天气预测模型设计报告

小组成员：侯传伟 2023210558 信研 2306

张书奇 2023210575 信研 2306

张天雨 2023210576 信研 2306

一、小组成员分工

张天雨：线性回归模型、RNN 模型代码、大部分 Web 页面代码、报告撰写

张书奇：RFR 模型代码、小部分 Web 网页代码、报告撰写

侯传伟：LSTM 模型代码、小部分 Web 网页代码、报告撰写

二、数据来源

我们小组的数据来源是 2345 天气（https://tianqi.2345.com），数据获取方式：使用 Python 爬虫爬取北京市三年（2019-2021）的每日天气数据（部分数据请见图 1）。再将其转换为 csv 文件方便后续使用。

日期	最高温度	最低温度	天气	风力风向	空气质量
2019/1/1	1	-10	晴~多云	西北风1级	56
2019/1/2	1	-9	多云	东北风1级	60
2019/1/3	2	-7	霾	东北风1级	165
2019/1/4	2	-7	晴	西北风2级	50
2019/1/5	0	-8	多云	东北风2级	29
2019/1/6	3	-7	多云	东南风1级	84
2019/1/7	2	-7	多云	西北风2级	61
2019/1/8	1	-10	晴	西北风2级	28
2019/1/9	3	-9	晴	西南风1级	60
2019/1/10	4	-7	晴~多云	西南风1级	105
2019/1/11	5	-7	霾	东北风1级	133
2019/1/12	6	-5	霾	西南风1级	229
2019/1/13	6	-7	晴	东北风1级	165

图 1 部分数据示例

在实际使用时，使用了**日期、最高温度、最低温度**三个数据。

三、所选用模型和训练过程

我们小组一共训练了四个模型，分别为线性回归模型（Linear）、循环神经网络（RNN）、随机森林模型（RFR）和长短期记忆模型（LSTM）。

其中，Linear、RNN、LSTM 使用 Pytorch 完成，RFR 使用 Numpy 完成

这些模型在训练前共有的步骤有：

①处理数据；

②定义所使用的模型；

③使用该模型训练所给的训练集；

④使用训练完成的模型在测试集中进行训练，并将预测的结果打印出来。

在以下各模型单独的章节中除非有特殊情况，否则对步骤①和步骤④不再赘述，着重叙述步骤②和步骤③。

3.1 线性回归模型

（1）线性回归模型的定义如图 2 所示，该模型有三个全连接层（FC）和两个 ReLU 层。

模型定义：

```python
# 定义温度预测模型
class TemperaturePredictor(nn.Module):
    def __init__(self, input_size, hidden_size1, hidden_size2, output_size):
        super(TemperaturePredictor, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size1)
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size1, hidden_size2)
        self.relu2 = nn.ReLU()
        self.fc3 = nn.Linear(hidden_size2, output_size)

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu1(x)
        x = self.fc2(x)
        x = self.relu2(x)
        x = self.fc3(x)
        return x
```

图 2 线性回归模型定义

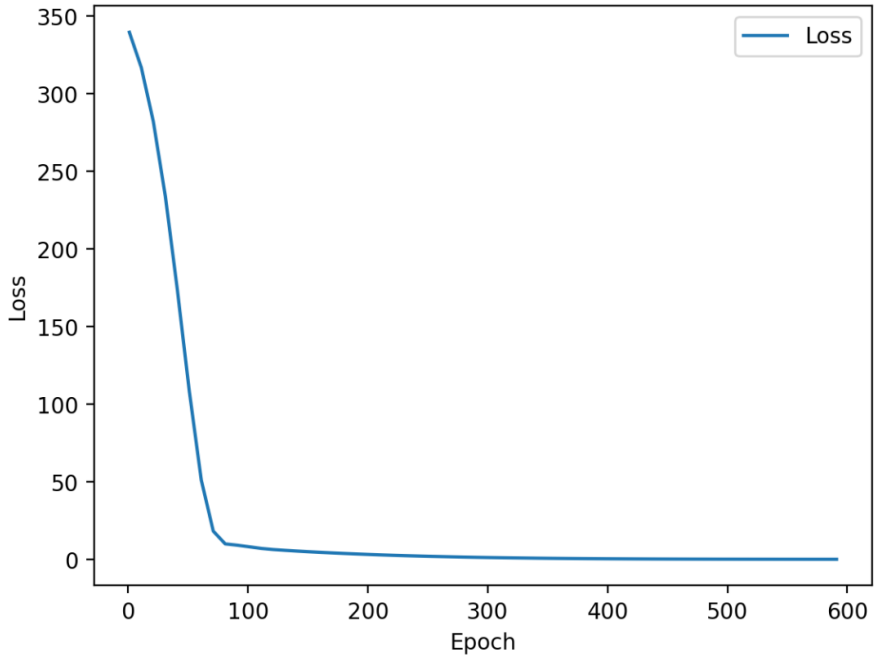
（2）使用该模型进行训练的步骤如图 3 所示，我们一共训练了 600 轮，并在每轮打印了损失（Loss）。

模型训练：

```python
# 训练温度预测模型
num_epochs = 600
for epoch in range(num_epochs):
    temp_outputs = temperature_model(X_train_tensor)
    temp_loss = temp_criterion(temp_outputs, temp_y_train_tensor)

    temp_optimizer.zero_grad()
    temp_loss.backward()
    temp_optimizer.step()

    if (epoch + 1) % 10 == 0:
        print(f'Temperature Model - Epoch [{epoch+1}/{num_epochs}], Loss: {temp_loss.item():.4f}')
```

图 3 使用线性回归模型训练所给数据集

（3）模型训练 Loss：



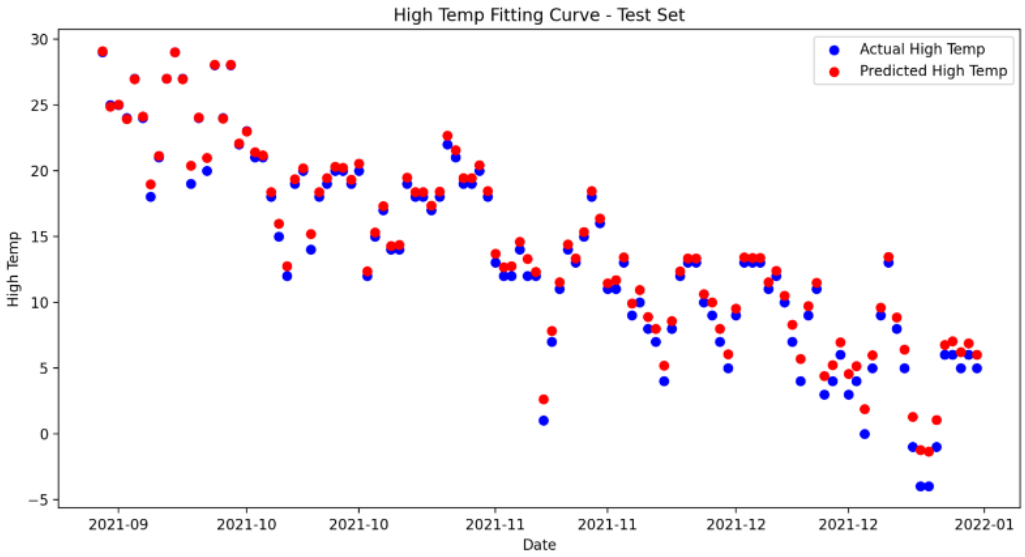
（4）测试结果：

Test Temperature Mean Squared Error: 0.4847

模型测试结果

	Date	Predicted High Temp	Predicted Low Temp	Actual High Temp	Actual Low Temp
0	2021-09-13	29.0592	20.2138	29	20
1	2021-09-14	24.8729	17.0931	25	17
2	2021-09-15	25.013	19.1898	25	19
3	2021-09-16	23.9233	17.1189	24	17
4	2021-09-17	26.9503	17.1015	27	17
5	2021-09-18	24.1309	19.1996	24	19
6	2021-09-19	18.9714	16.0857	18	16
7	2021-09-20	21.1363	15.1222	21	15
8	2021-09-21	26.9816	14.1905	27	14
9	2021-09-22	29.0143	17.1633	29	17

拟合曲线



可以看出效果不错

3.2 循环神经网络（RNN）

（1）循环神经网络需要对数据做特殊的处理，即将数据转换为模型可用的序列数据。步骤如图 4 所示。

数据预处理：

```python
# 将数据转换为模型可用的序列数据

def create_sequences(data, seq_length):
    sequences = []
    targets = []
    dates = []
    for i in range(len(data) - seq_length):
        seq = data[i:i + seq_length]
        target = data[i + seq_length:i + seq_length + 1].flatten()
        date_index = i + seq_length

        date = data[date_index,0]
        sequences.append(seq)
        targets.append(target)
        dates.append(date)
    return torch.tensor(sequences), torch.tensor(targets), dates

seq_length = 15

X_train, y_train, train_dates = create_sequences(train_data, seq_length)
X_test, y_test, test_dates = create_sequences(test_data, seq_length)
```

图 4 数据序列化步骤

在将数据序列化之后定义 RNN 模型，其定义如图 5 所示，我们定义了一个 3 层，具有 128 个隐藏单元的 RNN。

```python
# 定义RNN模型
class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, num_layers):
        super(RNN, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.rnn = nn.RNN(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)
        out, _ = self.rnn(x, h0)
        out = self.fc(out[:, -1, :])
        return out

# 定义模型参数
input_size = len(target_cols)
hidden_size = 128
output_size = len(target_cols)
num_layers = 3
```

图 5 RNN 定义

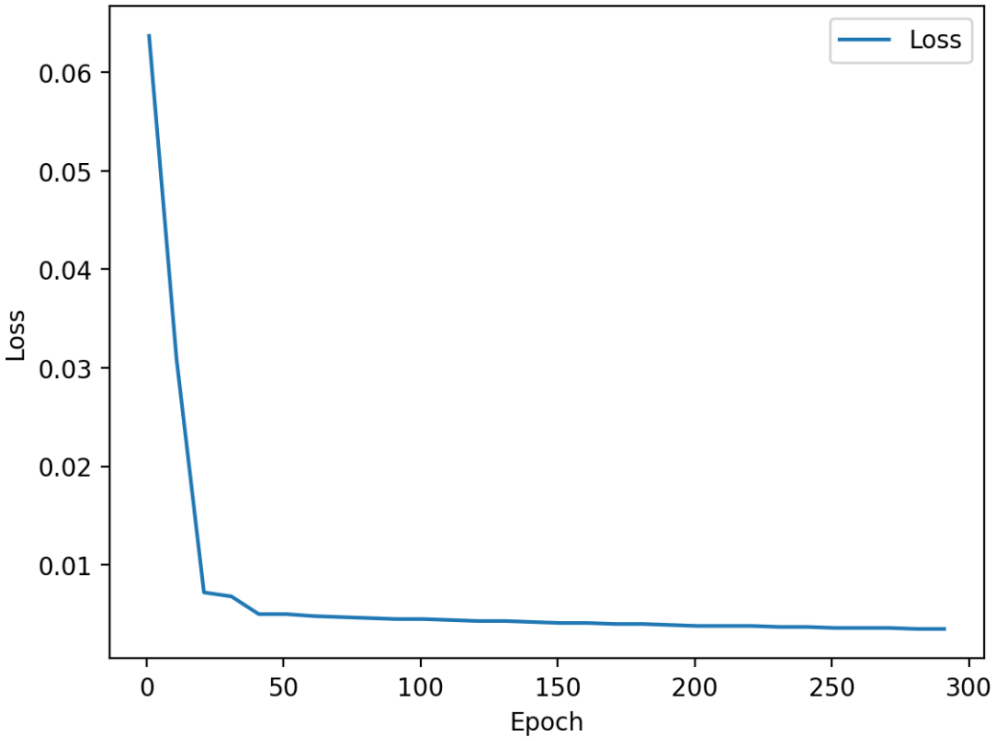
（2）而训练 RNN 的过程见图 6，我们共训练了 300 轮，并打印了每一轮的损失函数。

模型训练：

```python
# 训练模型
num_epochs = 300
for epoch in range(num_epochs):
    model.train()
    outputs = model(X_train.float())
    optimizer.zero_grad()
    loss = criterion(outputs, y_train.float())
    loss.backward()
    optimizer.step()

    if (epoch+1) % 10 == 0:
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')
```

图 6　RNN 训练过程

（3）模型训练 Loss：



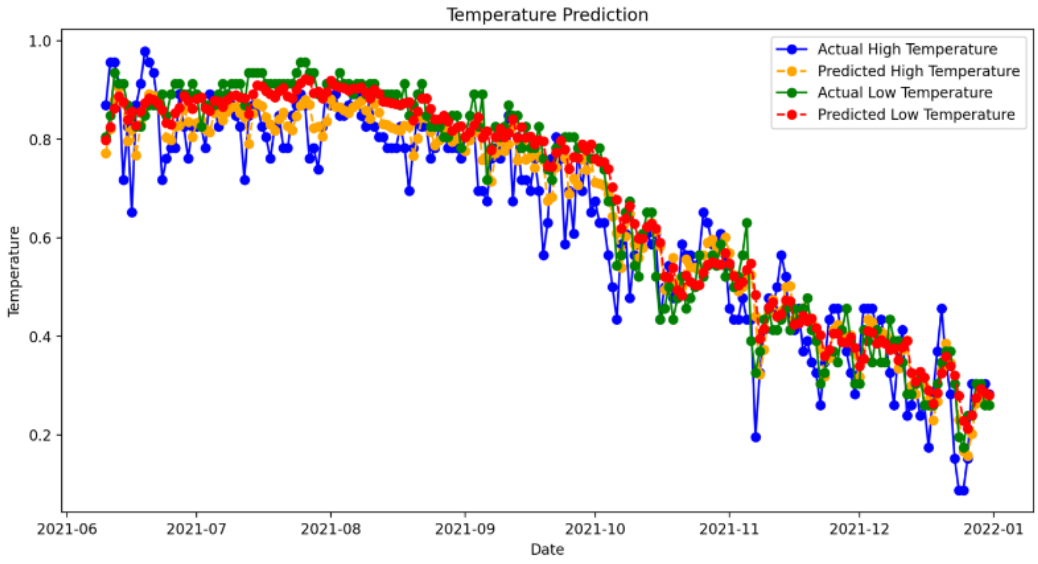
（4）测试结果：

Mean Squared Error on Test Data: 0.0034

模型测试结果

Date	Predicted High Temp	Predicted Low Temp	Actual High Temp	Actual Low Temp
2021-06-10	0.7714	0.798	0.8696	0.8043
2021-06-11	0.821	0.8247	0.9565	0.8478
2021-06-12	0.8801	0.8627	0.9565	0.9348
2021-06-13	0.9045	0.8871	0.913	0.913
2021-06-14	0.8733	0.8756	0.7174	0.913
2021-06-15	0.796	0.8395	0.8261	0.8696
2021-06-16	0.8199	0.8563	0.6522	0.8478
2021-06-17	0.7681	0.8276	0.8696	0.8261
2021-06-18	0.8299	0.8568	0.913	0.8261
2021-06-19	0.8651	0.8732	0.9783	0.8478

拟合曲线



可以看出效果也不错

3.3 随机森林模型（RFR）

（1）随机森林模型的定义见图 7-图 9，我们定义了一个大小为

10 的随机森林，并定义了训练函数和预测函数。在训练时对每个树创建一个决策树回归器，并在预测时取每个回归器的预测值进行平均得到预测数组

模型定义：

```
class RandomForestRegressor:
    """
    随机森林回归器
    """

    def __init__(self, n_estimators=10, random_state=0):
        # 随机森林的大小
        self.n_estimators = n_estimators
        # 随机森林的随机种子
        self.random_state = random_state
        # 决策树数组
        self.trees = []
```

图 7　RFR 的初始化函数

（2）模型训练：

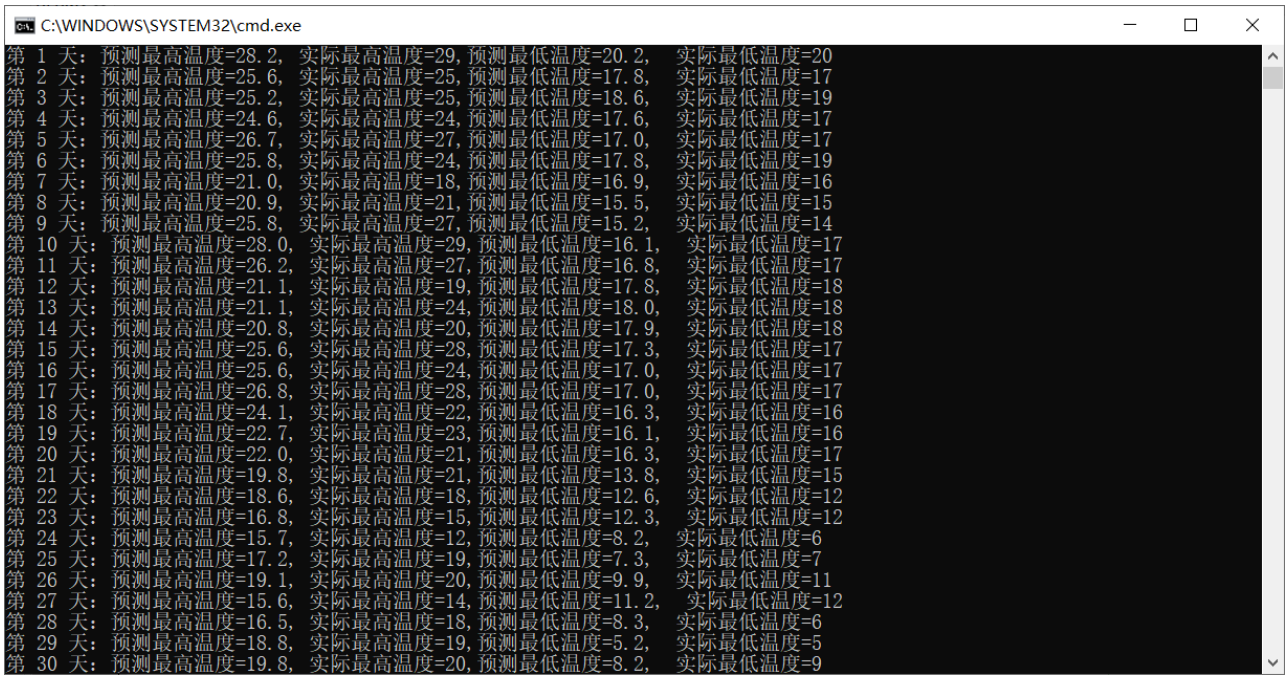
```
def fit(self, X, y):
    """
    随机森林回归器拟合
    """
    n = X.shape[0]
    rs = np.random.RandomState(self.random_state)
    for i in range(self.n_estimators):
        # 创建决策树回归器
        dt = DecisionTreeRegressor(random_state=rs.randint(np.iinfo(np.int32).max), max_features=1.0)
        # 根据随机生成的权重，拟合数据集
        indices = rs.randint(0, n, n)
        dt.fit(X.iloc[indices], y.iloc[indices])
        self.trees.append(dt)
```

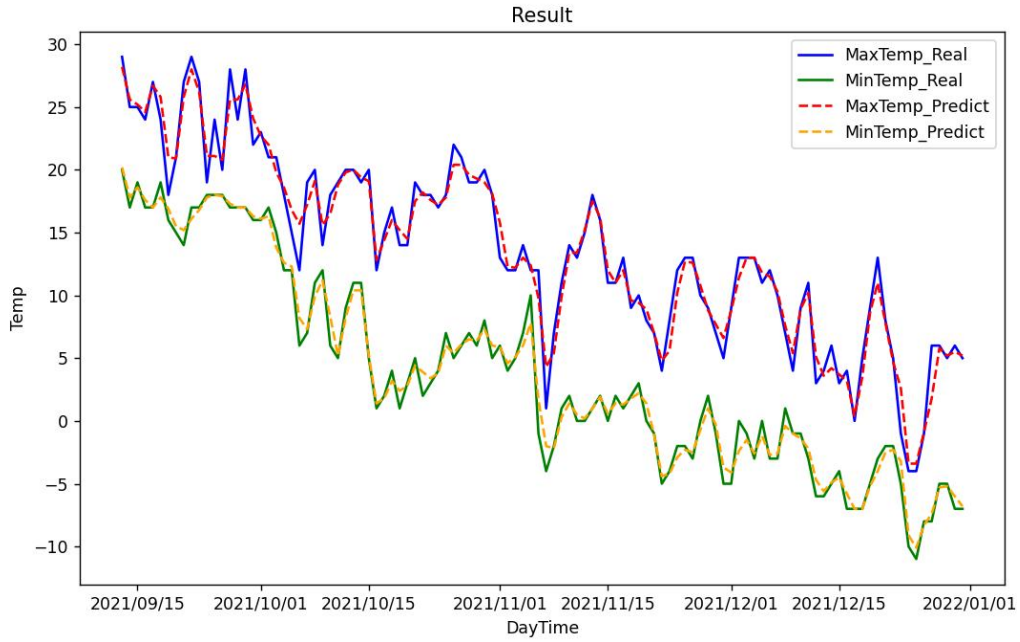
图 8　RFR 的训练函数

```python
def predict(self, X):
    """
    随机森林回归器预测
    """
    # 预测结果
    ys = np.zeros((X.shape[0], self.n_estimators, 2))
    for i in range(self.n_estimators):
        # 决策树回归器
        dt = self.trees[i]
        # 依次预测结果
        ys[:, i, :] = dt.predict(X)
    # 预测结果取平均
    predictions = np.mean(ys, axis=1)
    return predictions
```

图 9 RFR 的预测函数

（3）测试结果：





可以看出效果很好

3.4 长短期记忆模型（LSTM）

（1）长短期记忆模型的定义见图 10，我们定义了一个有两个 LSTM 层，一个 ReLU 层和两个全连接层的 LSTM 模型。隐藏神经元个数为 150。

模型定义：

```
class WeatherLSTM(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, num_layers):
        super(WeatherLSTM, self).__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers=num_layers, batch_first=True)
        self.relu = nn.ReLU()
        self.fc1 = nn.Linear(hidden_size, 50)
        self.fc2 = nn.Linear(50, output_size)

    def forward(self, x):
        out, _ = self.lstm(x)
        out = self.relu(out[:, -1, :])
        out = self.fc1(out)
        out = self.fc2(out)
        return out

input_size = 3
hidden_size = 150
output_size = 2
num_layers = 2
model = WeatherLSTM(input_size, hidden_size, output_size, num_layers)
```

图 10　LSTM 模型定义

（2）LSTM 模型的训练过程如图 11 所示，我们一共训练了 150

轮，并打印了每轮的损失。

模型训练：

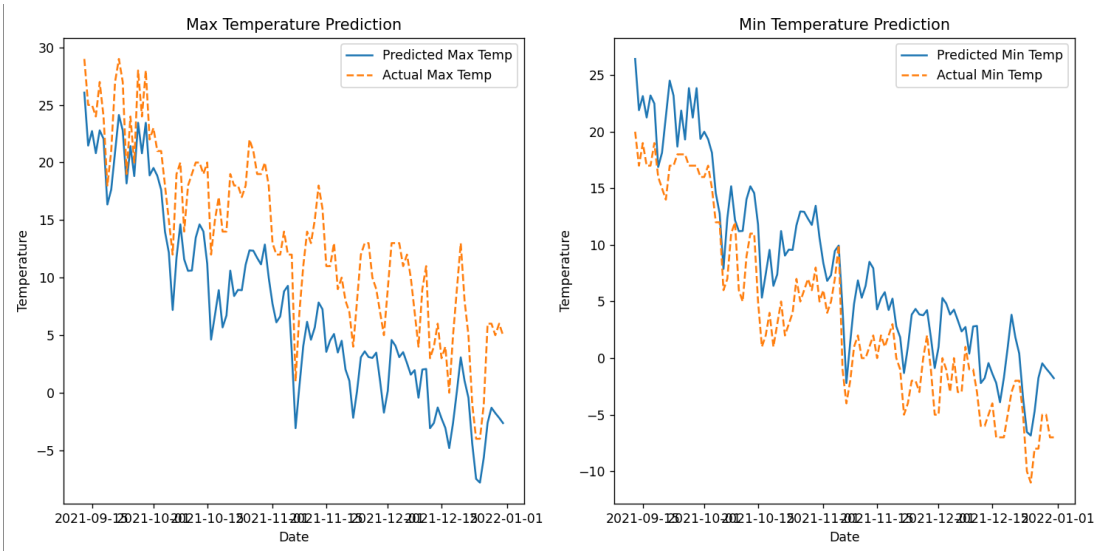
```python
# 训练模型
num_epochs = 150
for epoch in range(num_epochs):
    outputs = model(X_train)
    loss = criterion(outputs, y_train)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if (epoch + 1) % 10 == 0:
        print(f'Epoch [{epoch + 1}/{num_epochs}], Loss: {loss.item():.4f}')
```

图 11　LSTM 模型训练过程

（3）模型训练 Loss：

```
Epoch [10/150], Loss: 0.2957
Epoch [20/150], Loss: 0.1609
Epoch [30/150], Loss: 0.0394
Epoch [40/150], Loss: 0.0426
Epoch [50/150], Loss: 0.0392
Epoch [60/150], Loss: 0.0310
Epoch [70/150], Loss: 0.0269
Epoch [80/150], Loss: 0.0220
Epoch [90/150], Loss: 0.0166
Epoch [100/150], Loss: 0.0114
Epoch [110/150], