稻飞虱预测代码实验报告

本项目旨在通过多种气象数据预测稻飞虱的首迁期及其虫量。以下是项目过程中遇到的主要问题及相应解决方案。本实验使用python语言进行实现。

项目存在的问题和解决方法

数据量不足

项目初期面临数据量不足的挑战。我们需要对17个区域分别进行预测，但每个区域仅有约1000条数据，这对于机器学习模型来说难以实现准确预测。
为了解决数据量不足的问题，我们将17个区域划分为4个稻区，使每个稻区的数据量增加至约6000条，从而满足机器学习模型训练的基本需求。该策略不仅显著提升了样本量，还有效减少了区域间的差异性对预测结果的影响。

预测目标不便于预测

另一个问题是，稻飞虱的首迁期数据量极其稀少。从2000年至今，仅记录了23个首迁期数据，这对于机器学习模型来说几乎不可用。为了解决这个问题，我们将预测目标从首迁期调整为首迁期与基准日期的差值，这一转变大幅增加了可用数据量，提升了模型的学习能力。调整后，预测的方差从约20万显著降低至3000左右。

噪声的影响使预测值出现了巨大波动

在解决数据量和目标问题后，我们预计预测误差会显著下降。然而，模型的预测值与真实值之间的误差方差仍然保持在1000左右。即便剔除异常值，方差依然徘徊在500左右。
通过分析输入数据与预测目标的相关性，我们采用特征选择方法强化输入变量。高相关性特征得以保留，而低相关性甚至负相关特征被剔除。优化后，不同模型的预测误差方差降至1以下，显著提升了预测性能。

代码的具体实现

预先导入的库：

1. pandas ： Pandas 是一个强大的数据处理和分析库，特别擅长处理表格型数据（如Excel、CSV文件等）。
   导入方法：`import pandas as pd`
2. numpy：NumPy 是 Python 的科学计算基础库，提供多维数组对象（ndarray）和一系列高效的数学运算工具。它是许多数据科学库（如 Pandas、Scikit-learn）的底层依赖。 NumPy 的数组计算比 Python 原生列表快得多，适合处理大规模数值数据。
   导入方法：`import numpy as np`
3. scikit-learn
   - Scikit-learn 是一个机器学习库，提供了许多常用的机器学习算法和工具，包括分类、回归、聚类、降维等。它还提供数据预处理、模型评估和超参数调优的方法。
     因为这个方法库没有预装在python3的库中，需要从外部导入,在终端中输入：
     `pip insatll Scikit-learn`
   - sklearn有很多常用的方法，例如数据分割与标准化(如 train_test_split()，StandardScaler)，机器学习模型(如 LinearRegression，RandomForestClassifier),模型评估(如 cross_val_score()，confusion_matrix())

- 导入方法:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
```

4. xgboost
   - XGBoost 是一种基于梯度提升框架的高效机器学习库，以其高性能和灵活性著称，广泛应用于各种比赛（如 Kaggle）和生产环境。它特别适合处理结构化数据，并支持并行处理、大规模数据的分布式训练。
   - XGBoost库的安装：

```
pip install -i https://pypi.tuna.tsinghua.edu.cn/simple xgboost
```

代码展示：

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, r2_score
```

数据的导入

本次实验中使用了四大稻区的综合库，相比于原始数据，增加了日期列和日期-首迁期连列作为标签名。、 在下面展示的代码中，我们选取“combined_华南.xlsx”文件做为示范。除了“”combined_华南.xlsx”，我们还有其他的三个需要分别预测。

```
# 导入数据
df = pd.read_excel('dealed_data(xlsx)/combined_华南.xlsx')

# 时间转换
df['首迁期'] = pd.to_datetime(df['首迁期'])
df['首迁期'] = df['首迁期'].apply(lambda x: x.timestamp())
df['日期'] = df['日期'].apply(lambda x: x.timestamp())
```

由于日期和首迁期这两个标签的格式较为特殊，不利于机器学习模型进行预测分析，因此我们需要将日期格式转换为时间戳。时间戳是指从格林威治标准时间1970年1月1日0时0分0秒（对应于北京时间1970年1月1日8时0

分0秒）起至当前时刻的总秒数。通过这种转换，可以更方便地将数据输入到机器学习模型中，从而提高模型的预测效果。

异常值的处理

由于数据中的异常值可能会对模型的预测性能产生不利影响，导致预测结果与真实情况之间出现偏差，因此有必要进行异常值检测，并将检测到的异常值予以删除，以确保模型能够基于更为准确的数据进行预测。

```
# 异常值检测与清洗
# 使用Z-score方法来检测异常值
from scipy import stats
z_scores = np.abs(stats.zscore(df.select_dtypes(include=[np.number])))  # 只对数值型数据计算Z-score
df_cleaned = df[(z_scores < 3).all(axis=1)]  # 删除Z-score大于3的异常值
```

