的预测值

在路径预测任务中，随机森林能够通过对历史数据的学习来生成合理的预测，避免过拟合问题，我们主要使用决策树生成预测值，而每棵树的输出通过加权平均进行汇总，得到最终的预测结果。

具体预测值与真实值对比图如下图 6.7 所示，每个点表示一个预测值与真实值的对比

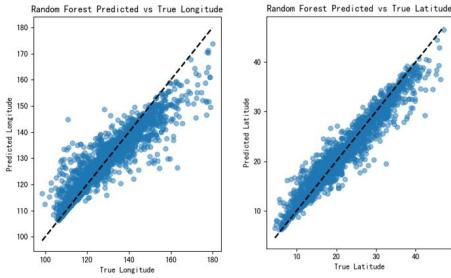


图 6.7 随机森林预测值与真实值对比图

从图中可以看到，大多数点都聚集在虚线附近，说明模型的经度预测效果较好，但左图经度在极值区域（即高值和低值区域）有一些偏差。右图在低纬度区域模型表现稍有偏差，部分点偏离理想线。

### (三) XGBoost 模型路径预测

**XGBoost** 是一种基于梯度提升的机器学习算法，被广泛应用于各种回归和分类任务。它在路径预测中表现优异，主要因为其在提升模型精度的同时具有较高的计算效率。**XGBoost** 在梯度提升决策树（**GBDT**）的基础上进行了改进，引入了正则化、并行计算等技术，使其在精度和速度方面有显著提升。

**XGBoost** 的目标是通过最小化目标函数来找到最优模型。该目标函数可以分为两部分：损失函数和正则化项。具体公式如下：

(1) 目标函数：

$$\mathcal{L}(\theta) = \sum_{i=1}^n l(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k)$$

其中  $\mathcal{L}(\theta)$  是总损失；  $l(y_i, \hat{y}_i)$  是损失函数，通常采用均方误差 (MSE)；  $\Omega(f_k)$  是第  $k$  棵树的正则化项，用于控制模型复杂度，防止过拟合。

(2) 树的结构得分：

为了减少计算复杂度，**XGBoost** 采用二阶泰勒展开来近似损失函数。通过泰勒展开后，损失函数可以表示为：

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^n \left[ g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t)$$

其中  $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$  是损失函数的一阶导数， $h_i = \partial_{\hat{y}^{(t-1)}}^2 l(y_i, \hat{y}_i^{(t-1)})$  是损失函数的二阶导数。

(3) 正则化项：

$$\Omega(f) = \gamma T + \frac{1}{2} \lambda \sum_{i=1}^n w_i^2$$

其中  $T$  是树的叶节点数量；  $w_i$  是叶节点的权重；  $\lambda$  和  $\gamma$  超参数，用于控制正则化的强度。

在路径预测任务中，**XGBoost** 可以利用路径数据的复杂特征，通过回归树的集成来生成精确的预测结果。

具体预测值与真实值对比图如下图 6.8 所示，每个点表示一个预测值与真实值的对比

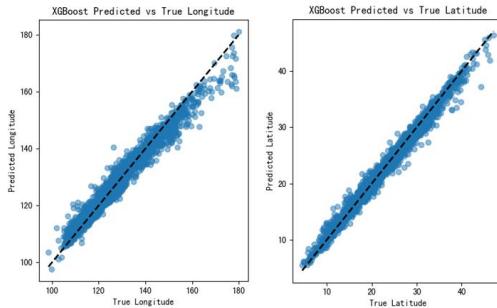


图 6.8 XGBoost 预测值与真实值对比图

从图 6.8 左图可以看出大部分点非常密集地分布在虚线附近，说明 XGBoost 模型在经度预测上表现出高度的准确性。虽然有少数点偏离了虚线，特别是在高经度范围（160-180 左右）和低经度范围（100-120 左右），但总体偏离不大，说明模型在极值区域的预测表现也相对稳定。

右图更加密集，并且几乎完全沿着虚线排列，这表明模型在纬度预测上的精度很高。在低纬度和高纬度值附近，几乎没有明显偏离的点，说明模型在整个纬度范围内的预测一致性和准确性都较强。

#### (四) LightGBM 模型路径预测

LightGBM 是一个基于决策树的分布式梯度提升框架，以其高效的计算和低内存占用而闻名。其快速的训练速度和准确性使其在路径预测中非常适合。

与传统的 GBDT 不同，LightGBM 使用基于直方图的决策树算法，提升了处理大规模数据的速度。

在路径预测任务中，我们将 LightGBM 分别对经度和纬度进行回归预测，然后将预测的经度和纬度组合在一起，形成路径的二维坐标预测。

具体预测值与真实值对比图如下图 6.9 所示，每个点表示一个预测值与真实值的对比。

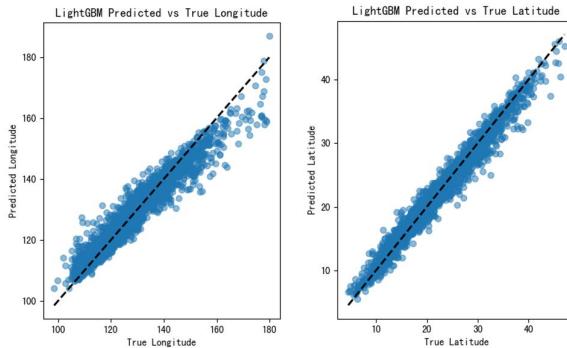


图 6.9 LightGBM 预测值与真实值对比图

上图 4.5 中数据点大致分布在虚线附近，表明模型的预测值，整体趋势较为准确。但是，尽管大多数点接近虚线，在经度图中部分点偏离理想线较远，尤其是在较高和较低的经度值区域。这说明模型在这些范围的预测误差可能更大。在纬度图中，偏离理想线的点相对较少，预测效果略好于经度。

#### (五) CatBoost 模型路径预测

CatBoost 是一种基于梯度提升决策树的机器学习模型，具有在处理分类特征和高维数据方面的优势。在路径预测任务中，CatBoost 能够通过对历史数据的学习，生成对未来路径的精确预测。此外，CatBoost 支持 GPU 加速，这使得模型训练速度较快非常适合台风路径的数据集。

在路径预测任务中，我们为了获得准确的预测结果，分别对经度和纬度进行建模，然后将预测结果合并为路径的二维坐标，通过对经度和纬度分别进行回归训练，提升模型的精度。

具体预测值与真实值对比图如下图 6.10 所示，每个点表示一个预测值与真实值的对比。

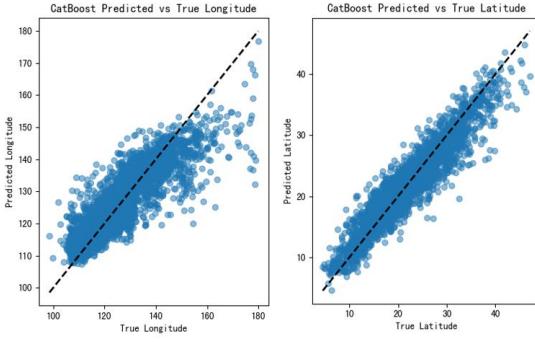


图 6.10 CatBoost 预测值与真实值对比图

上图 6.10 左图我们看到数据点分布较为分散，尤其是在高经度区域（接近 180）和低经度区域（接近 100）出现了较大的偏离。这表明在这些区域，模型的预测存在较大误差，可能是由于训练数据在这些区间不足或模型未能很好捕捉数据的非线性特征。

而右图整体上，数据点在对角线上下分布较为密集，且偏离程度比经度图低，说明模型在预测纬度时相对表现更好。

### 6.3 集合模型路径预测

为了进一步提高预测精度，我们构建了一个集成模型。我们根据每个模型的 **MSE** 计算了权重，并将这些权重应用于各个模型的预测结果，最终得到一个综合预测结果。

其中权重的计算我们使用了两种不同的方法来计算模型的权重，分别是基于误差总和的权重计算和基于归一化误差的权重计算

#### (1) 基于误差总和的权重计算

假设有  $n$  个模型，每个模型的误差分别为  $e_1, e_2 \dots e_n$ ，首先计算所有误差的总和  $E$ ， $E$  的公式如下

$$E = \sum_{i=1}^n e_i$$

然后每个模型的权重  $w_i$  计算如下：

$$w_i = \frac{e_i}{E}$$

#### (2) 基于归一化误差的权重计算

首先将每个模型的误差归一化到  $[0, 1]$  区间。假设最小误差为  $e_{\min}$ ，最大误差为  $e_{\max}$ ，则归一化误差  $ne_i$  计算如下：

$$ne_i = \frac{e_i - e_{\min}}{e_{\max} - e_{\min}}$$

归一化误差越小，权重越大。因此，权重  $w_i$  计算如下：

$$w_i = 1 - ne_i$$

并且为了确保所有权重之和为 1，对权重进行归一化：

$$w_i = \frac{w_i}{\sum_{j=1}^n w_j}$$

其中，我们评估模型性能的指标使用 **MSE** 和 **MAE** 来评估，具体如下

#### (1) 均方误差 (**MSE**)

均方误差 (**MSE**) 是预测值与实际值之间差异的平方的平均值。假设  $n$  个样本，预测值为  $\hat{y}_i$ ，实际值为  $y_i$ ，则 **MSE** 的公式为：

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

## (2) 平均绝对误差 (MAE)

平均绝对误差 (MAE) 是预测值与实际值之间差异的绝对值的平均值。假设有  $n$  个样本，预测值为  $\hat{y}_i$ ，实际值为  $y_i$ ，则 MAE 的公式为：

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

在得到了所有的预测结果后，将每个模型的预测结果乘以其对应的权重，然后加权平均得到集合预测。

## 6.4 预测结果分析

### 6.4.1 台风贝碧嘉的数据预处理

我们从气象网上爬取了 2024 年 9 月 13 日-17 日的第 13 号台风贝碧嘉的数据，下面表 6.1 为前五个示例数据。

表 6.1 第 13 号台风贝碧嘉示例数据

时间	经度	纬度	强度	风力	风速	移动方向	移动速度	中心气压
2024-09-10 20:00:00	145.4	12.4	5	8.0	18.0	315.0	26.0	998.0
2024-09-10 23:00:00	144.9	12.9	5	8.0	18.0	315.0	24.0	998.0
2024-09-11 02:00:00	144.5	13.3	5	8.0	18.0	315.0	22.0	998.0
2024-09-11 05:00:00	143.9	13.4	5	9.0	23.0	315.0	20.0	990.0

并且对其做缺失值探索，发现如图 6.11 中，移动方向和移动速度有着缺失值。

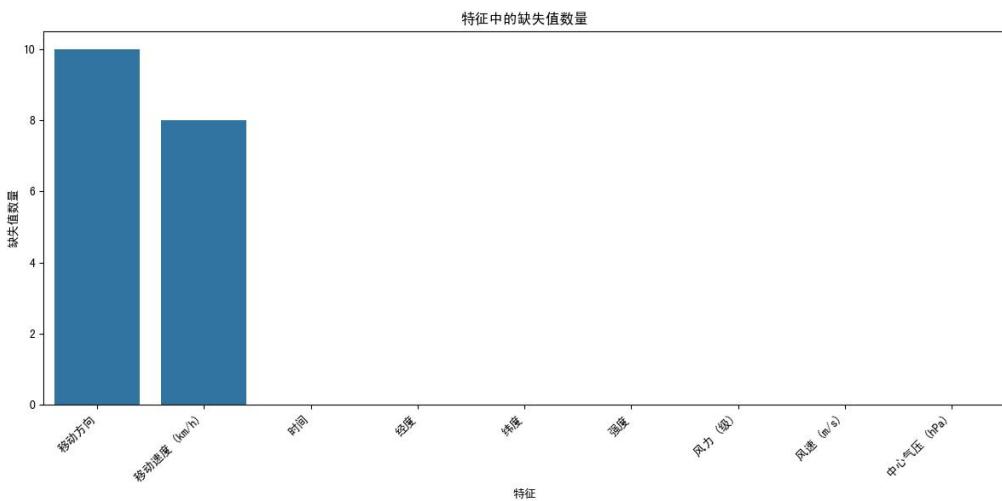


图 6.11

对于这部分的缺失值，我们选择了用 0 作为补充其缺失值，然后对其上面的做特征工程。

### 6.4.2 台风贝碧嘉数据预测和分析

将处理好后的数据作为测试集给其他已经训练好的模型进行预测。下面的表 6.2 就是各模型的预测结果情况

表 6.2 各模型预测结果情况

模型	MSE	MAE
LSTM	8.654053648254456	2.4493602017795357

随机森林	12.872746462941178	2.7758235294117615
XGBoost	12.400897943149298	2.850684728061452
LightGBM	10.033035837641723	2.522505592118157
CatBoost	10.233920300312972	2.508749063833762
集合模型	7.763692052569926	2.207663881978546

可以看得出，随机森林和 XGBoost 的效果不如其他模型，而且这两个模型的 MSE 和 MAE 都较高，说明它们在当前任务中的表现不如 LSTM 和集合模型。而 LightGBM 和 CatBoost 的性能介于 LSTM 和随机森林和 XGBoost 之间，表现中等。

LSTM 模型在 MSE 指标上表现的非常好，这说明 LSTM 在预测任务中具有较高的准确性。但是集合模型要明显的比其他模型效果好，它的低 MAE 表明它在减少预测误差的绝对值方面表现出色，可能是因为它结合了多个模型的优点，减少了单个模型的偏差。

我们同时也计算了不同模型的预测误差分布情况，如图 6.12 所示

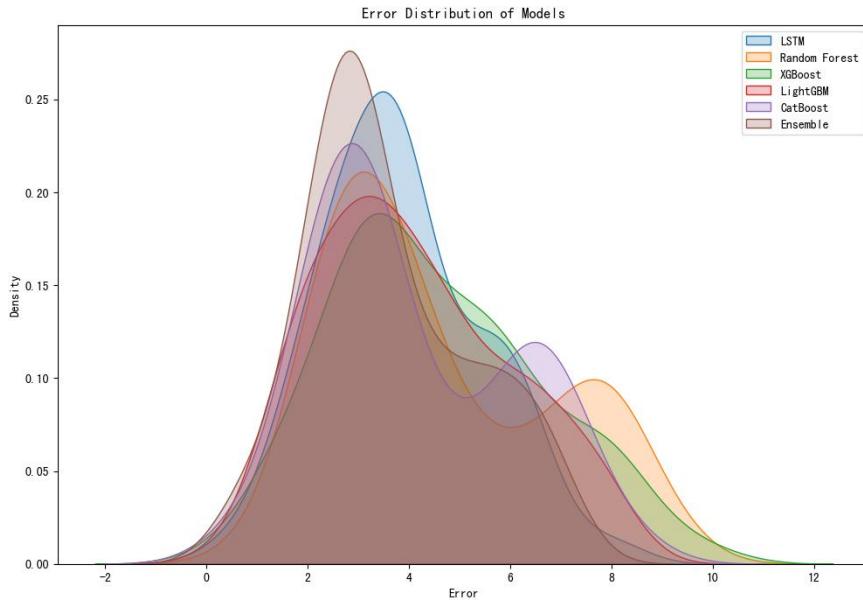


图 6.12 不同模型的预测误差分布情况

其中我们可以看到 Ensemble 模型，也就是集合模型的误差分布曲线的峰值明显集中在较低的误差区间，且分布范围较窄，说明其误差较小，表现较为稳定。这表明 Ensemble 模型在所有模型中整体表现最好，误差最低。

而 LSTM 的误差分布有一个明显的峰值，但分布较宽，特别是在高误差区域，曲线有较长的尾部。这表明 LSTM 在大多数情况下有较好的预测效果，但偶尔也会产生较大误差。

随机森林的误差曲线比 LSTM 更加分散，说明它的预测不够精确，并且高误差情况更多。XGBoost 的误差分布在图中偏右，且分布范围较广，误差较大。这表明 XGBoost 在此任务上的表现相对较弱。LightGBM 和 CatBoost 模型的误差分布相对接近，但分布密度较低。相比于 Ensemble，它们的误差分布范围较宽，说明其在此任务中的稳定性和精度稍逊于 Ensemble 模型。

我们也将预测值和真实值构建在世界地图上，结果如下图 6.13 所示

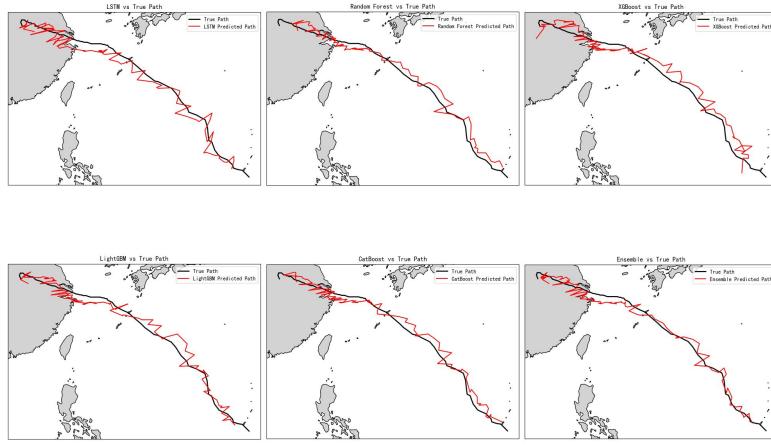


图 6.13

我们通过图 6.13 可以发现，其中集合模型的效果是最贴合真实路径的，其他的模型都存在着一定的误差

最后根据题目，我们提取出了 2024 年 9 月 13 日-17 日每日 14 点的第 13 号台风贝碧 嘉的数据，如下表 6.3 所示

表 6.3

时间	经度	纬度	集合模型预测的经度	集合模型预测的维度
2024-09-13 14:00:00	135.6	23.6	136.491798	24.034648
2024-09-14 14:00:00	130.3	27.3	131.208968	26.959543
2024-09-15 14:00:00	126.2	30.1	126.450671	29.375684
2024-09-16 14:00:00	120.5	31.5	121.426795	30.631901
2024-09-17 14:00:00	116.3	33.0	117.826862	32.571588

我们可以发现，预测的经度已经相当精确了，经过提取后，将经度/维度得到台风中心位置，得到下表 6.4.

表 6.4

时间	台风中心位置(经度/维度)
2024-09-13 14:00:00	136.49/24.03
2024-09-14 14:00:00	131.21/26.96
2024-09-15 14:00:00	126.45/29.38
2024-09-16 14:00:00	121.43/30.63
2024-09-17 14:00:00	117.83/32.57

## 6.5 运用动态时间规整算法与实际路线做对比

根据题目，我们使用 Dynamic Time Warping (DTW) 动态时间规整算法与台风实际行进路线进行对比，用于评估不同模型的预测路径与真实路径之间的相似度。DTW 是一种用于测量两个时间序列之间相似度的算法，即使它们在时间轴上不完全对齐。

下表 6.5 是不同模型和 DTW 的距离，其中 Ensemble 是集合模型.

表 6.5 不同模型和 DTW 的距离

模型	DTW 距离
LSTM	51.654983921050956
Random Forest	45.063549999999993
XGBoost	50.903199195861774

LightGBM	43.327199295045396
CatBoost	48.01918876035756
Ensemble	38.615711776985215

其中 DTW 距离越小，模型的预测路径与真实路径越接近。Ensemble 模型的 DTW 距离最小，表明其在大多数情况下预测误差最小，是最佳的选择。

## 七、问题三模型的建立与求解

### 7.1 数据预处理

在数据处理阶段，我们进行了以下步骤以确保数据的准确性和一致性。首先对时间转换，将台风路径数据中的时间字段从字符串格式转换为 `datetime` 格式，以便进行时间匹配。这一步骤确保了台风路径数据与降水场数据在时间上的对齐，从而能够准确地分析特定时刻的降水量与台风中心距离之间的关系。然后，使用欧几里得距离公式计算每个网格点与台风中心的距离，通过计算每个网格点与台风中心的距离，我们可以分析不同距离下的降水量变化。最后，我们对数据进行标准化，对输入数据进行标准化处理，以提高拟合的稳定性和准确性。

数据预处理包含的数学模型：

#### 1. 计算距离公式

使用欧几里得距离计算每个网格点与台风中心的距离。

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

#### 2. 标准化

对输入数据进行标准化处理，以提高拟合的稳定性和准确性。标准化后的数  $X_{norm}$  具有零均值和单位方差，有助于提高拟合的稳定性。

$$X_{norm} = \frac{X - \mu_X}{\sigma_X}$$

原始数据（如 D, V 等）经过标准化处理，其中 X 的均值被减去，然后除 X 的标准差。

#### 3. 误差函数

定义了残差函数，用于衡量模型预测值与观测值之间的差异。

$$\text{residuals} = P_{fit} - P_{obs}$$

$P_{fit}$ 是拟合模型预测的降水量， $P_{obs}$ 是观测到的降水量，误差函数衡量了 P1 与 P2 之间的差异。

#### 4. 最小二乘拟合

通过最小化残差平方和来找到最优参数。

$$\min_{A, \alpha, \beta} \sum (P_{fit} - P_{obs})^2$$

目标是找到参数 A,  $\alpha$ ,  $\beta$ ，使得模型预测值  $P_{fit}$  与观测值  $P_{obs}$  之间的平方误差之和最小。

### 7.2 建立模型

问题三要求建立台风在登陆后风速和降水量之间的关系，以及降水量与距台风中心距离之间的关系。

## 7.2.1 风速与距离的关系模型

台风登陆后，风速通常呈现出逐渐衰减的趋势。我们可以使用指数衰减模型来描述风速随距离的变化：

$$V(r) = V_0 e^{-\alpha r}$$

风速  $V(r)$  表示距台风中心为  $r$  的位置的风速，初始风速  $V_0$  是台风登陆时的风速，衰减系数  $\alpha$  表示风速随距离  $r$  增加而减弱的速度。

模型拟合：使用历史数据对风速与距离的关系进行拟合，估计衰减系数  $\alpha$  可以使用最小二乘法来最小化预测值与实际观测值之间的误差。

## 7.2.2 降水量与距离的关系模型

降水量通常也会随着距台风中心的距离增加而减小，但受其他因素（如地形、湿度等）影响较大。可以采用多元回归模型来描述：

$$P(r) = \beta_0 + \beta_1 r + \beta_2 V(r) + \beta_3 H + \epsilon$$

$P(r)$  表示距台风中心为  $r$  的位置的降水量，由回归系数  $\beta_0, \beta_1, \beta_2, \beta_3$ 、风速  $V_0$  和湿度  $H$  决定，并包含随机误差项  $\epsilon$ 。

经过计算了每个时间点上台风中心附近（距离台风中心最近的网格点）的降水量后，距离与降水量关系结果如表 7.1 所示：

表 7.1 距离与降水量关系数据

Time	Distance	Precipitation
2022-09-16 04:00:00	0.070710678	75.88999939
2022-09-16 05:00:00	0.070710678	63.27999878
2022-09-16 06:00:00	0.070710678	69.12000275
2022-09-16 07:00:00	0.070710678	65.69000244
2022-09-16 14:00:00	0.070710678	0.25999999
2022-09-16 15:00:00	0.158113883	0.720000029
2022-09-16 16:00:00	0.552268051	1.75999999
2022-09-16 17:00:00	0.95131488	1.419999957

通过最小二乘法拟合降水量模型后，生成了一个包含观测降水量和拟合降水量的数据集，如表 7.2 所示：

表 7.2 观测降水量和拟合降水量数据

Distance	Observed_Precipitation	Fitted_Precipitation
0.070710678	0.26	5.49E-12
0.158113883	0.72	8.75E-12
0.552268051	1.76	4.95E-12
0.95131488	1.42	3.28E-12

## 7.3 可视化分析

在本研究中，我们生成了多个图表来展示台风路径、降水量分布以及模型拟合结果。以下是对每个图表的详细描述：

### （一）台风路径与风速图

如图 7.1 展示了 2022 年某区域内的台风路径，并用颜色表示不同位置的风速。使用 OpenStreetMap 作为底图，提供了地理参考。每个点代表台风路径上的一个位置，点的大小和颜色反映了该位置的最大风速 (m/s)。通过这张图，可以直观地看到台风路径及其风速的变化情况。

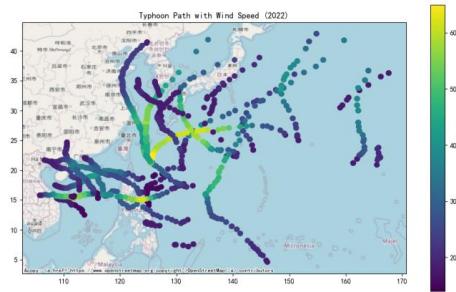


图 7.1 2022 年台风路径及风速

## (二) 台风路径与降水量图

如图 7.2 展示了 2022 年某区域内的台风路径，并用颜色表示不同位置的降水量。使用 OpenStreetMap 作为底图，提供了地理参考。每个点代表台风路径上的一个位置，点的大小和颜色反映了该位置的降水量 (mm)。通过这张图，可以直观地看到台风路径及其降水量的变化情况。

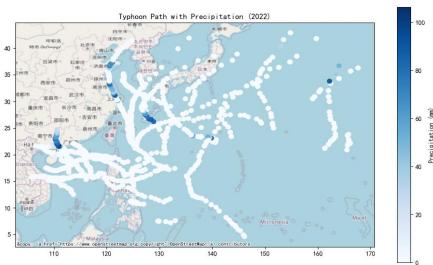


图 7.2 2022 年台风路径及降水量

## (三) 某一天的降水量分布图

如图 7.3 展示了 2022 年 1 月 1 日某区域内的降水量分布情况。使用 imshow 函数绘制了降水量数据，颜色越深表示降水量越大。色彩条 (colorbar) 显示了降水量的数值范围 (mm)。通过这张图，可以直观地看到特定日期内整个区域的降水量分布情况。

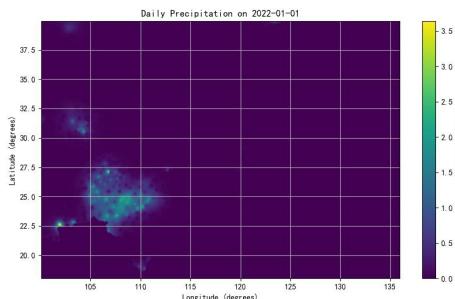


图 7.3 2022 年 1 月 1 日的日降水量

## (四) 台风路径及降水量分布图

如图 7.4 综合展示了 2022 年某区域内的台风路径、降水量分布以及风速信息。使用 OpenStreetMap 作为底图，提供了地理参考。通过 contourf 函数绘制了降水量分布，颜色越深表示降水量越大。台风路径上的点用颜色表示风速 (m/s)，点的大小和颜色反映了风速的大小。通过这张图，可以同时观察到台风路径、降水量分布以及风速的变化情况，有助于更全面地理解台风的影响。

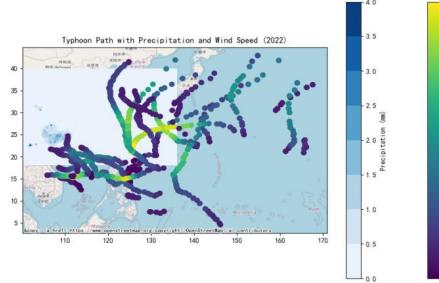


图 7.4 2022 年台风路径及降水量和风速

### (五) 观测与拟合降水量对比图

如图 7.5 展示了观测到的降水量与模型拟合的降水量之间的对比。横轴表示距离台风中心的距离 (km)，纵轴表示降水量 (mm)。蓝色散点表示观测到的降水量数据，红色曲线表示模型拟合的降水量。通过这张图，可以直观地看到模型对观测数据的拟合效果，评估模型的预测能力。如果拟合曲线与观测数据点接近，说明模型具有较好的预测性能。

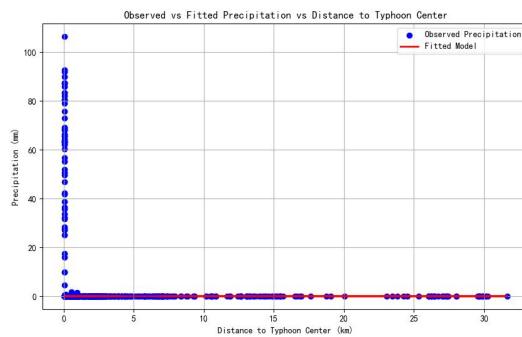


图 7.5 观测与拟合降水量随距台风中心距离的变化。

通过上述图表，我们可以全面地了解 2022 年某区域内的台风路径、降水量分布以及风速变化情况。这些图表不仅帮助我们可视化数据，还为模型拟合和预测提供了有力的支持。具体来说，台风路径与风速图 和 台风路径与降水量图 提供了台风路径及其相关气象参数的直观展示。某一天的降水量分布图展示了特定日期内的降水量分布情况。台风路径及降水量分布图综合展示了台风路径、降水量分布以及风速信息。观测与拟合降水量对比图评估了模型对观测数据的拟合效果，验证了模型的有效性。

## 7.4 建立综合的预测模型

### 7.4.1 风速与台风中心路径

Holland 公式是一种经典的方法，用于描述台风的风场分布，尤其是在台风 中心周围风速与距离的关系上。这个公式通常用来描述台风在登陆后的风速分布。Holland 公式考虑了台风中心气压差、台风半径和台风强度等因素。以下是 Holland 公式的数学表示：

(一) 计算最大风速半径  $R_m$  和 Holland 参数 B:

1. 最大风速半径  $R_m$  的对数:  $\ln(R_m) = 3.015 - 6.291 \times 10^{-5} \times (\Delta P)^2 + 0.337 \times \text{纬度}$
2.  $R_m$  的计算:  $R_m = \exp(\ln(R_m))$
3. Holland 参数 B 的计算:  $B = 1.4 + 0.01 \times \Delta P - 0.0026 \times R_m$  ( $\Delta P$  是气压差  $1010 - P_c$  (单位: hPa)

(二) 使用 Holland 公式计算中心风速  $V_{max}$

Holland 风速公式:

$$V_{max} = \sqrt{\frac{B}{\rho} \times (P_0 - P_c) \times 100 \times e}$$

Holland 参数 B 与空气密度  $\rho = 1.15 \text{ kg/m}^3$ 、标准大气压  $P_0$  和台风中心的气压  $P_c$ （单位：hPa）相关，并通过自然对数的底数 e 进行计算。

#### 7.4.2 使用拟合的降水量公式计算降水量

降水量公式：

$$P_{obs} = A \cdot \exp(-\alpha \cdot D) \cdot \left(\frac{1}{V}\right) \cdot \left(\frac{1}{P_c - P_{red}}\right) \cdot (V_{max}^\beta)$$

拟合参数 A、 $\alpha$  和  $\beta$  用于计算在距离台风中心 D 处的降水量，考虑了风速 V、台风中心气压  $P_c$ 、参考气压  $P_{red} = 1013 \text{ (hPa)}$  和最大风速  $V_{max}$  的影响。

经过上述处理后，计算风速和预测降水量如表 7.3 所示

表 7.3 计算风速和预测降水量数据

台风英文名称	当前台风时间	纬度	经度	移动速度	风速	气压	计算风速	预测降水量
BEBINCA	2024091600	30.9	121.6	16	64	970	130.0574599	-3.15E-05
BEBINCA	2024091603	30.6	123	25	42	955	158.2678249	-1.38E-05
BEBINCA	2024091606	30.7	122.3	25	45	955	158.2498542	-1.47E-05
BEBINCA	2024091609	31	121.6	22	40	965	139.5875202	-1.74E-05
BEBINCA	2024091612	31.3	120.9	20	38	975	120.2202741	-2.35E-05
BEBINCA	2024091615	31.5	120.2	18	30	978	114.1966823	-2.15E-05
BEBINCA	2024091618	31.6	119.8	12	28	980	110.1230799	-2.20E-05
BEBINCA	2024091621	31.8	119.1	21	28	980	110.0882729	-2.19E-05
BEBINCA	2024091700	32	118.5	19	25	985	99.64272426	-2.54E-05
BEBINCA	2024091703	32.2	118	16	20	990	88.6942299	-2.80E-05
BEBINCA	2024091706	32.5	117.3	15	18	998	69.51530852	-4.89E-05
BEBINCA	2024091709	32.6	116.9	13	18	998	69.50178532	-4.89E-05
BEBINCA	2024091712	33	116.5	13	18	998	69.44720773	-4.85E-05
BEBINCA	2024091715	33	116.2	14	18	998	69.44720773	-4.85E-05
BEBINCA	2024091717	33.1	116.1	11	15	1000	64.15123066	-5.11E-05
BEBINCA	2024091723	33.3	115.4	8	15	1000	64.12519318	-5.09E-05

#### 7.4.3 可视化分析

通过模型生成了三个主要的可视化图表，分别是台风中心风速的对比图、预测降水量的时间序列图以及台风路径的地图。以下是对这些效果图的分析：

##### (一) 台风中心风速的对比图：

该图展示了观测到的台风中心最大风速（红色实线）与通过 Holland 公式计算得到的最大风速（蓝色虚线）随时间的变化以及预测降水量的时间序列图：显示了根据先前拟合的降水模型预测的降水量随时间的变化。图表可以帮助理解台风期间预计的降水模式和强度变化。如图 7.6 所示：

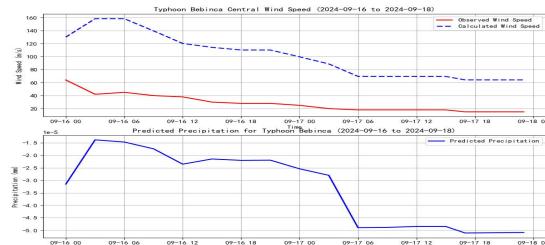


图 7.6 贝碧嘉台风中心风速和降水量 (2024.9.16-2024.9.18)

如果两条曲线趋势相似，说明 Holland 公式的参数设置合理，能够较好地反映实际情况。如果两者存在显著差异，可能需要调整 Holland 公式的参数或考虑其他影响因素（如地形、海温等）对风速的影响。观察是否有特定时间段两者的偏差较大，这可能指示出模型在某些条件下表现不佳。降水量峰值出现的时间点是否与实际观测或其他气象数据相符。注意任何异常高的降水量预测值，这可能是由于输入数据错误或模型参数设置不合理导致的。

## （二）台风路径的地图

如图 7.7 描述：地图上标记了台风路径上的各个位置，并叠加了 OpenStreetMap 作为背景图层。

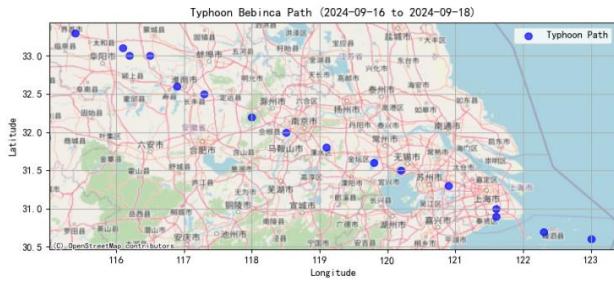


图 7.7 贝碧嘉台风路径（2024. 9. 16–2024. 9. 18）

地图直观地展示了台风的移动轨迹，有助于识别台风的主要活动区域及其潜在影响范围。结合地理信息（例如城市、人口密集区），可以评估台风对不同地区的潜在威胁程度。对比历史数据，了解台风路径是否存在特定模式或偏好。

## 八、模型总结

### 8.1 模型优点

- 利用历史数据进行建模：通过对历史台风数据的分析，模型提取了降水量与台风特征之间的联系。历史数据为模型参数提供了支持，使其更好地适应实际观测的复杂性，从而增强了泛化能力。
- 综合多个影响因素：除了风速和中心气压，模型还考虑了与台风降水密切相关的多种因素，例如距离台风中心的距离、最大风速和移动速度等。这使得模型能够更详细地描绘降水量的变化，全面反映台风过程中的多维影响。
- 优秀的可视化效果：模型的可视化部分包括风速与降水量的时间变化图和台风路径的地理展示，使预测结果更为直观。这些可视化工具有助于从空间和时间两个角度理解台风的影响，为后续的灾害应对提供重要的信息支持。

### 8.2 模型的缺点

- 模型对初始参数的敏感性：在降水量拟合过程中，模型对初始参数的设置表现出较强的敏感性。如果初始参数选择不当，可能会导致模型无法收敛或陷入局部最优解。为了获得合理的拟合结果，可能需要多次调整初始参数，这在一定程度上降低了模型的自动化程度和实用性。
- 未能完全捕捉台风路径的复杂性：台风路径预测是一个高度非线性的问题，受到多种因素的影响，包括副热带高压和其他台风的相互作用。尽管采用了 FPCA 和 DTW 等方法进行台风路径的预测与匹配，实际路径的复杂性仍超出了模型的表达能力。特别是在台风路径发生急剧转向或多台风相互影响时，模型可能难以准确预测路径。

## 九、参考文献

- [1] Kossin, J. P., et al. (2014). "A globally consistent reanalysis of hurricane track and intensity." *Journal of Climate*, 27(6), 2485-2496.
- [2] Chavas, D. R., & Emanuel, K. A. (2010). "A method for estimating the intensity of tropical cyclones from their wind field." *Geophysical Research Letters*, 37(19).
- [3] Yuan, Y., et al. (2020). "A deep learning approach for tropical cyclone track prediction." *Nature Communications*, 11, 5657.
- [4] 李华 (2021). "基于随机森林的台风分类研究." 气象科技, 49(1), 113-120.
- [5] Spearman, C. (1904). The proof and measurement of association between two things. *The American Journal of Psychology*, 15(1), 72-101.
- [6] Breiman, L. (2001). "Random Forests." *Machine Learning*, 45(1), 5-32. DOI: 10.1023/A:1010933404324.
- [7] Cortes, C., & Vapnik, V. (1995). "Support-Vector Networks." *Machine Learning*, 20(3), 273-297. DOI: 10.1023/A:1022627411411.
- [8] Hochreiter, S., & Schmidhuber, J. (1997). "Long Short-Term Memory." *Neural Computation*, 9(8), 1735-1780. DOI: 10.1162/neco.1997.9.8.1735.
- [9] Kipf, T. N., & Welling, M. (2017). "Semi-Supervised Classification with Graph Convolutional Networks." *Proceedings of the 5th International Conference on Learning Representations (ICLR)*. DOI: 10.48550/arXiv.1609.02907.
- [10] 任嘉鹏. 基于机器学习的流量预测及基站休眠方法研究[D]. 吉林大学, 2020

## 十、附录

本次论文所做实验全部都基于 Pycharm, VScode 和 Jupyter 程序行编写。

### 10.1 问题一代码

#### 1. 数据预处理部分代码

---

代码 1	数据预处理部分代码
------	-----------

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
import numpy as np
# 读取数据文件
file_path = '1945-2023.csv'
typhoon_data = pd.read_csv(file_path, encoding='UTF-8')
# 修正列名
column_names = [
    '台风编号', '台风中文名称', '台风英文名称', '台风起始时间', '台风结束时间',
    '当前台风时间', '经度', '纬度', '台风强度', '台风等级', '风速', '气压', '移动方向',
    '移动速度'
]
typhoon_data.columns = column_names
# 转换时间相关列的格式
typhoon_data['台风起始时间'] = pd.to_datetime(typhoon_data['台风起始时间'],
                                             errors='coerce')
typhoon_data['台风结束时间'] = pd.to_datetime(typhoon_data['台风结束时间'],
                                             errors='coerce')
typhoon_data['当前台风时间'] = pd.to_datetime(typhoon_data['当前台风时间'],
                                             errors='coerce')
# 提取年、月等时间特征
typhoon_data['年'] = typhoon_data['台风起始时间'].dt.year
typhoon_data['月'] = typhoon_data['台风起始时间'].dt.month
# 删除气压、台风等级和台风强度存在空值的行
typhoon_data.dropna(subset=['气压', '台风等级', '台风强度'], inplace=True)
# 计算每条台风的缺失移动方向和移动速度
def calculate_missing_speed_and_direction(df):
    # 确保数据按时间排序
    df.sort_values(by=['台风编号', '当前台风时间'], inplace=True)
    # 按台风编号分组处理
    for typhoon_id, group in df.groupby('台风编号'):
        # 计算经纬度差异
        group['经度差'] = group['经度'].diff() * np.cos(np.radians(group['纬度']))
        group['纬度差'] = group['纬度'].diff()
        # 计算时间差（以小时为单位）
        group['时间差'] = group['当前台风时间'].diff().dt.total_seconds() / 3600.0
```

---

---

```
# 计算移动方向（对所有数据计算）
group['移动方向'] = np.degrees(np.arctan2(
    group['纬度差'],
    group['经度差']
)) % 360
# 计算移动距离（使用 haversine 公式）
R = 6371 # 地球半径，单位为公里
group['移动距离'] = R * np.arccos(
    np.sin(np.radians(group['纬度'])) * np.sin(np.radians(group['纬度'].shift(1))) +
    np.cos(np.radians(group['纬度'])) * np.cos(np.radians(group['纬度'].shift(1))) *
    np.cos(np.radians(group['经度'])) - np.radians(group['经度'].shift(1)))
)
# 计算移动速度（仅对缺失值计算）
missing_speed_mask = group['移动速度'].isna() & (group['时间差'] > 0) # 确保时间差大于 0
group.loc[missing_speed_mask, '移动速度'] = group.loc[missing_speed_mask,
'移动距离'] / group.loc[
    missing_speed_mask, '时间差']
# 将计算结果更新回原 DataFrame
df.loc[group.index, '移动方向'] = group['移动方向']
df.loc[group.index, '移动速度'] = group['移动速度']
df.dropna(subset=['移动方向', '移动速度'], inplace=True)
# 清理临时列
df.drop(columns=['经度差', '纬度差', '时间差', '移动距离'], inplace=True, errors='ignore')
return df
# 应用计算函数
typhoon_data = calculate_missing_speed_and_direction(typhoon_data)
# 台风强度数值化映射
intensity_mapping = {
    '超强台风(Super TY)': 1,
    '强热带风暴(STS)': 2,
    '强台风(STY)': 3,
    '热带低压(TD)': 4,
    '热带风暴(TS)': 5,
    '台风(TY)': 6
}
# 应用映射
typhoon_data['台风强度'] = typhoon_data['台风强度'].map(intensity_mapping)
# 保存处理后的数据为新的 CSV 文件
output_file_path = 'processed_typhoon_data.csv' # 请将此路径替换为您的存储路径
typhoon_data.to_csv(output_file_path, index=False)
```

---

---

```
print(f"缺失值已删除, 处理后的数据已保存到 {output_file_path}")
```

---

## 2. 数据可视化代码

---

代码 2	数据可视化部分代码
------	-----------

---

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pylab import mpl
import numpy as np
import geopandas as gpd
mpl.rcParams['font.sans-serif'] = ['SimHei']
# 读取预处理后的数据文件
file_path = 'processed_typhoon_data.csv'
typhoon_data = pd.read_csv(file_path)

# 1. 总体统计信息
print("数据集描述性统计信息: ")
print(typhoon_data.describe())
```

### # 2. 特征分布分析

```
# 风速分布
plt.figure(figsize=(10, 6))
sns.histplot(typhoon_data['风速'], bins=30, kde=True, color='skyblue')
plt.title("风速分布")
plt.xlabel("风速 (标准化)")
plt.ylabel("频数")
plt.show()
```

```
# 气压分布
plt.figure(figsize=(10, 6))
sns.histplot(typhoon_data['气压'], bins=30, kde=True, color='salmon')
plt.title("气压分布")
plt.xlabel("气压 (标准化)")
plt.ylabel("频数")
plt.show()
```

```
# 等级分布
plt.figure(figsize=(10, 6))
sns.histplot(typhoon_data['台风等级'], bins=30, kde=True, color='lightgreen')
plt.title("等级分布")
```

---

---

```
plt.xlabel("等级 (标准化)")
plt.ylabel("频数")
plt.show()

# 3. 时间特征分析

# 台风月份分布
plt.figure(figsize=(12, 6))
sns.countplot(x='月', data=typhoon_data, palette='coolwarm')
plt.title("台风月份分布")
plt.xlabel("月份")
plt.ylabel("台风数量")
plt.show()

# 台风年份分布
plt.figure(figsize=(12, 6))
sns.countplot(x='年', data=typhoon_data, palette='viridis')
plt.xticks(rotation=90)
plt.title("台风年份分布")
plt.xlabel("年份")
plt.ylabel("台风数量")
plt.show()

# 4. 空间特征分析

# 经纬度分布
plt.figure(figsize=(10, 8))
plt.scatter(typhoon_data['经度'], typhoon_data['纬度'], alpha=0.5, s=10, c='blue')
plt.title("台风经纬度分布")
plt.xlabel("经度")
plt.ylabel("纬度")
plt.grid(True)
plt.show()

# 5. 台风路径示例 (随机选取 20 条台风路径)
world_map_path = 'ne_110m_admin_0_countries.shp'
try:
    world = gpd.read_file(world_map_path)
except FileNotFoundError:
    print(f"Error: World map file not found at path {world_map_path}. Please check the file path.")
    exit()

# 创建 GeoDataFrame
gdf = gpd.GeoDataFrame(
    typhoon_data,
```

---

---

```
geometry=gpd.points_from_xy(typhoon_data['经度'], typhoon_data['纬度']),
crs="EPSG:4326" # 使用 WGS 84 坐标系统
)

# 随机选取 20 条台风路径
random_typhoons = np.random.choice(typhoon_data['台风编号'].unique(), size=20, replace=False)

# 使用不同的颜色绘制每个台风的路径
colors = plt.cm.tab20.colors # 使用 20 种颜色

plt.figure(figsize=(15, 10))
ax = world.plot(color='lightgrey', edgecolor='black')

for i, typhoon_id in enumerate(random_typhoons):
    selected_typhoon = gdf[gdf['台风编号'] == typhoon_id]

    # 获取台风的中文名称
    typhoon_name = selected_typhoon['台风中文名称'].iloc[0]

    # 绘制路径为直线
    ax.plot(selected_typhoon['经度'], selected_typhoon['纬度'],
            linestyle='-', linewidth=2, color=colors[i % len(colors)], label=f'{typhoon_name} ({typhoon_id})')

# 添加图例
plt.legend(title='台风名称', bbox_to_anchor=(1.05, 1), loc='upper left', fontsize='small',
           title_fontsize='medium', borderaxespad=0.)

plt.title("20 条台风路径")
plt.xlabel("经度")
plt.ylabel("纬度")
plt.xlim(-180, 180)
plt.ylim(-90, 90)
plt.grid(True)
plt.show()

# 6. 台风位置地理图 (随机选取 100 条台风)
# 随机选取 100 条台风
random_typhoons_geo = np.random.choice(typhoon_data['台风编号'].unique(), size=100,
                                         replace=False)

# 筛选出随机选取的台风数据
selected_geo_data = typhoon_data[typhoon_data['台风编号'].isin(random_typhoons_geo)]

# 创建一个 GeoDataFrame
```

---

---

```
gdf_geo = gpd.GeoDataFrame(  
    selected_geo_data,  
    geometry=gpd.points_from_xy(selected_geo_data['经度'], selected_geo_data['纬度']),  
    crs="EPSG:4326" # 使用 WGS 84 坐标系统  
)  
  
# 创建绘图  
plt.figure(figsize=(15, 10))  
ax = world.plot(color='lightgrey', edgecolor='black')  
  
# 在地图上绘制所有选定台风的位置  
gdf_geo.plot(ax=ax, marker='o', color='red', markersize=5, alpha=0.6, label='Typhoon Locations')  
  
plt.title("台风经纬度分布")  
plt.xlabel("经度")  
plt.ylabel("纬度")  
plt.xlim(-180, 180)  
plt.ylim(-90, 90)  
plt.grid(True)  
plt.legend()  
plt.show()  
columns_to_test = ['台风强度', '台风等级', '风速', '气压', '移动方向', '移动速度']  
  
for column in columns_to_test:  
    column_data = data[column].dropna()  
    # 正态分布的 K-S 检验  
    standardized_data = (column_data - column_data.mean()) / column_data.std()  
    ks_stat, p_norm = stats.kstest(standardized_data, 'norm')  
    if p_norm > 0.05:  
        print(f'{column} 列符合正态分布 (p = {p_norm:.4f})')  
    else:  
        print(f'{column} 列不符合正态分布 (p = {p_norm:.4f})')  
  
    # 均匀分布的 K-S 检验  
    scaled_data = (column_data - column_data.min()) / (column_data.max() - column_data.min())  
    ks_stat, p_unif = stats.kstest(scaled_data, 'uniform')  
    if p_unif > 0.05:  
        print(f'{column} 列符合均匀分布 (p = {p_unif:.4f})')  
    else:  
        print(f'{column} 列不符合均匀分布 (p = {p_unif:.4f})')  
  
# 解释  
# h_norm 和 h_unif 的值为 0 表示未拒绝零假设，数据符合对应的分布。  
# p 值表示检验的显著性水平，通常与显著性水平（例如 0.05）比较，以确定结果。
```

---

---

```
# 2. Spearman 相关性分析
# 由于数据不符合正态分布，使用 Spearman 相关系数来分析变量之间的关系
columns_for_corr = ['台风强度', '台风等级', '风速', '气压', '移动方向', '移动速度']

# 提取各列数据，去除缺失值
valid_data = data[columns_for_corr].dropna()

# 计算 Spearman 相关系数矩阵
spearman_corr_matrix = valid_data.corr(method='spearman')

# 显示相关性矩阵和显著性水平
print('Spearman 相关系数矩阵：')
print(spearman_corr_matrix)

# 可视化 Spearman 相关性矩阵
plt.figure(figsize=(10, 8))
sns.heatmap(spearman_corr_matrix, annot=True, cmap='jet', center=0, vmin=-1, vmax=1,
            linewidths=.5)
plt.title('Spearman 相关性矩阵热图')
plt.xlabel('变量')
plt.ylabel('变量')
plt.show()
```

---

### 3.Kmeans 算法

---

代码 3 Kmeans 代码

---

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import cartopy.crs as ccrs
import cartopy.feature as cfeature
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

# 设置图形风格
sns.set(style="whitegrid")
file_path = 'processed_typhoon_data.csv'
typhoon_data = pd.read_csv(file_path)
plt.rcParams['font.sans-serif'] = ['SimHei'] # 使用 SimHei 字体
plt.rcParams['axes.unicode_minus'] = False # 正确显示负号

# 数据预处理
typhoon_data = typhoon_data.dropna(subset=['台风等级', '风速', '气压', '移动方向', '移动速度', '经度',
                                             '纬度', '月'])

# 定义季节函数
```

---

---

```
def get_season(month):
    if month in [6, 7, 8]:
        return 'Summer'
    elif month in [9, 10, 11]:
        return 'Autumn'
    else:
        return 'Other'

# 添加季节特征
typhoon_data['Season'] = typhoon_data['月'].apply(get_season)

# 分割数据集
summer_data = typhoon_data[typhoon_data['Season'] == 'Summer']
autumn_data = typhoon_data[typhoon_data['Season'] == 'Autumn']

# 选择用于 K-means 分类的特征
features_summer = summer_data[['台风等级', '风速', '气压', '移动方向', '移动速度', '经度', '纬度']]
features_autumn = autumn_data[['台风等级', '风速', '气压', '移动方向', '移动速度', '经度', '纬度']]

# 标准化特征
scaler_summer = StandardScaler()
features_scaled_summer = scaler_summer.fit_transform(features_summer)

scaler_autumn = StandardScaler()
features_scaled_autumn = scaler_autumn.fit_transform(features_autumn)

# 选择聚类数目
k = 3
kmeans_summer = KMeans(n_clusters=k, random_state=42)
summer_data['Cluster'] = kmeans_summer.fit_predict(features_scaled_summer)

kmeans_autumn = KMeans(n_clusters=k, random_state=42)
autumn_data['Cluster'] = kmeans_autumn.fit_predict(features_scaled_autumn)

# 将包含聚类结果的数据框保存为新的 CSV 文件
output_file_summer = 'Kmeans_classification_results_summer.csv'
summer_data.to_csv(output_file_summer, index=False)

output_file_autumn = 'Kmeans_classification_results_autumn.csv'
autumn_data.to_csv(output_file_autumn, index=False)

print(f"Summer K-means classification results saved to {output_file_summer}")
print(f"Autumn K-means classification results saved to {output_file_autumn}")

# 可视化结果
```

---

---

```
def plot_clusters_on_map(data, title, output_filename):
    # 随机选择 300 条数据
    sampled_data = data.sample(n=300, random_state=42)

    fig = plt.figure(figsize=(10, 6))
    ax = fig.add_subplot(1, 1, 1, projection=ccrs.PlateCarree())

    # 添加地理特征
    ax.add_feature(cfeature.LAND)
    ax.add_feature(cfeature.COASTLINE)
    ax.add_feature(cfeature.BORDERS, linestyle=':')

    sns.scatterplot(data=sampled_data, x='经度', y='纬度', hue='Cluster',
                    palette='Set1', style='Cluster', markers=["o", "s", "D"],
                    legend='full', transform=ccrs.PlateCarree())

    plt.title(title)
    plt.xlabel('经度')
    plt.ylabel('纬度')
    plt.legend(title='Cluster')
    plt.grid()

    plt.savefig(output_filename, dpi=300)
    plt.show()

# 为夏季和秋季数据分别绘制地图
plot_clusters_on_map(summer_data, 'K-means Clustering of Summer Typhoons',
'summer_clusters.png')
plot_clusters_on_map(autumn_data, 'K-means Clustering of Autumn Typhoons', 'autumn_clusters.png')
```

---

#### 4. 随机森林方法

---

代码 4	随机森林方法代码
------	----------

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import precision_score, f1_score, roc_curve, auc, precision_recall_curve

# 设置图形风格
sns.set(style="whitegrid")
file_path = 'processed_typhoon_data.csv'
typhoon_data = pd.read_csv(file_path)
plt.rcParams['font.sans-serif'] = ['SimHei'] # 使用 SimHei 字体
```

---

---

```
plt.rcParams['axes.unicode_minus'] = False # 正确显示负号

typhoon_data = typhoon_data.dropna(subset=['台风等级', '风速', '气压', '移动方向', '移动速度'])

# 生成 Typhoon_Season 标签
typhoon_data['Typhoon_Season'] = typhoon_data['月'].apply(lambda x: 'Summer' if x in [6, 7, 8] else ('Fall' if x in [9, 10, 11] else 'Other'))

# 选择特征和标签
X = typhoon_data[['台风等级', '风速', '气压', '移动方向', '移动速度']]
y = typhoon_data['Typhoon_Season']

# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 训练随机森林分类模型
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train, y_train)

# 进行预测
y_pred = rf_classifier.predict(X_test)

# 计算 Precision 和 F1 分数
precision = precision_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

# 打印分类报告
print(classification_report(y_test, y_pred))
print(f"Precision: {precision:.2f}")
print(f"F1 Score: {f1:.2f}")

# 混淆矩阵
conf_matrix = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(10, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=rf_classifier.classes_, yticklabels=rf_classifier.classes_)
plt.title('随机森林分类模型混淆矩阵')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# 计算 ROC 曲线和 AUC
```

---

---

```
y_scores = rf_classifier.predict_proba(X_test) # 获取每个类的概率
for i in range(len(rf_classifier.classes_)):
    fpr, tpr, _ = roc_curve(y_test, y_scores[:, i], pos_label=i)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f'ROC curve (class {rf_classifier.classes_[i]}) (area = {roc_auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--') # 画出对角线
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('假阳性率')
plt.ylabel('真正率')
plt.title('接收者操作特征曲线 (ROC)')
plt.legend(loc='lower right')
plt.show()

# 计算精度-召回曲线
plt.figure(figsize=(10, 6))
for i in range(len(rf_classifier.classes_)):
    precision, recall, _ = precision_recall_curve(y_test, y_scores[:, i], pos_label=i)
    plt.plot(recall, precision, label=f'Precision-Recall curve (class {rf_classifier.classes_[i]})')

plt.xlabel('召回率 recall')
plt.ylabel('精度 precision')
plt.title('精度-召回曲线')
plt.legend(loc='lower left')
plt.show()
```

---

## 5. 支持向量机方法代码

---

代码 5	支持向量机方法代码
------	-----------

---

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import precision_score, f1_score, roc_curve, roc_auc_score, precision_recall_curve
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

# 设置图形风格
sns.set(style="whitegrid")
file_path = 'processed_typhoon_data.csv'
typhoon_data = pd.read_csv(file_path)
plt.rcParams['font.sans-serif'] = ['SimHei'] # 使用 SimHei 字体
plt.rcParams['axes.unicode_minus'] = False # 正确显示负号
```

---

---

```
# 假
typhoon_data = typhoon_data.dropna(subset=['台风等级', '风速', '气压', '移动方向', '移动速度'])

# 生成 Typhoon_Season 标签
typhoon_data['Typhoon_Season'] = typhoon_data['月'].apply(lambda x: 'Summer' if x in [6, 7, 8] else ('Fall' if x in [9, 10, 11] else 'Other'))

# 选择特征和标签
X = typhoon_data[['台风等级', '风速', '气压', '移动方向', '移动速度']]
y = typhoon_data['Typhoon_Season'] # 现在使用新生成的标签

# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 训练支持向量机模型
svm_classifier = SVC(kernel='linear', random_state=42)
svm_classifier.fit(X_train, y_train)

# 进行预测
y_pred = svm_classifier.predict(X_test)

# 计算 Precision 和 F1 分数
precision = precision_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

# 打印分类报告
print(classification_report(y_test, y_pred))
print(f"Precision: {precision:.2f}")
print(f"F1 Score: {f1:.2f}")

# 混淆矩阵
conf_matrix = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(10, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=svm_classifier.classes_, yticklabels=svm_classifier.classes_)
plt.title('支持向量机分类模型混淆矩阵')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# 计算 ROC 曲线和 AUC
y_score = svm_classifier.decision_function(X_test) # 返回的是每类的决策函数值
```

---

---

```
n_classes = len(svm_classifier.classes_) # 获取类别数量

# 初始化存储各个类别的 FPR, TPR 和 AUC 值
fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(n_classes):
    # 将 y_test 转换为 one-hot 编码形式，便于计算每个类别的 ROC
    y_test_one_hot = np.array([1 if c == svm_classifier.classes_[i] else 0 for c in y_test])

    # 计算当前类别的 FPR, TPR
    fpr[i], tpr[i], _ = roc_curve(y_test_one_hot, y_score[:, i], pos_label=1)
    roc_auc[i] = roc_auc_score(y_test_one_hot, y_score[:, i])

# 绘制 ROC 曲线
plt.figure(figsize=(10, 6))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], lw=2, label=f'{svm_classifier.classes_[i]} ROC curve (AUC = {roc_auc[i]:.2f})')
plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--')
plt.xlabel('假阳性率')
plt.ylabel('真正率')
plt.title('接收者操作特征曲线 (ROC)')
plt.legend(loc='lower right')
plt.grid()
plt.show()

# 精度-召回曲线
n_classes = len(svm_classifier.classes_)

# 初始化存储各个类别的 Precision, Recall
precision = dict()
recall = dict()

for i in range(n_classes):
    # 将 y_test 转换为 one-hot 编码形式，便于计算每个类别的 Precision-Recall
    y_test_one_hot = np.array([1 if c == svm_classifier.classes_[i] else 0 for c in y_test])

    # 计算当前类别的 Precision, Recall
    precision[i], recall[i], _ = precision_recall_curve(y_test_one_hot, y_score[:, i], pos_label=1)

# 绘制 Precision-Recall 曲线
plt.figure(figsize=(10, 6))
for i in range(n_classes):
```

---

---

```
plt.plot(recall[i], precision[i], color='green', lw=2, label=f'{svm_classifier.classes_[i]} Precision-Recall curve')
plt.xlabel('召回率 recall')
plt.ylabel('精度 precision')
plt.title('精度-召回曲线')
plt.legend(loc='best')
plt.grid()
plt.show()
```

---

## 6.LSTM 模型方法

---

代码 6

LSTM 方法代码

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, precision_score, f1_score
from sklearn.metrics import roc_curve, auc, precision_recall_curve
from sklearn.preprocessing import LabelEncoder
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout
from keras.utils import to_categorical
from keras.callbacks import EarlyStopping, ReduceLROnPlateau
from keras.layers import BatchNormalization

# 设置图形风格
sns.set(style="whitegrid")
file_path = 'processed_typhoon_data.csv'
typhoon_data = pd.read_csv(file_path)
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False

# 数据预处理
typhoon_data = typhoon_data.dropna(subset=['台风等级', '风速', '气压', '移动方向', '移动速度'])
typhoon_data['Typhoon_Season'] = typhoon_data['月'].apply(lambda x: 'Summer' if x in [6, 7, 8] else ('Fall' if x in [9, 10, 11] else 'Other'))

# 特征和标签
X = typhoon_data[['台风等级', '风速', '气压', '移动方向', '移动速度']]
y = typhoon_data['Typhoon_Season']

# 标签编码
le = LabelEncoder()
y_encoded = le.fit_transform(y)
y_categorical = to_categorical(y_encoded)
```

---

---

```
# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y_categorical, test_size=0.3, random_state=42)

# 数据重塑以适应 LSTM 输入格式
X_train = X_train.values.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.values.reshape(X_test.shape[0], X_test.shape[1], 1)

# 构建 LSTM 模型
model = Sequential()
model.add(LSTM(100, return_sequences=True, input_shape=(X_train.shape[1], 1)))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(LSTM(50, return_sequences=True))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(LSTM(25))
model.add(Dropout(0.2))
model.add(Dense(20, activation='relu'))
model.add(Dense(y_categorical.shape[1], activation='softmax'))

# 编译模型
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# 设置早停策略和学习率调度
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=1e-6)

# 训练模型
history = model.fit(X_train, y_train, epochs=200, batch_size=16, validation_split=0.2,
                     callbacks=[early_stopping, reduce_lr])

# 进行预测
y_pred_prob = model.predict(X_test)
y_pred = np.argmax(y_pred_prob, axis=1)

# 计算 Precision 和 F1 分数
precision = precision_score(np.argmax(y_test, axis=1), y_pred, average='weighted')
f1 = f1_score(np.argmax(y_test, axis=1), y_pred, average='weighted')

# 打印分类报告
print(classification_report(np.argmax(y_test, axis=1), y_pred, target_names=le.classes_))
print(f"Precision: {precision:.2f}")
print(f"F1 Score: {f1:.2f}")

# 混淆矩阵
```

---

---

```
conf_matrix = confusion_matrix(np.argmax(y_test, axis=1), y_pred)

plt.figure(figsize=(10, 6))
sns.heatmap(conf_matrix,      annot=True,      fmt='d',      cmap='Blues',      xticklabels=le.classes_,
yticklabels=le.classes_)
plt.title('LSTM 分类模型混淆矩阵')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# 绘制训练过程中的准确率和损失
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='训练准确率')
plt.plot(history.history['val_accuracy'], label='验证准确率')
plt.title('模型准确率')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='训练损失')
plt.plot(history.history['val_loss'], label='验证损失')
plt.title('模型损失')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.show()

# 绘制 ROC 曲线
y_test_labels = np.argmax(y_test, axis=1)
fpr = {}
tpr = {}
roc_auc = {}
n_classes = y_categorical.shape[1]

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_labels, y_pred_prob[:, i], pos_label=i)
    roc_auc[i] = auc(fpr[i], tpr[i])

# 绘制每个类别的 ROC 曲线
plt.figure(figsize=(10, 6))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f'ROC 曲线 (类别 {le.classes_[i]}), AUC = {roc_auc[i]:.2f}')

```

---

---

```
plt.plot([0, 1], [0, 1], 'k--')
plt.title('多类别 ROC 曲线')
plt.xlabel('假阳性率 (FPR)')
plt.ylabel('真阳性率 (TPR)')
plt.legend(loc='lower right')
plt.show()
```

---

## 7. 图神经网络方法代码

---

代码 7

图神经网络方法代码

---

```
import pandas as pd
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from torch_geometric.data import Data
from torch_geometric.nn import GCNConv, GATConv
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc,
precision_recall_curve
from sklearn.neighbors import kneighbors_graph # 导入 KNN
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler

# 设置图形风格
sns.set(style="whitegrid")
file_path = 'processed_typhoon_data.csv'
typhoon_data = pd.read_csv(file_path)
plt.rcParams['font.sans-serif'] = ['SimHei'] # 使用 SimHei 字体
plt.rcParams['axes.unicode_minus'] = False # 正确显示负号

# 生成 Typhoon_Season 标签
typhoon_data['Typhoon_Season'] = typhoon_data['月'].apply(lambda x: 'Summer' if x in [6, 7, 8] else
('Fall' if x in [9, 10, 11] else 'Other'))
# 选择特征并处理
X = typhoon_data[['台风等级', '风速', '气压', '移动方向', '移动速度']].values
label_encoder = LabelEncoder()

# 将台风等级转换为数字编码作为标签
y_encoded = label_encoder.fit_transform(typhoon_data['Typhoon_Season'])

# 数据标准化
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

---

---

```
# 使用 KNN 创建稀疏图的边索引
knn_graph = kneighbors_graph(X, n_neighbors=10, mode='connectivity', include_self=False)
edges = knn_graph.nonzero()
edge_index = torch.tensor(edges, dtype=torch.long)

# 创建图数据对象
data = Data(x=torch.tensor(X, dtype=torch.float), edge_index=edge_index, y=torch.tensor(y_encoded,
dtype=torch.long))

# 划分训练集和测试集
train_mask = torch.zeros(data.num_nodes, dtype=torch.bool)
test_mask = torch.zeros(data.num_nodes, dtype=torch.bool)

train_indices, test_indices = train_test_split(range(data.num_nodes), test_size=0.3, random_state=42)
train_mask[train_indices] = True
test_mask[test_indices] = True

# 定义 GNN 模型
class GNNModel(nn.Module):
    def __init__(self):
        super(GNNModel, self).__init__()
        self.conv1 = GCNConv(X.shape[1], 64)
        self.conv2 = GATConv(64, 64, heads=4)
        self.conv3 = GCNConv(64 * 4, 128)
        self.dropout = nn.Dropout(0.3)
        self.fc = nn.Linear(128, len(np.unique(y_encoded)))

    def forward(self, data):
        x, edge_index = data.x, data.edge_index
        x = self.conv1(x, edge_index)
        x = torch.relu(x)
        x = self.dropout(x)
        x = self.conv2(x, edge_index)
        x = torch.relu(x)
        x = self.dropout(x)
        x = self.conv3(x, edge_index)
        x = torch.relu(x)
        x = self.dropout(x)
        x = self.fc(x)
        return x

# 实例化模型
model = GNNModel()
```

---

---

```
optimizer = optim.Adam(model.parameters(), lr=0.005)
criterion = nn.CrossEntropyLoss()

# 记录损失值
loss_values = []

# 训练模型
model.train()
for epoch in range(200):
    optimizer.zero_grad()
    out = model(data)
    loss = criterion(out[train_mask], data.y[train_mask])
    loss.backward()
    optimizer.step()

    loss_values.append(loss.item()) # 记录损失值
    if epoch % 5 == 0:
        print(f'Epoch {epoch}, Loss: {loss.item()}')

# 绘制损失曲线
plt.figure(figsize=(10, 6))
plt.plot(loss_values, label='Loss')
plt.title('Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

# 测试模型
model.eval()
with torch.no_grad():
    pred = model(data).argmax(dim=1)
    test_acc = (pred[test_mask] == data.y[test_mask]).sum().item() / test_mask.sum().item()
    print(f'Test Accuracy: {test_acc:.4f}')

    # 计算 F1 分数
    report = classification_report(data.y[test_mask].numpy(), pred[test_mask].numpy(),
                                    output_dict=True)
    f1_score = report['weighted avg']['f1-score']
    print(f'F1 Score: {f1_score:.4f}')

    # 混淆矩阵
    conf_matrix = confusion_matrix(data.y[test_mask].numpy(), pred[test_mask].numpy())

plt.figure(figsize=(10, 6))
```

---

---

```

sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=label_encoder.classes_,
            yticklabels=label_encoder.classes_)
plt.title('图神经网络模型混淆矩阵')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# 计算 ROC 曲线和 AUC
y_scores = model(data).softmax(dim=1).numpy()
plt.figure(figsize=(10, 6))
for i in range(len(label_encoder.classes_)):
    fpr, tpr, _ = roc_curve(data.y[test_mask].numpy(), y_scores[test_mask][:, i], pos_label=i)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f'ROC curve (class {label_encoder.classes_[i]}) (area = {roc_auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--') # 画出对角线
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('假阳率')
plt.ylabel('真正率')
plt.title('接收者操作特征曲线 (ROC)')
plt.legend(loc='lower right')
plt.show()

# 计算精度-召回曲线
plt.figure(figsize=(10, 6))
for i in range(len(label_encoder.classes_)):
    precision, recall, _ = precision_recall_curve(data.y[test_mask].numpy(), y_scores[test_mask][:,
        i], pos_label=i)
    plt.plot(recall, precision, label=f'Precision-Recall curve (class {label_encoder.classes_[i]})')

plt.xlabel('召回率 recall')
plt.ylabel('精度 precision')
plt.title('精度-召回曲线')
plt.legend(loc='lower left')
plt.show()

```

---

## 8.2024 年台风数据预处理及可视化代码

---

代码 8

2024 年台风数据预处理及可视化代码

---

```

import pandas as pd
import matplotlib.pyplot as plt
import geopandas as gpd
from collections import defaultdict
import numpy as np

```

---

# 设置绘图字体和正确显示负号

---

---

```
plt.rcParams['font.sans-serif'] = ['SimHei'] # 使用 SimHei 字体
plt.rcParams['axes.unicode_minus'] = False # 正确显示负号

# 读取 2024 年台风数据
filename_2024 = '2024 台风数据.csv'
data_2024 = pd.read_csv(filename_2024)

# 提取需要的列来绘制台风路径
longitude = data_2024['经度']
latitude = data_2024['纬度']
typhoon_ids = data_2024['台风英文名称']

# 获取唯一的台风编号
unique_typhoon_ids_2024 = typhoon_ids.unique()

# 使用不同颜色绘制每一个台风的路径，增强对比度
colors = plt.colormaps['tab20'](np.linspace(0, 1, len(unique_typhoon_ids_2024)))
plt.figure(figsize=(10, 8))

# 加载并绘制岸线数据
coastline_filename = 'GSHHS_L1.shp'
try:
    coastline_data = gpd.read_file(coastline_filename)
    coastline_data.plot(ax=plt.gca(), color='black', linewidth=0.5)
except FileNotFoundError:
    print(f"无法找到岸线文件: {coastline_filename}, 请确认文件路径是否正确。")

# 限定绘图范围为北纬 10-30 度，东经 100-140 度
plt.xlim([100, 140])
plt.ylim([10, 30])

# 定义中国省份的边界框（经纬度范围）
provinces = {
    'Guangdong': [109.5, 117.5, 20.1, 25.5],
    'Fujian': [116.7, 120.7, 23.4, 28.4],
    'Zhejiang': [118.0, 123.0, 27.2, 31.5],
    'Hainan': [108.6, 111.0, 18.0, 20.5],
    'Jiangsu': [116.3, 121.9, 30.5, 35.0],
    'Shandong': [114.7, 122.7, 34.4, 38.4],
    'Guangxi': [104.5, 112.0, 20.5, 26.5],
    'Shanghai': [120.9, 122.1, 30.7, 31.9]
}

# 判定每个台风经过的省份
passed_provinces = defaultdict(list)
```

---

---

```
for typhoon_id in unique_typhoon_ids_2024:
    indices = typhoon_ids == typhoon_id
    typhoon_longitudes = longitude[indices].values
    typhoon_latitudes = latitude[indices].values

    # 判定台风路径是否经过某个省份
    for province_name, bounds in provinces.items():
        lon_min, lon_max, lat_min, lat_max = bounds
        if np.any((typhoon_longitudes >= lon_min) & (typhoon_longitudes <= lon_max) &
                  (typhoon_latitudes >= lat_min) & (typhoon_latitudes <= lat_max)):
            passed_provinces[typhoon_id].append(province_name)

    # 显示每个台风经过的省份
for typhoon_id in unique_typhoon_ids_2024:
    provinces_passed = ', '.join(passed_provinces[typhoon_id])
    print(f'台风 {typhoon_id} 经过的省份: {provinces_passed}')

# 绘制台风路径
for i, typhoon_id in enumerate(unique_typhoon_ids_2024):
    indices = typhoon_ids == typhoon_id
    plt.plot(longitude[indices], latitude[indices], color=colors[i], label=f' 台 风 {typhoon_id}', linewidth=1.5)

# 图形设置
plt.xlabel('经度')
plt.ylabel('纬度')
plt.title('2024 年台风路径')
plt.legend()
plt.grid(True)
plt.show()

import pandas as pd
import matplotlib.pyplot as plt
import geopandas as gpd
from collections import defaultdict
```

---

## 9.2024 年台风分类代码

---

代码 8 2024 年台风数据预处理及可视化代码

---

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, f1_score
from datetime import datetime
```

---

# 设置绘图字体和正确显示负号代码 8 2024 年台风数据预处理及可视化代码

```
import pandas as pd
import matplotlib.pyplot as plt
import geopandas as gpd
from collections import defaultdict
import numpy as np
```

```
# 设置绘图字体和正确显示负号
plt.rcParams['font.sans-serif'] = ['SimHei'] # 使用 SimHei 字体
plt.rcParams['axes.unicode_minus'] = False # 正确显示负号
```

```
# 读取 2024 年台风数据
filename_2024 = '2024 台风数据.csv'
data_2024 = pd.read_csv(filename_2024)
```

```
# 提取需要的列来绘制台风路径
longitude = data_2024['经度']
latitude = data_2024['纬度']
typhoon_ids = data_2024['台风英文名称']
```

```
# 获取唯一的台风编号
unique_typhoon_ids_2024 = typhoon_ids.unique()
```

```
# 使用不同颜色绘制每一个台风的路径，增强对比度
colors = plt.colormaps['tab20'](np.linspace(0, 1, len(unique_typhoon_ids_2024)))
plt.figure(figsize=(10, 8))
```

```
# 加载并绘制岸线数据
coastline_filename = 'GSHHS_L1.shp'
try:
    coastline_data = gpd.read_file(coastline_filename)
    coastline_data.plot(ax=plt.gca(), color='black', linewidth=0.5)
except FileNotFoundError:
    print(f"无法找到岸线文件: {coastline_filename}, 请确认文件路径是否正确。")
```

```
# 限定绘图范围为北纬 10-30 度，东经 100-140 度
plt.xlim([100, 140])
plt.ylim([10, 30])
```

```
# 定义中国省份的边界框（经纬度范围）
provinces = {
    'Guangdong': [109.5, 117.5, 20.1, 25.5],
    'Fujian': [116.7, 120.7, 23.4, 28.4],
    'Zhejiang': [118.0, 123.0, 27.2, 31.5],
    'Hainan': [108.6, 111.0, 18.0, 20.5],
```

---

---

```
'Jiangsu': [116.3, 121.9, 30.5, 35.0],
'Shandong': [114.7, 122.7, 34.4, 38.4],
'Guangxi': [104.5, 112.0, 20.5, 26.5],
'Shanghai': [120.9, 122.1, 30.7, 31.9]
}

# 判定每个台风经过的省份
passed_provinces = defaultdict(list)
for typhoon_id in unique_typhoon_ids_2024:
    indices = typhoon_ids == typhoon_id
    typhoon_longitudes = longitude[indices].values
    typhoon_latitudes = latitude[indices].values

    # 判定台风路径是否经过某个省份
    for province_name, bounds in provinces.items():
        lon_min, lon_max, lat_min, lat_max = bounds
        if np.any((typhoon_longitudes >= lon_min) & (typhoon_longitudes <= lon_max) &
                  (typhoon_latitudes >= lat_min) & (typhoon_latitudes <= lat_max)):
            passed_provinces[typhoon_id].append(province_name)

    # 显示每个台风经过的省份
    for typhoon_id in unique_typhoon_ids_2024:
        provinces_passed = ', '.join(passed_provinces[typhoon_id])
        print(f'台风 {typhoon_id} 经过的省份: {provinces_passed}')

    # 绘制台风路径
    for i, typhoon_id in enumerate(unique_typhoon_ids_2024):
        indices = typhoon_ids == typhoon_id
        plt.plot(longitude[indices], latitude[indices], color=colors[i], label=f' 台 风 {typhoon_id}', linewidth=1.5)

    # 图形设置
    plt.xlabel('经度')
    plt.ylabel('纬度')
    plt.title('2024 年台风路径')
    plt.legend()
    plt.grid(True)
    plt.show()

import pandas as pd
import matplotlib.pyplot as plt
import geopandas as gpd
from collections import defaultdict
import numpy as np

plt.rcParams['font.sans-serif'] = ['SimHei'] # 使用 SimHei 字体
```

---

---

```
plt.rcParams['axes.unicode_minus'] = False # 正确显示负号

# 读取 2024 年台风数据
filename_2024 = '2024 台风数据.csv'
data_2024 = pd.read_csv(filename_2024)

# 提取与模型对应的特征列
classification_columns = ['风速', '气压', '移动速度']

# 将台风起始时间解析为月份，并根据月份确定台风类别
typhoon_start_time_2024 = data_2024['台风起始时间']
typhoon_start_time_2024 = typhoon_start_time_2024.astype(str)
start_dates_2024 = pd.to_datetime(typhoon_start_time_2024, format='%Y%m%d%H', errors='coerce')
months_2024 = start_dates_2024.dt.month

# 根据月份确定台风类别（夏台风、秋台风）
data_2024['台风类别'] = "其他"
summer_months = [6, 7, 8]
autumn_months = [9, 10, 11]
data_2024.loc[months_2024.isin(summer_months), '台风类别'] = "夏台风"
data_2024.loc[months_2024.isin(autumn_months), '台风类别'] = "秋台风"

# 读取历史台风数据
filename_history = 'Kmeans_classification_results.csv' # 假设文件名
data_history = pd.read_csv(filename_history)

# 确保历史数据包含台风类别标签，并将台风类别转化为数值型编码
data_history['Cluster'] = data_history['Cluster'].astype('category').cat.codes

# 提取训练特征和标签
X = data_history[classification_columns].dropna()
y = data_history['Cluster'].loc[X.index]

# 划分训练集和验证集
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# 初始化并训练随机森林分类模型
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train, y_train)

# 评估模型在验证集上的性能
y_pred_val = rf_classifier.predict(X_val)
precision = precision_score(y_val, y_pred_val, average='weighted')
f1 = f1_score(y_val, y_pred_val, average='weighted')
print(f'Validation Precision: {precision:.2f}, F1 Score: {f1:.2f}')
```

---

---

```
# 使用训练好的模型对 2024 年数据进行预测
classification_data_2024 = data_2024[classification_columns].dropna()
data_2024['台风类别编码'] = rf_classifier.predict(classification_data_2024)

# 将预测的类别加入到 2024 年的数据表中
data_2024['台风类别预测'] = data_2024['台风类别编码'].replace(
    {0: '其他', 1: '夏台风', 2: '秋台风'}
)

# 确保同一编号下的台风具有相同的分类结果
unique_typhoon_ids_2024 = data_2024['台风编号'].unique()

for typhoon_id in unique_typhoon_ids_2024:
    indices = data_2024['台风编号'] == typhoon_id
    current_clusters = data_2024.loc[indices, '台风类别预测']
    unique_cluster = current_clusters.unique()
    if len(unique_cluster) > 1:
        unified_cluster = current_clusters.mode()[0]
        data_2024.loc[indices, '台风类别预测'] = unified_cluster

# 保存处理后的 2024 年台风数据
data_2024.to_csv('2024 年台风分类结果随机森林.csv', index=False)
```

---

## 10.2 问题二代码

读取数据

```
import pandas as pd
import numpy as np
import struct
import os
import dask.dataframe as dd

# 加载.bin 文件
def load_bin_file(file_path, dtype):
    with open(file_path, 'rb') as f:
        data = np.fromfile(f, dtype=dtype)
    return data

# 路线数据
path = './data/中国近海台风路径集合(1945-2023)/1945-2023.csv'
path2 = './data/processed_1945-2023.xlsx'
path3 = './data/中国近海台风路径集合(1945-2023)/1945-2023.xlsx'
path4 = './data/processed_typhoon_data(1).csv'

# 加载路线数据
typhoon_route = pd.read_csv(path4, encoding='gbk')
```

## 特征工程

```
def preprocess_typhoon_route(typhoon_route):
    # 转换时间格式
    time_columns = ['台风起始时间', '台风结束时间', '当前台风时间']
    for col in time_columns:
        typhoon_route[col] = pd.to_datetime(typhoon_route[col], errors='coerce')

    # 台风强度数值化映射
    intensity_abbr_mapping = {
        '超强台风(Super TY)': 1,
        '强热带风暴(STS)': 2,
        '强台风(STY)': 3,
        '热带低压(TD)': 4,
        '热带风暴(TS)': 5,
        '台风(TY)': 6
    }

    # 更详细的方向与角度的映射
    detailed_direction_mapping = {
        '西西北': 292.5,
        '西北': 315,
        'NW': 315,
        '西': 270,
        'W': 270,
        '偏西': 270,
        '北西北': 337.5,
        '北': 0,
        '偏北': 0,
        '东北': 45,
        '北东北': 22.5,
        '西西南': 247.5,
        '东东北': 67.5,
        '北北东': 22.5,
        '西北北': 337.5,
        '北西': 315,
        '东北东': 67.5,
        'ENE 2': 67.5,
        '东': 90,
        '偏东': 90,
        '西南': 225,
        '西南西': 202.5,
        '东北北': 22.5,
        '西北偏西': 292.5,
        '北北西': 337.5,
        '北东': 45,
```

```

'东东南': 112.5,
'南': 180,
'偏南': 180,
'南西南': 202.5,
'东北偏北': 22.5,
'南西': 247.5,
'东南': 135,
'东南东': 112.5,
'南东南': 157.5,
'东北偏东': 67.5,
'西南偏西': 247.5,
'南南西': 202.5,
'西南南': 225,
'南南东': 157.5,
'东南偏东': 112.5,
'北偏西': 315,
'南东': 135,
'西北偏北': 330,
'西北西': 296.25,
'WNW': 296.25,
}

# 使用 map() 将台风强度转换为英文缩写
typhoon_route['台风强度'] = typhoon_route['台风强度'].map(intensity_abbr_mapping)

# 使用 map() 将移动方向转换为英文缩写
typhoon_route[' 移动 方向 '] = typhoon_route[' 移动 方向 '].map(detailed_direction_mapping)

# 先将时间列转换为日期时间格式
typhoon_route['台风起始时间'] = pd.to_datetime(typhoon_route['台风起始时间'])
typhoon_route['台风结束时间'] = pd.to_datetime(typhoon_route['台风结束时间'])
typhoon_route['当前台风时间'] = pd.to_datetime(typhoon_route['当前台风时间'])

# 计算特征
# 台风已持续时间（以小时为单位）
typhoon_route['台风已持续时间_小时'] = (typhoon_route['当前台风时间'] - typhoon_route['台风起始时间']).dt.total_seconds() / 3600

# 台风预计总持续时间（以小时为单位）
typhoon_route['台风预计总持续时间_小时'] = (typhoon_route['台风结束时间'] - typhoon_route['台风起始时间']).dt.total_seconds() / 3600

# 台风预计剩余时间（以小时为单位）

```

```

typhoon_route['台风预计剩余时间_小时'] = (typhoon_route['台风结束时间'] - typhoon_route['当前台风时间']).dt.total_seconds() / 3600

# 提取周期性特征
typhoon_route['Hour'] = typhoon_route['当前台风时间'].dt.hour
typhoon_route['Day'] = typhoon_route['当前台风时间'].dt.day
typhoon_route['Quarter'] = typhoon_route['当前台风时间'].dt.quarter

# 计算台风在当前时间的持续时间
typhoon_route['台风持续状态'] = (typhoon_route['当前台风时间'] >= typhoon_route['台风起始时间']) & (typhoon_route['当前台风时间'] <= typhoon_route['台风结束时间'])

# 用 1 表示持续中，0 表示未持续
typhoon_route['台风持续状态'] = typhoon_route['台风持续状态'].astype(int)

# 添加季节性特征
typhoon_route['Hour_sin'] = np.sin(2 * np.pi * typhoon_route['Hour'] / 24)
typhoon_route['Hour_cos'] = np.cos(2 * np.pi * typhoon_route['Hour'] / 24)
typhoon_route['Day_sin'] = np.sin(2 * np.pi * typhoon_route['Day'] / 31)
typhoon_route['Day_cos'] = np.cos(2 * np.pi * typhoon_route['Day'] / 31)
typhoon_route['Month_sin'] = np.sin(2 * np.pi * typhoon_route['Month'] / 12)
typhoon_route['Month_cos'] = np.cos(2 * np.pi * typhoon_route['Month'] / 12)
typhoon_route['Quarter_sin'] = np.sin(2 * np.pi * typhoon_route['Quarter'] / 4)
typhoon_route['Quarter_cos'] = np.cos(2 * np.pi * typhoon_route['Quarter'] / 4)

# 添加历史特征
typhoon_route['24 小时平均风速'] = typhoon_route['风速'].rolling(window=24, min_periods=1).mean()
typhoon_route['24 小时平均强度'] = typhoon_route['台风强度'].rolling(window=24, min_periods=1).mean()

# 添加距离特征
typhoon_route['经度差'] = typhoon_route['经度'].diff()
typhoon_route['纬度差'] = typhoon_route['纬度'].diff()
# typhoon_route['移动速度'] = np.sqrt(typhoon_route['经度差']**2 + typhoon_route['纬度差']**2) / typhoon_route['台风已持续时间_小时']

# 添加风速变化率
# typhoon_route['风速变化率'] = typhoon_route['风速'].diff() / typhoon_route['台风已持续时间_小时']

# 添加角度变化率
# typhoon_route['方向变化率'] = typhoon_route['移动方向'].diff() / typhoon_route['台风已持续时间_小时']

```

```
typhoon_route = preprocess_typhoon_route(typhoon_route)
typhoon_route['经度差'].fillna(0, inplace=True)
typhoon_route['纬度差'].fillna(0, inplace=True)
```

检验数据集缺失值数量和画图显示

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
plt.rcParams["font.sans-serif"]=[ "SimHei"] #设置字体
plt.rcParams["axes.unicode_minus"] = False #该语句解决图像中的“-”负号的乱码问题

nan_count = typhoon_route.isna().sum()

# 将 nan_count 转换为 DataFrame 便于可视化
nan_count_df = nan_count.reset_index()
nan_count_df.columns = ['特征', '缺失值数量']
print(nan_count_df)

# 绘制缺失值图表
plt.figure(figsize=(12, 6))
sns.barplot(x='特征', y='缺失值数量', data=nan_count_df.sort_values(by='缺失值数量',
ascending=False))
plt.title('特征中的缺失值数量')
plt.xlabel('特征')
plt.ylabel('缺失值数量')
plt.xticks(rotation=45, ha='right') # 旋转 x 轴标签以便更好地显示
plt.tight_layout()
plt.show()
```

构建在世界地图上的路径图

```
import pandas as pd
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import cartopy.feature as cfeature
from matplotlib.colors import ListedColormap
import numpy as np

# 将时间字段转换为 datetime 类型
typhoon_route['台风起始时间'] = pd.to_datetime(typhoon_route['台风起始时间'])
typhoon_route['台风结束时间'] = pd.to_datetime(typhoon_route['台风结束时间'])
typhoon_route['当前台风时间'] = pd.to_datetime(typhoon_route['当前台风时间'])

# 过滤出当前时间在起始时间和结束时间之间的数据
typhoon_route = typhoon_route[(typhoon_route['当前台风时间'] >= typhoon_route['台风起
```

```

始时间']) &
        (typhoon_route['当前台风时间'] <= typhoon_route['台
风结束时间'])]

# 创建地图
fig, ax = plt.subplots(figsize=(20, 15), subplot_kw={'projection': ccrs.PlateCarree()})

# 添加海岸线
ax.add_feature(cfeature.COASTLINE)
ax.add_feature(cfeature.BORDERS, linestyle=':')
ax.add_feature(cfeature.LAND, edgecolor='black')
ax.add_feature(cfeature.OCEAN)

# 获取所有台风编号
typhoon_ids = typhoon_route['台风编号'].unique()

# 创建颜色映射
cmap = plt.cm.get_cmap('tab20', len(typhoon_ids))

# 绘制台风路径
for i, typhoon_id in enumerate(typhoon_ids):
    group = typhoon_route[typhoon_route['台风编号'] == typhoon_id]
    color = cmap(i)
    ax.plot(group['经度'], group['纬度'], marker='o', color=color, markersize=5,
            label=typhoon_id, transform=ccrs.PlateCarree())
    ax.plot(group['经度'], group['纬度'], color=color, linewidth=1, label=None,
            transform=ccrs.PlateCarree())

# 设置地图范围
ax.set_global()

# 添加标题和图例
ax.set_title('台风路径图', fontsize=20)
ax.legend(loc='upper left', bbox_to_anchor=(1, 1), fontsize=12)

# 显示地图
plt.show()

```

## 模型训练

数据集切分

```

import pandas as pd
import numpy as np

```

```

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# 特征选择
features = ['台风强度', '台风等级', '风速', '气压', '移动方向', 'Year',
             'Month', '台风已持续时间_小时', 'Hour', 'Day', 'Quarter', '台风持续状态',
             'Hour_sin', 'Hour_cos', 'Day_sin', 'Day_cos',
             'Month_sin', 'Month_cos', 'Quarter_sin', 'Quarter_cos', '24 小时平均风速', '24
             小时平均强度', '经度差', '纬度差']

# 标签
labels = ['经度', '纬度']

# 分割数据集
X = typhoon_route[features]
y = typhoon_route[labels]

# 标准化特征
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 分割训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

```

### LSTM 模型训练

```

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, Bidirectional
from sklearn.metrics import mean_squared_error, mean_absolute_error
from tensorflow.keras.optimizers import Adam

# 重塑数据以适应 LSTM 输入
X_train_lstm = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
X_test_lstm = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))

# 构建 LSTM 模型
lstm_model = Sequential()
lstm_model.add(Bidirectional(LSTM(128, return_sequences=True,
                                 input_shape=(X_train_lstm.shape[1], X_train_lstm.shape[2]))))
lstm_model.add(Dropout(0.2))
lstm_model.add(LSTM(128, return_sequences=False))
lstm_model.add(Dropout(0.2))

```

```

lstm_model.add(Dense(64, activation='relu'))
lstm_model.add(Dense(2))

# 编译模型
optimizer = Adam(learning_rate=0.001)
lstm_model.compile(loss='mean_squared_error', optimizer=optimizer)

# 训练模型
lstm_model.fit(X_train_lstm, y_train, epochs=100, batch_size=64, verbose=1,
validation_split=0.2)

# 预测
lstm_pred = lstm_model.predict(X_test_lstm)

```

### 随机森林模型训练

```

from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectFromModel

# 数据预处理
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 特征选择
selector = SelectFromModel(RandomForestRegressor(n_estimators=100, random_state=42))
selector.fit(X_train_scaled, y_train)
X_train_selected = selector.transform(X_train_scaled)
X_test_selected = selector.transform(X_test_scaled)

# 定义随机森林模型
rf_model = RandomForestRegressor(random_state=42)

# 定义超参数搜索空间
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2']
}

# 使用网格搜索进行超参数调优

```

```

grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=5,
scoring='neg_mean_squared_error', n_jobs=-1)
grid_search.fit(X_train_selected, y_train)

# 获取最佳模型
best_rf_model = grid_search.best_estimator_

# 预测
rf_pred = best_rf_model.predict(X_test_selected)

# 打印最佳超参数
print("Best parameters found: ", grid_search.best_params_)

```

### XGBoost 模型训练代码

```

import xgboost as xgb
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectFromModel

# 数据预处理
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 特征选择
selector = SelectFromModel(xgb.XGBRegressor(n_estimators=100, random_state=42))
selector.fit(X_train_scaled, y_train)
X_train_selected = selector.transform(X_train_scaled)
X_test_selected = selector.transform(X_test_scaled)

# 定义 XGBoost 模型
xgb_model = xgb.XGBRegressor(random_state=42)

# 定义超参数搜索空间
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 5, 7, 9],
    'learning_rate': [0.01, 0.05, 0.1, 0.2],
    'subsample': [0.8, 0.9, 1.0],
    'colsample_bytree': [0.8, 0.9, 1.0],
    'gamma': [0, 0.1, 0.2],
    'min_child_weight': [1, 3, 5]
}

```

```

# 使用网格搜索进行超参数调优
grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid, cv=5,
scoring='neg_mean_squared_error', n_jobs=-1)
grid_search.fit(X_train_selected, y_train)

# 获取最佳模型
best_xgb_model = grid_search.best_estimator_

# 预测
xgb_pred = best_xgb_model.predict(X_test_selected)

# 打印最佳超参数
print("Best parameters found: ", grid_search.best_params_)

```

### LightGBM 模型训练

```

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error
import lightgbm as lgb
from tqdm import tqdm

# 分别处理经度和纬度# 分别处理经度和纬度
y_train_lon = y_train['经度']
y_train_lat = y_train['纬度']

y_test_lon = y_test['经度']
y_test_lat = y_test['纬度']

# 定义 LightGBM 模型
lgb_model = lgb.LGBMRegressor(device='gpu', random_state=42)

# 使用网格搜索进行超参数调优
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.05, 0.1],
    'max_depth': [3, 5, 7],
    'min_child_samples': [1, 2, 4]
}

# 训练经度预测模型
grid_search_lon = GridSearchCV(estimator=lgb_model, param_grid=param_grid, cv=5,
scoring='neg_mean_squared_error', n_jobs=-1)

```

```

with tqdm(total=len(param_grid['n_estimators']) * len(param_grid['learning_rate']) *
len(param_grid['max_depth']) * len(param_grid['min_child_samples'])) * 5) as pbar:
    grid_search_lon.fit(X_train, y_train_lon)
    pbar.update(1)
best_lgb_model_lon = grid_search_lon.best_estimator_
lgb_pred_lon = best_lgb_model_lon.predict(X_test)

# 训练纬度预测模型
grid_search_lat = GridSearchCV(estimator=lgb_model, param_grid=param_grid, cv=5,
scoring='neg_mean_squared_error', n_jobs=-1)
with tqdm(total=len(param_grid['n_estimators']) * len(param_grid['learning_rate']) *
len(param_grid['max_depth']) * len(param_grid['min_child_samples'])) * 5) as pbar:
    grid_search_lat.fit(X_train, y_train_lat)
    pbar.update(1)
best_lgb_model_lat = grid_search_lat.best_estimator_
lgb_pred_lat = best_lgb_model_lat.predict(X_test)

# 合并预测结果
lgb_pred = np.column_stack((lgb_pred_lon, lgb_pred_lat))

```

### CatBoost 模型训练

```

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error
from catboost import CatBoostRegressor
from tqdm import tqdm

# 分别处理经度和纬度
y_train_lon = y_train['经度']
y_train_lat = y_train['纬度']

y_test_lon = y_test['经度']
y_test_lat = y_test['纬度']

# 定义 CatBoost 模型
cat_model = CatBoostRegressor(task_type='GPU', random_state=42)

# 使用网格搜索进行超参数调优
param_grid = {
    'n_estimators': [100, 200], # 树的数量
    'learning_rate': [0.01, 0.05, 0.1], # 学习率
    'max_depth': [3, 5, 7], # 树的深度
    'min_child_samples': [1, 2, 4] # 叶子节点最小样本数
}

```

```

}

# 训练经度预测模型
grid_search_lon = GridSearchCV(estimator=cat_model, param_grid=param_grid, cv=5,
scoring='neg_mean_squared_error', n_jobs=2)
with tqdm(total=len(param_grid['n_estimators']) * len(param_grid['learning_rate']) *
len(param_grid['max_depth']) * len(param_grid['min_child_samples']) * 5) as pbar:
    grid_search_lon.fit(X_train, y_train_lon)
    pbar.update(1)
best_cat_model_lon = grid_search_lon.best_estimator_
cat_pred_lon = best_cat_model_lon.predict(X_test)

# 训练纬度预测模型
grid_search_lat = GridSearchCV(estimator=cat_model, param_grid=param_grid, cv=5,
scoring='neg_mean_squared_error', n_jobs=2)
with tqdm(total=len(param_grid['n_estimators']) * len(param_grid['learning_rate']) *
len(param_grid['max_depth']) * len(param_grid['min_child_samples']) * 5) as pbar:
    grid_search_lat.fit(X_train, y_train_lat)
    pbar.update(1)
best_cat_model_lat = grid_search_lat.best_estimator_
cat_pred_lat = best_cat_model_lat.predict(X_test)

# 合并预测结果
cat_pred = np.column_stack((cat_pred_lon, cat_pred_lat))

```

## 加权投票预测

```

from sklearn.metrics import mean_squared_error, mean_absolute_error

# 计算每个模型的均方误差
mse_lstm = mean_squared_error(y_test, lstm_pred)
mse_rf = mean_squared_error(y_test, rf_pred)
# mse_gbm = mean_squared_error(y_test, gbm_pred)
mse_xgb = mean_squared_error(y_test, xgb_pred)
# mse_arima = mean_squared_error(y_test, arima_pred)
mse_lgb = mean_squared_error(y_test, lgb_pred)
# mse_sqs = mean_squared_error(y_test_seq.reshape(-1, 2), y_pred_seq.reshape(-1, 2))
# mse_sarima = mean_squared_error(y_test, sarima_pred)
mse_cat = mean_squared_error(y_test, cat_pred)

# 计算每个模型的平均绝对误差
mae_lstm = mean_absolute_error(y_test, lstm_pred)
mae_rf = mean_absolute_error(y_test, rf_pred)
# mae_gbm = mean_absolute_error(y_test, gbm_pred)
mae_xgb = mean_absolute_error(y_test, xgb_pred)
# mae_arima = mean_absolute_error(y_test, arima_pred)

```

```

mae_lgb = mean_absolute_error(y_test, lgb_pred)
# mae_sqs = mean_absolute_error(y_test_seq.reshape(-1, 2), y_pred_seq.reshape(-1, 2))
# mae_sarima = mean_absolute_error(y_test, sarima_pred)
mae_cat = mean_absolute_error(y_test, cat_pred)

# 打印每个模型的性能
print(f'LSTM MSE: {mse_lstm}, MAE: {mae_lstm}')
print(f'Random Forest MSE: {mse_rf}, MAE: {mae_rf}')
# print(f'GBM MSE: {mse_gbm}, MAE: {mae_gbm}')
print(f'XGBoost MSE: {mse_xgb}, MAE: {mae_xgb}')
# print(f'ARIMA MSE: {mse_arima}, MAE: {mae_arima}')
print(f'LightGBM MSE: {mse_lgb}, MAE: {mae_lgb}')
# print(f'sqs MSE: {mse_sqs}, MAE: {mae_sqs}')
# print(f'sarima MSE: {mse_sarima}, MAE: {mae_sarima}')
print(f'CatBoost MSE: {mse_cat}, MAE: {mae_cat}')

# 计算每个模型的权重
total_mse = mse_lstm + mse_rf + mse_xgb + mse_lgb + mae_cat
weights = [
    mse_lstm / total_mse,
    mse_rf / total_mse,
    mse_xgb / total_mse,
    mse_lgb / total_mse,
    mae_cat / total_mse
]

# 计算每个模型的归一化误差
min_error = min([mse_lstm, mse_rf, mse_xgb, mse_lgb, mse_cat])
max_error = max([mse_lstm, mse_rf, mse_xgb, mse_lgb, mse_cat])
normalized_errors = [(mse - min_error) / (max_error - min_error) for mse in [mse_lstm, mse_rf, mse_xgb, mse_lgb, mse_cat]]

# 计算权重
weights = [1 - ne for ne in normalized_errors]
total_weights = sum(weights)
weights = [w / total_weights for w in weights]

# 打印权重
print("Model weights:", weights)

# 初始化综合预测结果
ensemble_pred = np.zeros_like(lstm_pred)

```

```

# 加权投票
ensemble_pred += weights[0] * lstm_pred
ensemble_pred += weights[1] * rf_pred
ensemble_pred += weights[2] * xgb_pred
ensemble_pred += weights[3] * lgb_pred
ensemble_pred += weights[4] * cat_pred

# 评估综合模型
mse = mean_squared_error(y_test, ensemble_pred)
mae = mean_absolute_error(y_test, ensemble_pred)
print(f"Ensemble MSE: {mse}, MAE: {mae}")

```

### 13号贝碧嘉台风的路径预测

```

import pandas as pd

bejia_data = pd.read_csv('./贝碧嘉台风路径(改).csv')

def preprocess_typhoon_route(typhoon_route):
    # 转换时间格式
    time_columns = ['时间']
    for col in time_columns:
        typhoon_route[col] = pd.to_datetime(typhoon_route[col], errors='coerce')

    # 台风强度数值化映射
    intensity_abbr_mapping = {
        '超强台风(Super TY)': 1,
        '强热带风暴(STS)': 2,
        '强台风(STY)': 3,
        '热带低压(TD)': 4,
        '热带风暴(TS)': 5,
        '台风(TY)': 6
    }

    # 更详细的方向与角度的映射
    detailed_direction_mapping = {
        '西西北': 292.5,
        '西北': 315,
        'NW': 315,
        '西': 270,
        'W': 270,
        '偏西': 270,
        '北西北': 337.5,
        '北': 0,
        '偏北': 0,
    }

```

```

'东北': 45,
'北东北': 22.5,
'西西南': 247.5,
'东东北': 67.5,
'北北东': 22.5,
'西北北': 337.5,
'北西': 315,
'东北东': 67.5,
'ENE 2': 67.5,
'东': 90,
'偏东': 90,
'西南': 225,
'西南西': 202.5,
'东北北': 22.5,
'西北偏西': 292.5,
'北北西': 337.5,
'北东': 45,
'东东南': 112.5,
'南': 180,
'偏南': 180,
'南西南': 202.5,
'东北偏北': 22.5,
'南西': 247.5,
'东南': 135,
'东南东': 112.5,
'南东南': 157.5,
'东北偏东': 67.5,
'西南偏西': 247.5,
'南南西': 202.5,
'西南南': 225,
'南南东': 157.5,
'东南偏东': 112.5,
'北偏西': 315,
'南东': 135,
'西北偏北': 330,
'西北西': 296.25,
'WNW': 296.25,
}

# 先将时间列转换为日期时间格式
typhoon_route['时间'] = pd.to_datetime(typhoon_route['时间'])

# 提取周期性特征
typhoon_route['Year'] = typhoon_route['时间'].dt.year
typhoon_route['Month'] = typhoon_route['时间'].dt.month

```

```

typhoon_route['Hour'] = typhoon_route['时间'].dt.hour
typhoon_route['Day'] = typhoon_route['时间'].dt.day
typhoon_route['Quarter'] = typhoon_route['时间'].dt.quarter

# 计算台风在当前时间的持续时间
typhoon_route['台风持续状态'] = 1

# 添加季节性特征
typhoon_route['Hour_sin'] = np.sin(2 * np.pi * typhoon_route['Hour'] / 24)
typhoon_route['Hour_cos'] = np.cos(2 * np.pi * typhoon_route['Hour'] / 24)
typhoon_route['Day_sin'] = np.sin(2 * np.pi * typhoon_route['Day'] / 31)
typhoon_route['Day_cos'] = np.cos(2 * np.pi * typhoon_route['Day'] / 31)
typhoon_route['Month_sin'] = np.sin(2 * np.pi * typhoon_route['Month'] / 12)
typhoon_route['Month_cos'] = np.cos(2 * np.pi * typhoon_route['Month'] / 12)
typhoon_route['Quarter_sin'] = np.sin(2 * np.pi * typhoon_route['Quarter'] / 4)
typhoon_route['Quarter_cos'] = np.cos(2 * np.pi * typhoon_route['Quarter'] / 4)

# 添加历史特征
typhoon_route['24 小时平均风速'] = typhoon_route['风速 (m/s)'].rolling(window=24,
min_periods=1).mean()
typhoon_route['24 小时平均强度'] = typhoon_route['强度'].rolling(window=24,
min_periods=1).mean()

# 添加距离特征
typhoon_route['经度差'] = typhoon_route['经度'].diff()
typhoon_route['纬度差'] = typhoon_route['纬度'].diff()

return typhoon_route
preprocess_typhoon_route(bejia_data)

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
plt.rcParams["font.sans-serif"]=[ "SimHei"] #设置字体
plt.rcParams["axes.unicode_minus"] = False #该语句解决图像中的“-”负号的乱码问题

load_que_data(bejia_data)

# 将 NaN 替换为 0
bejia_data.fillna(0, inplace=True)

# 特征选择
features = ['强度', '风力 (级)', '风速 (m/s)', '移动方向', '移动速度 (km/h)',  

'中心气压 (hPa)', 'Year', 'Month', 'Hour', 'Day', 'Quarter', '台风持续状态',  

'Hour_sin', 'Hour_cos', 'Day_sin', 'Day_cos', 'Month_sin', 'Month_cos',

```

```

'Quarter_sin', 'Quarter_cos', '24 小时平均风速', '24 小时平均强度', '经度差', '纬度差'
']
# 标签
labels = ['经度', '纬度']
# 分割数据集
X_pre = bejia_data[features]
y = bejia_data[labels]
# 标准化特征
scaler = StandardScaler()
bejia_data_scaled = scaler.fit_transform(X_pre)

# 重塑数据以适应 LSTM 输入
bejia_data_lstm = bejia_data_scaled.reshape((bejia_data_scaled.shape[0], 1,
bejia_data_scaled.shape[1]))

# 使用训练好的模型进行预测
lstm_pred = lstm_model.predict(bejia_data_lstm)

rf_pred = rf_model.predict(bejia_data_scaled)

xgb_pred = xgb_model.predict(bejia_data_scaled)

lgb_pred_lon = best_lgb_model_lon.predict(bejia_data_scaled)
lgb_pred_lat = best_lgb_model_lat.predict(bejia_data_scaled)
lgb_pred = np.column_stack((lgb_pred_lon, lgb_pred_lat))

cat_pred_lon = best_cat_model_lon.predict(bejia_data_scaled)
cat_pred_lat = best_cat_model_lat.predict(bejia_data_scaled)
cat_pred = np.column_stack((cat_pred_lon, cat_pred_lat))

# 计算每个模型的均方误差
mse_lstm = mean_squared_error(y, lstm_pred)
mse_rf = mean_squared_error(y, rf_pred)
mse_xgb = mean_squared_error(y, xgb_pred)
mse_lgb = mean_squared_error(y, lgb_pred)
mse_cat = mean_squared_error(y, cat_pred)

# 计算每个模型的平均绝对误差
mae_lstm = mean_absolute_error(y, lstm_pred)
mae_rf = mean_absolute_error(y, rf_pred)
mae_xgb = mean_absolute_error(y, xgb_pred)
mae_lgb = mean_absolute_error(y, lgb_pred)
mae_cat = mean_absolute_error(y, cat_pred)

# 打印每个模型的性能

```

```

print(f"LSTM MSE: {mse_lstm}, MAE: {mae_lstm}")
print(f"Random Forest MSE: {mse_rf}, MAE: {mae_rf}")
print(f"XGBoost MSE: {mse_xgb}, MAE: {mae_xgb}")
print(f"LightGBM MSE: {mse_lgb}, MAE: {mae_lgb}")
print(f"CatBoost MSE: {mse_cat}, MAE: {mae_cat}")

# 计算每个模型的权重
total_mse = mse_lstm + mse_rf + mse_xgb + mse_lgb + mse_cat
weights = [
    mse_lstm / total_mse,
    mse_rf / total_mse,
    mse_xgb / total_mse,
    mse_lgb / total_mse,
    mse_cat / total_mse
]

# 计算每个模型的归一化误差
min_error = min([mse_lstm, mse_rf, mse_xgb, mse_lgb, mse_cat])
max_error = max([mse_lstm, mse_rf, mse_xgb, mse_lgb, mse_cat])
normalized_errors = [(mse - min_error) / (max_error - min_error) for mse in [mse_lstm, mse_rf, mse_xgb, mse_lgb, mse_cat]]

# 计算权重
weights = [1 - ne for ne in normalized_errors]
total_weights = sum(weights)
weights = [w / total_weights for w in weights]

# 打印权重
print("Model weights:", weights)

# 初始化综合预测结果
ensemble_pred = np.zeros_like(lstm_pred)

# 加权投票
ensemble_pred += weights[0] * lstm_pred
ensemble_pred += weights[1] * rf_pred
ensemble_pred += weights[2] * xgb_pred
ensemble_pred += weights[3] * lgb_pred
ensemble_pred += weights[4] * cat_pred

# 评估综合模型
mse = mean_squared_error(y, ensemble_pred)
mae = mean_absolute_error(y, ensemble_pred)
print(f"Ensemble MSE: {mse}, MAE: {mae}")

```

数据描述性分析的 3 个图，即不同强度台风的经纬度分布图、台风强度与风速的关系图、每年台风的数量图

```
# 可视化经纬度数据分布
plt.figure(figsize=(10, 6))
sns.scatterplot(data=typhoon_route, x='经度', y='纬度', hue='台风强度', palette='viridis',
size='风速', sizes=(20, 200))
plt.title('台风经纬度分布')
plt.xlabel('经度')
plt.ylabel('纬度')
plt.legend()
plt.show()

# 分析台风强度与其他变量的关系
plt.figure(figsize=(12, 6))
sns.boxplot(x='台风强度', y='风速', data=typhoon_route)
plt.title('台风强度与风速的关系')
plt.xlabel('台风强度')
plt.ylabel('风速')
plt.show()

# 时间序列分析
typhoon_route['台风起始时间'] = pd.to_datetime(typhoon_route['台风起始时间'])
typhoon_route['Year'] = typhoon_route['台风起始时间'].dt.year
typhoon_route['Month'] = typhoon_route['台风起始时间'].dt.month

# 统计每年的台风数量
typhoon_count = typhoon_route['Year'].value_counts().sort_index()
plt.figure(figsize=(10, 6))
typhoon_count.plot(kind='bar')
plt.title('每年台风数量')
plt.xlabel('年份')
plt.ylabel('台风数量')
plt.show()

# 移动方向和速度的分析
plt.figure(figsize=(12, 6))
sns.scatterplot(data=typhoon_route, x='移动速度', y='移动方向', hue='台风等级',
palette='coolwarm')
plt.title('移动速度与移动方向的关系')
plt.xlabel('移动速度 (km/h)')
plt.ylabel('移动方向 (°)')
```

```
plt.legend()  
plt.show()
```

### 各模型预测值和真实值对比图

```
import matplotlib.pyplot as plt  
import numpy as np  
  
# 绘制各个模型的预测值与真实值对比图  
models = ['LSTM', 'Random Forest', 'XGBoost', 'LightGBM', 'CatBoost']  
predictions = [lstm_pred, rf_pred, xgb_pred, lgb_pred, cat_pred]  
  
plt.figure(figsize=(20, 10))  
  
for i, (model, pred) in enumerate(zip(models, predictions)):  
    plt.subplot(2, 5, i + 1)  
    plt.scatter(y_test['经度'], pred[:, 0], alpha=0.5)  
    plt.plot([y_test['经度'].min(), y_test['经度'].max()], [y_test['经度'].min(), y_test['经度'].max()], 'k--', lw=2)  
    plt.xlabel('True Longitude')  
    plt.ylabel('Predicted Longitude')  
    plt.title(f'{model} Predicted vs True Longitude')  
  
    plt.subplot(2, 5, i + 6)  
    plt.scatter(y_test['纬度'], pred[:, 1], alpha=0.5)  
    plt.plot([y_test['纬度'].min(), y_test['纬度'].max()], [y_test['纬度'].min(), y_test['纬度'].max()], 'k--', lw=2)  
    plt.xlabel('True Latitude')  
    plt.ylabel('Predicted Latitude')  
    plt.title(f'{model} Predicted vs True Latitude')  
  
plt.tight_layout()  
plt.show()
```

### 不同模型的预测误差分布情况图

```
# 计算每个模型的预测误差  
errors = {  
    'LSTM': np.sqrt((y_test['经度'] - lstm_pred[:, 0])**2 + (y_test['纬度'] - lstm_pred[:, 1])**2),  
    'Random Forest': np.sqrt((y_test['经度'] - rf_pred[:, 0])**2 + (y_test['纬度'] - rf_pred[:, 1])**2),  
    'XGBoost': np.sqrt((y_test['经度'] - xgb_pred[:, 0])**2 + (y_test['纬度'] - xgb_pred[:, 1])**2),  
    'LightGBM': np.sqrt((y_test['经度'] - lgb_pred[:, 0])**2 + (y_test['纬度'] - lgb_pred[:, 1])**2),  
    'CatBoost': np.sqrt((y_test['经度'] - cat_pred[:, 0])**2 + (y_test['纬度'] - cat_pred[:, 1])**2),
```

```

1])**2)
}

# 绘制误差分布图
plt.figure(figsize=(12, 8))
for model, error in errors.items():
    sns.kdeplot(error, label=model, shade=True)
plt.title('Error Distribution of Models')
plt.xlabel('Error')
plt.ylabel('Density')
plt.legend()
plt.show()

```

预测值和真实值的路径对比图（在世界地图上）

```

import pandas as pd
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import cartopy.feature as cfeature
import numpy as np

# 创建一个 DataFrame 来存储真实值和预测值
data = {
    'True Longitude': y['经度'],
    'True Latitude': y['纬度'],
    'LSTM Predicted Longitude': lstm_pred[:, 0],
    'LSTM Predicted Latitude': lstm_pred[:, 1],
    'Random Forest Predicted Longitude': rf_pred[:, 0],
    'Random Forest Predicted Latitude': rf_pred[:, 1],
    'XGBoost Predicted Longitude': xgb_pred[:, 0],
    'XGBoost Predicted Latitude': xgb_pred[:, 1],
    'LightGBM Predicted Longitude': lgb_pred[:, 0],
    'LightGBM Predicted Latitude': lgb_pred[:, 1],
    'CatBoost Predicted Longitude': cat_pred[:, 0],
    'CatBoost Predicted Latitude': cat_pred[:, 1],
    'Ensemble Predicted Longitude': ensemble_pred[:, 0],
    'Ensemble Predicted Latitude': ensemble_pred[:, 1]
}

df = pd.DataFrame(data)

# 创建地图
fig, axes = plt.subplots(2, 3, figsize=(20, 15), subplot_kw={'projection': ccrs.PlateCarree()})
axes = axes.flatten()

# 模型列表

```

```

models = ['LSTM', 'Random Forest', 'XGBoost', 'LightGBM', 'CatBoost', 'Ensemble']

# 绘制每个模型的预测路径和真实路径
for i, model in enumerate(models):
    ax = axes[i]

    # 添加海岸线和陆地
    ax.add_feature(cfeature.COASTLINE)
    ax.add_feature(cfeature.LAND, facecolor='lightgrey')

    # 绘制真实路径
    ax.plot(df['True Longitude'], df['True Latitude'], label='True Path', color='black',
            linewidth=2, transform=ccrs.PlateCarree())

    # 绘制预测路径
    ax.plot(df[f'{model} Predicted Longitude'], df[f'{model} Predicted Latitude'],
            label=f'{model} Predicted Path', color='red', linewidth=1.5, transform=ccrs.PlateCarree())

    # 添加图例
    ax.legend()

    # 设置标题和坐标轴标签
    ax.set_title(f'{model} vs True Path')
    ax.set_xlabel('Longitude')
    ax.set_ylabel('Latitude')

# 调整子图之间的间距
plt.tight_layout()

# 显示地图
plt.show()

```

使用 DTW 计算预测值与真实值的对比，并画图

```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from dtw import dtw
import cartopy.crs as ccrs
import cartopy.feature as cfeature

# 创建一个 DataFrame 来存储真实值和预测值
data = {
    'True Longitude': y['经度'],
    'True Latitude': y['纬度'],
    'LSTM Predicted Longitude': lstm_pred[:, 0],
}

```

```

'LSTM Predicted Latitude': lstm_pred[:, 1],
'Random Forest Predicted Longitude': rf_pred[:, 0],
'Random Forest Predicted Latitude': rf_pred[:, 1],
'XGBoost Predicted Longitude': xgb_pred[:, 0],
'XGBoost Predicted Latitude': xgb_pred[:, 1],
'LightGBM Predicted Longitude': lgb_pred[:, 0],
'LightGBM Predicted Latitude': lgb_pred[:, 1],
'CatBoost Predicted Longitude': cat_pred[:, 0],
'CatBoost Predicted Latitude': cat_pred[:, 1],
'Ensemble Predicted Longitude': ensemble_pred[:, 0],
'Ensemble Predicted Latitude': ensemble_pred[:, 1]
}

df = pd.DataFrame(data)

# 计算每个模型的预测路径与真实路径之间的 DTW 距离
models = ['LSTM', 'Random Forest', 'XGBoost', 'LightGBM', 'CatBoost', 'Ensemble']
dtw_distances = {}

for model in models:
    # 计算经度和纬度的 DTW 距离
    dtw_lon = dtw(df['True Longitude'], df[f'{model} Predicted Longitude']).distance
    dtw_lat = dtw(df['True Latitude'], df[f'{model} Predicted Latitude']).distance
    dtw_distances[model] = (dtw_lon + dtw_lat) / 2

# 打印 DTW 距离
for model, distance in dtw_distances.items():
    print(f'{model} DTW Distance: {distance}')

# 创建地图
fig, axes = plt.subplots(2, 3, figsize=(20, 15), subplot_kw={'projection': ccrs.PlateCarree()})
axes = axes.flatten()

# 绘制每个模型的预测路径和真实路径
for i, model in enumerate(models):
    ax = axes[i]

    # 添加海岸线和陆地
    ax.add_feature(cfeature.COASTLINE)
    ax.add_feature(cfeature.LAND, facecolor='lightgrey')

    # 绘制真实路径
    ax.plot(df['True Longitude'], df['True Latitude'], label='True Path', color='black',
            linewidth=2, transform=ccrs.PlateCarree())

```

```

# 绘制预测路径
ax.plot(df[f'{model} Predicted Longitude'], df[f'{model} Predicted Latitude'],
label=f'{model} Predicted Path', color='red', linewidth=1.5, transform=ccrs.PlateCarree())

# 添加图例
ax.legend()

# 设置标题和坐标轴标签
ax.set_title(f'{model} vs True Path (DTW Distance: {dtw_distances[model]:.2f})')
ax.set_xlabel('Longitude')
ax.set_ylabel('Latitude')

# 调整子图之间的间距
plt.tight_layout()

# 显示地图
plt.show()

```

提取 2024 年 9 月 13 日到 17 日每日 14 点的数据

```

import pandas as pd

data = {
    'True Longitude': y['经度'],
    'True Latitude': y['纬度'],
    'Ensemble Predicted Longitude': ensemble_pred[:, 0],
    'Ensemble Predicted Latitude': ensemble_pred[:, 1]
}

# 合并真实值和预测值到原始数据中
df = pd.DataFrame(data)

# 将预测值和真实值合并到原始数据中
original_data = bejia_data.merge(df, left_index=True, right_index=True)
original_data[['时间', '经度', '纬度', 'Ensemble Predicted Longitude', 'Ensemble Predicted Latitude']]

# 提取 2024 年 9 月 13 日到 17 日每日 14 点的数据
specific_times = pd.to_datetime(['2024-09-13 14:00:00', '2024-09-14 14:00:00', '2024-09-15
14:00:00', '2024-09-16 14:00:00', '2024-09-17 14:00:00'])

filtered_data = original_data[original_data['时间'].isin(specific_times)]

# 打印提取的数据
print(filtered_data)

```

```
# 添加一列“台风中心位置(经度/纬度)”
filtered_data['台风中心位置(经度 / 纬度)'] = filtered_data.apply(lambda row:
f'{row["Ensemble Predicted Longitude"]:.2f}/{row["Ensemble Predicted Latitude"]:.2f}', axis=1)
```

## 10.3 问题三代码

### 9 降水量和风速关系代码

---

代码 9 降水量与风速关系代码

---

```
import os
import pyproj
import netCDF4 as nc
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import least_squares
from datetime import datetime, timedelta
import geopandas as gpd
import contextily as ctx
import requests
from requests.adapters import HTTPAdapter
from urllib3.util.retry import Retry
from xyzservices import TileProvider
from PIL import Image, ImageCms
import io

# 设置 PROJ_LIB 环境变量
proj_lib_path = r'D:\Anaconda\envs\DL\Library\share\proj'
os.environ['PROJ_LIB'] = proj_lib_path

# 或者使用 pyproj.datadir.set_data_dir 方法
# pyproj.datadir.set_data_dir(proj_lib_path)

# 设置绘图字体和正确显示负号
plt.rcParams['font.sans-serif'] = ['SimHei'] # 使用 SimHei 字体
plt.rcParams['axes.unicode_minus'] = False # 正确显示负号

# 读取 NetCDF 文件路径（使用已经提取的文件）
file_path = 'extracted_precipitation_2022.nc'

# 读取经度和纬度数据
dataset = nc.Dataset(file_path)
longitude = dataset.variables['longitude'][:]
latitude = dataset.variables['latitude'][:]
time = dataset.variables['time'][:]

# 将时间单位转换为日期，原时间是 "hours since 1961-01-01 00:00:00"
```

---

---

```
base_date = datetime(1961, 1, 1)
dates = [base_date + timedelta(hours=int(t)) for t in time]

# 读取降水量数据
precipitation_data = dataset.variables['precipitation'][:]

# 将小于 0 的降水量数据替换为 0
precipitation_data = np.maximum(precipitation_data, 0)

# 显示提取的数据的基本信息
print('提取的数据维度 (时间 x 纬度 x 经度): ', precipitation_data.shape)

# 读取台风路径数据 (从上传的 CSV 文件中读取并保留原始列标题)
filename = 'processed_typhoon_data.csv'
typhoon_data = pd.read_csv(filename)

# 将时间格式 YYYY/MM/DD HH:MM 转换为 datetime 格式
typhoon_data['Time'] = pd.to_datetime(typhoon_data['当前台风时间'].astype(str),
format='%Y/%m/%d %H:%M')

# 过滤出 2022 年的台风数据
typhoon_data_2022 = typhoon_data[typhoon_data['Time'].dt.year == 2022].copy()

# 使用 .loc 来避免设置副本的警告
typhoon_data_2022.loc[:, 'Time'] = pd.to_datetime(typhoon_data_2022['当前台风时间'].astype(str),
format='%Y/%m/%d %H:%M')

# 计算台风登陆后，降水量与台风中心的距离关系
distance_precipitation = []
for _, row in typhoon_data_2022.iterrows():
    typhoon_lat = row['纬度']
    typhoon_lon = row['经度']
    typhoon_time = row['Time']

    # 找到最接近的时间索引
    time_idx = np.argmin([abs((d - typhoon_time).total_seconds()) for d in dates])

    # 计算降水场中每个点与台风中心的距离
    lon_grid, lat_grid = np.meshgrid(longitude, latitude)
    distances = np.sqrt((lat_grid - typhoon_lat) ** 2 + (lon_grid - typhoon_lon) ** 2)

    # 提取对应时间的降水量数据
    precipitation_at_time = precipitation_data[time_idx, :, :]

    # 将距离和降水量对应起来，计算与台风中心的距离和降水量关系
    distance_precipitation.append((distances, precipitation_at_time))
```

---

---

```
distances = distances.flatten()
precipitation_at_time = precipitation_at_time.flatten()
valid_indices = ~np.isnan(precipitation_at_time)

# 确保有效索引与距离数据匹配
valid_distances = distances[valid_indices]
valid_precipitation = precipitation_at_time[valid_indices]

# 保存距离最小的降水量（即台风中心附近的降水量）
if len(valid_distances) > 0:
    min_idx = np.argmin(valid_distances)
    distance_precipitation.append({
        'Time': typhoon_time,
        'Distance': valid_distances[min_idx],
        'Precipitation': valid_precipitation[min_idx]
    })

# 转换为 DataFrame
distance_precipitation_df = pd.DataFrame(distance_precipitation)

# 将降水量和距离关系写入 Excel 表格
distance_precipitation_df.to_excel('typhoon_distance_precipitation_2022.xlsx', index=False)

# 使用待定系数公式拟合降水量数据
P_ref = 1013 # 参考气压，标准大气压 (hPa)

# 提取所需数据
D = distance_precipitation_df['Distance'].values
P_obs = distance_precipitation_df['Precipitation'].values

# 获取其他台风参数
V = typhoon_data_2022['移动速度'].values # 假设单位为 km/h
P_c = typhoon_data_2022['气压'].values # 修正为表中的列名 '气压'
V_max = typhoon_data_2022['风速'].values # 修正为表中的列名 '风速'

# 删除无效数据（如 D, V, P_c, V_max, P_obs 中包含 NaN 或 Inf 的值）
valid_indices = ~(np.isnan(D) | np.isnan(V) | np.isnan(P_c) | np.isnan(V_max) | np.isnan(P_obs) |
                  np.isinf(D) | np.isinf(V) | np.isinf(P_c) | np.isinf(V_max) | np.isinf(P_obs))

# 输出各个条件下被筛选的数据数量
print(f'总数据点数: {len(D)}')
print(f'NaN 或 Inf 值数量 (D, V, P_c, V_max, P_obs) : {np.sum(~valid_indices)}')
print(f'有效数据点数量: {np.sum(valid_indices)}')

# 确保至少有一些有效数据点
```

---

---

```
if np.sum(valid_indices) < 10:
    raise ValueError('有效数据点不足，无法进行拟合，请检查数据预处理步骤。')

D = D[valid_indices]
V = V[valid_indices]
P_c = P_c[valid_indices]
V_max = V_max[valid_indices]
P_obs = P_obs[valid_indices]

# 对数据进行标准化以提高拟合的稳定性
D_norm = (D - np.mean(D)) / np.std(D)
V_norm = (V - np.mean(V)) / np.std(V)
P_c_norm = (P_c - np.mean(P_c)) / np.std(P_c)
V_max_norm = (V_max - np.mean(V_max)) / np.std(V_max)

# 定义待定系数拟合的目标函数
def model_fun(params, X):
    A, alpha, beta = params
    D, V, P_c, V_max = X.T
    return A * np.exp(-alpha * D) * (1 / (V + np.finfo(float).eps)) * (1 / np.maximum(P_c - P_ref, 1)) *
    (np.abs(V_max) ** beta)

# 定义误差函数
def residuals(params, X, y):
    return model_fun(params, X) - y

# 初始猜测参数 A, alpha, beta
initial_params = [0.5, 0.5, 0.5] # 修改初始参数，使其不至于过小

# 定义拟合数据矩阵
X_data = np.vstack((D_norm, V_norm, P_c_norm, V_max_norm)).T

# 确保残差函数初始值是有限的
y_initial = model_fun(initial_params, X_data)
if not np.all(np.isfinite(y_initial)):
    raise ValueError('Initial residuals are not finite, please adjust initial parameters or check input data.')

# 进行拟合
result = least_squares(residuals, initial_params, bounds=[[0, 0, -np.inf], [np.inf, np.inf, np.inf]], args=(X_data, P_obs))

# 检查拟合是否成功
if not result.success:
    print('拟合未能成功收敛，请检查数据或尝试不同的初始参数。')
```

---

---

```
# 显示拟合结果
b_est = result.x
print('拟合的参数 (A, alpha, beta): ', b_est)

# 使用拟合的模型绘制拟合曲线和观测数据的对比
P_fit = model_fun(b_est, X_data)

plt.figure(figsize=(10, 6))
plt.scatter(D, P_obs, color='b', label='Observed Precipitation')
plt.plot(D, P_fit, color='r', linewidth=2, label='Fitted Model')
plt.xlabel('Distance to Typhoon Center (km)')
plt.ylabel('Precipitation (mm)')
plt.title('Observed vs Fitted Precipitation vs Distance to Typhoon Center')
plt.legend()
plt.grid(True)
plt.show()

# 将拟合结果保存到 CSV
fitted_results = pd.DataFrame({'Distance': D, 'Observed_Precipitation': P_obs, 'Fitted_Precipitation': P_fit})
fitted_results.to_csv('fitted_typhoon_precipitation_2022.csv', index=False)

# 创建 GeoDataFrame 用于地图绘制
# 确保两个 DataFrame 有共同的时间索引
distance_precipitation_df['Time'] = pd.to_datetime(distance_precipitation_df['Time'])
typhoon_data_2022['Time'] = pd.to_datetime(typhoon_data_2022['Time'])

# 合并数据
merged_data = pd.merge(typhoon_data_2022, distance_precipitation_df[['Time', 'Precipitation']], on='Time')

# 创建 GeoDataFrame
gdf = gpd.GeoDataFrame(
    merged_data,
    geometry=gpd.points_from_xy(merged_data['经度'], merged_data['纬度']),
    crs="EPSG:4326" # WGS84 坐标系
)

# 自定义请求会话，增加重试次数
session = requests.Session()
retry = Retry(total=5, backoff_factor=1, status_forcelist=[500, 502, 503, 504])
adapter = HTTPAdapter(max_retries=retry)
session.mount('http://', adapter)
session.mount('https://', adapter)
```

---

---

```
# 创建自定义瓦片提供者
class CustomTileProvider(TileProvider):
    def __init__(self, session):
        super().__init__(
            name='OpenStreetMap',
            url='https://tile.openstreetmap.org/{z}/{x}/{y}.png',
            attribution='&copy; OpenStreetMap contributors'
        )
        self.session = session

    def get_image(self, extent, zoom):
        url = self.build_url(extent, zoom)
        response = self.session.get(url)
        if response.status_code == 200:
            # 使用 PIL 处理图像并忽略颜色配置文件
            image = Image.open(io.BytesIO(response.content))
            image = ImageCms.profileToProfile(image, ImageCms.createProfile('sRGB'),
ImageCms.createProfile('sRGB'))
            return ctx.utils.imread(np.array(image))
        else:
            raise Exception(f"Failed to download tile from {url}")

# 使用自定义瓦片提供者添加底图
fig, ax = plt.subplots(figsize=(12, 8))
gdf.plot(ax=ax, column='风速', cmap='viridis', markersize=50, legend=True, legend_kwds={'label': 'Wind Speed (m/s)'})
ctx.add_basemap(ax, crs=gdf.crs, source=CustomTileProvider(session), attribution=None) # 使用 OpenStreetMap 作为底图
ax.set_title('Typhoon Path with Wind Speed (2022)')
plt.show()

# 绘制台风路径及降水量图
fig, ax = plt.subplots(figsize=(12, 8))
gdf.plot(ax=ax, column='Precipitation', cmap='Blues', markersize=50, legend=True, legend_kwds={'label': 'Precipitation (mm)'})
ctx.add_basemap(ax, crs=gdf.crs, source=CustomTileProvider(session), attribution=None) # 使用 OpenStreetMap 作为底图
ax.set_title('Typhoon Path with Precipitation (2022)')
plt.show()

# 绘制某一天的降水量分布
selected_day_index = 0 # 选择 2022 年的第一天
precipitation_at_day = precipitation_data[selected_day_index, :, :]
```

---

---

```
plt.figure(figsize=(10, 6))
plt.imshow(precipitation_at_day, extent=(longitude.min(), longitude.max(), latitude.min(),
latitude.max()), origin='lower', aspect='auto')
plt.colorbar(label='Precipitation (mm)')
plt.title('Daily Precipitation on 2022-01-01')
plt.xlabel('Longitude (degrees)')
plt.ylabel('Latitude (degrees)')
plt.grid(True)
plt.show()

# 绘制台风路径及降水量分布图
fig, ax = plt.subplots(figsize=(12, 8))
cs = ax.contourf(longitude, latitude, precipitation_at_day, cmap='Blues')
gdf.plot(ax=ax, column='风速', cmap='viridis', markersize=50, legend=True, legend_kwds={'label': 'Wind Speed (m/s)'})
ctx.add_basemap(ax, crs=gdf.crs, source=CustomTileProvider(session), attribution=None) # 使用 OpenStreetMap 作为底图
cbar = fig.colorbar(cs, ax=ax)
cbar.set_label('Precipitation (mm)')
ax.set_title('Typhoon Path with Precipitation and Wind Speed (2022)')
plt.show()
```

---

## 10 预测贝碧嘉行进途中的中心风力及降水量代码

---

代码 10                   预测贝碧嘉行进途中的中心风力及降水量

---

```
import netCDF4 as nc
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import least_squares
from datetime import datetime, timedelta # 确保导入 datetime 模块
import geopandas as gpd
import contextily as ctx
import os # 导入 os 模块

# 设置绘图字体和正确显示负号
plt.rcParams['font.sans-serif'] = ['SimHei'] # 使用 SimHei 字体
plt.rcParams['axes.unicode_minus'] = False # 正确显示负号

# 设置 PROJ_LIB 环境变量
proj_lib_path = r'D:\Anaconda\envs\DL\Library\share\proj'
os.environ['PROJ_LIB'] = proj_lib_path

# 读取贝碧嘉台风数据
filename = '问题三预测数据.xlsx'
typhoon_data_bebinca = pd.read_excel(filename)
```

---

---

```
# 将时间格式 YYYYMMDDHH 转换为 datetime 格式
typhoon_data_bebinca['Time'] = pd.to_datetime(typhoon_data_bebinca['当前台风时间'].astype(str),
format='%Y%m%d%H')

# 过滤出 2024 年 9 月 16 日至 18 日的数据
date_filter      = (typhoon_data_bebinca['Time']      >=      datetime(2024,      9,      16))      &
(date_filter)    = (typhoon_data_bebinca['Time'] <= datetime(2024, 9, 18))
typhoon_data_filtered = typhoon_data_bebinca[date_filter]

# 提取数据
latitudes = typhoon_data_filtered['纬度'].values
longitudes = typhoon_data_filtered['经度'].values
P_c = typhoon_data_filtered['气压'].values # 单位为 hPa
V_max = typhoon_data_filtered['风速'].values # 最大风速 (单位: m/s)
lat = typhoon_data_filtered['纬度'].values # 纬度

# 定义空气密度 rho 和环境气压 P_0
rho = 1.15 # kg/m^3
P_0 = 1013 # 标准大气压 hPa

# 计算最大风速半径 R_m 和 Holland 参数 B
dp = 1010 - P_c
ln_Rmax = 3.015 - 6.291e-5 * (dp ** 2) + 0.0337 * lat
R_m = np.exp(ln_Rmax) # 最大风速半径 (单位: km)
B = 1.4 + 0.01 * dp - 0.0026 * R_m

# 使用 Holland 公式计算中心风速
V_max_calc = np.sqrt((B / rho) * ((P_0 - P_c) * 100) * np.exp(1)) # 将气压差转换为 Pa

# 使用之前拟合的降水公式计算降水量
# 拟合的参数: A, alpha, beta
A = 0.0070
alpha = 0.0106
beta = 0.9089

# 计算距离台风中心的降水量 (假设某固定距离, 例如 R_m)
D = R_m # 使用最大风速半径作为距离
V = V_max_calc # 使用 Holland 公式计算得到的风速
P_ref = 1013 # 标准大气压 hPa
P_obs = A * np.exp(-alpha * D) * (1 / V) * (1 / (P_c - P_ref)) * (V_max ** beta)

# 创建一个新的 DataFrame 来存放计算结果
result_data = typhoon_data_filtered.copy()
result_data['Calculated_Wind_Speed'] = V_max_calc
```

---

---

```
result_data['Predicted_Precipitation'] = P_obs

# 保存新的 DataFrame 到 CSV 文件
output_filename = 'typhoon_bebinca_prediction_results.csv'
result_data.to_csv(output_filename, index=False, encoding='utf-8-sig')
print(f'New data has been saved to {output_filename}')

# 绘制结果可视化
plt.figure(figsize=(10, 10))

plt.subplot(2, 1, 1)
plt.plot(typhoon_data_filtered['Time'], V_max, 'r', linewidth=2, label='Observed Wind Speed')
plt.plot(typhoon_data_filtered['Time'], V_max_calc, 'b--', linewidth=2, label='Calculated Wind Speed')
plt.xlabel('Time')
plt.ylabel('Wind Speed (m/s)')
plt.title('Typhoon Bebinca Central Wind Speed (2024-09-16 to 2024-09-18)')
plt.grid(True)
plt.legend()

plt.subplot(2, 1, 2)
plt.plot(typhoon_data_filtered['Time'], P_obs, 'b', linewidth=2, label='Predicted Precipitation')
plt.xlabel('Time')
plt.ylabel('Precipitation (mm)')
plt.title('Predicted Precipitation for Typhoon Bebinca (2024-09-16 to 2024-09-18)')
plt.grid(True)
plt.legend()

plt.tight_layout()
plt.show()

# 创建 GeoDataFrame 用于地图绘制
gdf = gpd.GeoDataFrame(
    typhoon_data_filtered,
    geometry=gpd.points_from_xy(typhoon_data_filtered['经度'], typhoon_data_filtered['纬度']),
    crs="EPSG:4326" # WGS84 坐标系
)
# 地图显示台风路径
fig, ax = plt.subplots(figsize=(10, 6))
gdf.plot(ax=ax, color='blue', markersize=50, label='Typhoon Path', alpha=0.7)
ctx.add_basemap(ax, crs=gdf.crs, source=ctx.providers.OpenStreetMap.Mapnik)
OpenStreetMap 作为底图
ax.set_xlabel('Longitude')
ax.set_ylabel('Latitude')
ax.set_title('Typhoon Bebinca Path (2024-09-16 to 2024-09-18)')
ax.grid(True)
```

---

---

```
ax.legend()  
plt.show()
```

---