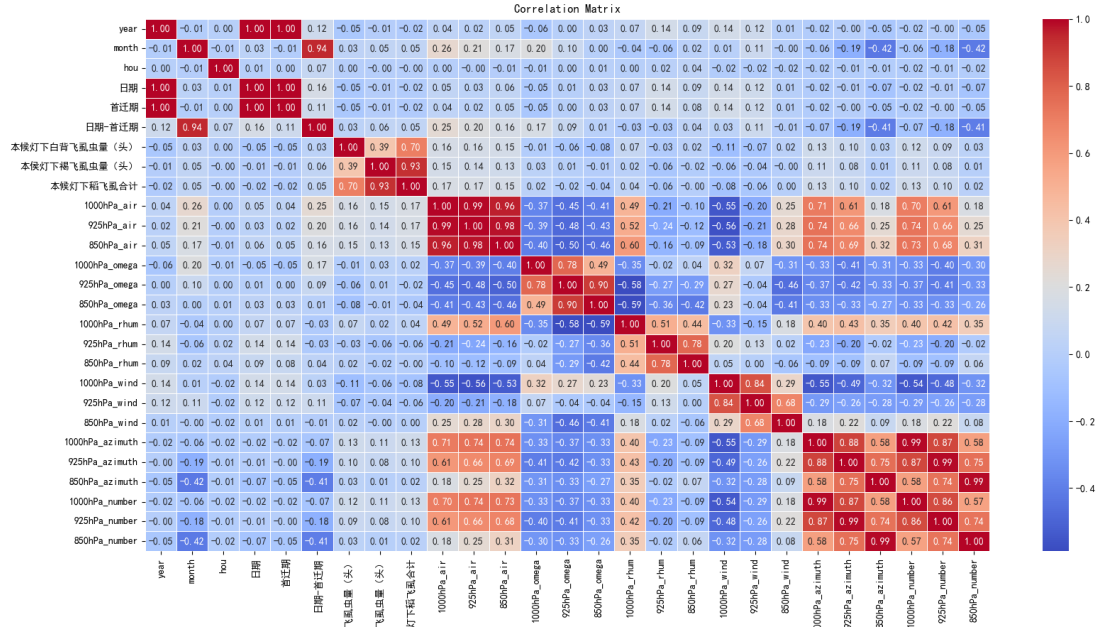
数据相关性的可视化

这段代码的目的是通过绘制一个热力图来直观地展示我们的特征值和目标值中各特征之间的相关性，帮助我们识别哪些特征之间可能存在强烈的线性关系。

```
# 相关性矩阵
correlation_matrix = df_cleaned.corr()

# 绘制热力图，查看各特征与目标变量的相关性
plt.figure(figsize=(20, 12))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f',
linewidths=0.5)
plt.title("Correlation Matrix")
plt.show()
```

通过上述代码和输入的数据，我们可以得到华南稻区的如下相关性数据图：



特征强化选择

在之前的实验中，由于未对特征相关性进行筛选，预测过程中混入了过多的噪声，导致预测结果偏差较大。因此，有必要进行特征选择，筛选出具有较强相关性的关键因素，去除相关性较弱甚至存在负相关的因素，从而提升预测的准确性，获得更加可靠的预测结果。

```
# 选择和目标变量"日期-首迁期"相关性较强的特征
target = "日期-首迁期"
cor_target = correlation_matrix[target].sort_values(ascending=False)
print(cor_target)

# 选择和目标变量相关性较高的特征
# 可以选择相关性高于某个阈值的特征，假设阈值为0.1
selected_features = cor_target[cor_target.abs() > 0.1].index.tolist()
print("Selected features:", selected_features)
```

在上述相关性矩阵中，已计算出每个特征与目标变量之间的相关性。在统计学中，相关性系数大于 0.1 表示变量之间存在轻微的线性关系，通常用于初步筛选，以保留可能对目标变量有一定影响的特征，避免遗漏潜在的重要因素。而相关性系数大于 0.3 则表明变量之间存在较明显的线性关系，此类特征对目标变量的影响更显著，适合用于进一步分析或模型训练。基于此，我们筛选出相关性大于 0.1 的特征，用于模型预测，以提高预测效果。

经过上述得到筛选，我们得到一下的相关性强的特征：

```
Selected features: ['日期-首迁期', 'month', '1000hPa_air', '925hPa_air',
'1000hPa_omega', '日期', '850hPa_air', 'year', '首迁期', '925hPa_wind',
'925hPa_number', '925hPa_azimuth', '850hPa_number', '850hPa_azimuth']

日期-首迁期              1.000000
```

```
month              0.944746
1000hPa_air        0.245740
925hPa_air         0.197176
1000hPa_omega      0.167131
日期                 0.159597
850hPa_air         0.159176
year               0.117981
首迁期                0.114867
925hPa_wind        0.110698
```

数据标准化

数据标准化和归一化是数据预处理中的重要步骤，用于将数据转换为适合模型输入的形式，从而提高建模效果。许多机器学习算法（如支持向量机、逻辑回归、KNN 等）依赖于特征的尺度，标准化能使这些算法更快收敛并获得更好的结果。

```
# 特征和目标变量
X = df_cleaned[selected_features].drop([target], axis=1)
y = df_cleaned[target] / 365

# 特征标准化
ss = StandardScaler()
X_std = ss.fit_transform(X)
```

对数据集进行分割

对数据集进行分割的主要目的是为了评估模型的泛化能力，确保模型在未见过的数据上表现良好。通过 train_test_split 方法，可以将数据集划分为训练集和测试集。

```
X_train, X_test, y_train, y_test = train_test_split(X_std, y, test_size=0.2,
random_state=0)
```

随机森林模型预测

```
# 随机森林回归
rf = RandomForestRegressor(n_estimators=150, max_depth=10, random_state=0)
rf.fit(X_train, y_train)

y_pred_rf = rf.predict(X_test)
print("Random Forest R2:", r2_score(y_test, y_pred_rf))
print("Random Forest MSE:", mean_squared_error(y_test, y_pred_rf))
print('\n')
```

得到的结果：

```
Random Forest R2: 0.9759953992818229
Random Forest MSE: 0.0020019855095925183
```

其中R2为模型对目标变量的解释能力，取值范围为 [0, 1]，如果r2值小于0，就说明模型的拟合效果比简单使用目标变量的均值预测还要差。 MSE模型的平均预测误差，MSE 是回归模型的一个常用误差指标，值越小，说明预测结果越接近真实值。

支持向量机模型预测

```
# 支持向量机回归
svm_model = SVR(kernel='poly', degree=3, gamma='auto')
svm_model.fit(X_train, y_train)

y_pred_svm = svm_model.predict(X_test)
print("SVM R2:", r2_score(y_test, y_pred_svm))
print("SVM MSE:", mean_squared_error(y_test, y_pred_svm))
print('\n')
```

得到的结果：

```
SVM R2: 0.8535032440738298
SVM MSE: 0.012217840488570135
```

XGB模型预测

```
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, r2_score

# 初始化XGBoost模型
xgb_model = XGBRegressor(n_estimators=100, learning_rate=0.1, max_depth=5)
xgb_model.fit(X_train, y_train)

# 预测并评估
y_pred_xgb = xgb_model.predict(X_test)
print("XGBoost R2:", r2_score(y_test, y_pred_xgb))
print("XGBoost MSE:", mean_squared_error(y_test, y_pred_xgb))
print('\n')
```

得到结果：

```
XGBoost R2: 0.9895721311893507
XGBoost MSE: 0.0008696850449607012
```

KNN模型预测

```
# 初始化KNN模型
knn_model = KNeighborsRegressor(n_neighbors=5)
knn_model.fit(X_train, y_train)

# 预测并评估
y_pred_knn = knn_model.predict(X_test)
print("KNN R2:", r2_score(y_test, y_pred_knn))
print("KNN MSE:", mean_squared_error(y_test, y_pred_knn))
print('\n')
```

得到结果：

```
KNN R2: 0.8922162426554712
KNN MSE: 0.008989173488304898
```

初始线性回归模型预测

```
# 初始化线性回归模型
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)

# 预测并评估
y_pred_lr = lr_model.predict(X_test)
print("Linear Regression R2:", r2_score(y_test, y_pred_lr))
print("Linear Regression MSE:", mean_squared_error(y_test, y_pred_lr))
print('\n')
```

得到结果：

```
Linear Regression R2: 1.0
Linear Regression MSE: 2.4440071094281747e-29
```

线性回归拟合多项式回归模型预测

```
### 多层感知机模型预测
```python
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```
```

```
# 扩展特征为多项式duo
poly = PolynomialFeatures(degree=3)
X_poly = poly.fit_transform(X_train)

# 使用线性回归拟合多项式回归
poly_model = LinearRegression()
poly_model.fit(X_poly, y_train)

# 对测试集进行预测
X_test_poly = poly.transform(X_test)
y_pred_poly = poly_model.predict(X_test_poly)

print("Polynomial Regression R2:", r2_score(y_test, y_pred_poly))
print("Polynomial Regression MSE:", mean_squared_error(y_test, y_pred_poly))
print('\n')
```

得到结果：

```
Polynomial Regression R2: 1.0
Polynomial Regression MSE: 1.1274443190064055e-26
```

初始化多层感知机回归模型

```
mlp_model = MLPRegressor(hidden_layer_sizes=(100, 100), max_iter=1000,
random_state=0)
mlp_model.fit(X_train, y_train)

# 预测并评估
y_pred_mlp = mlp_model.predict(X_test)
print("MLP R2:", r2_score(y_test, y_pred_mlp))
print("MLP MSE:", mean_squared_error(y_test, y_pred_mlp))
```

得到结果：

```
MLP R2: 0.9275893202130275
MLP MSE: 0.006039056153243568
```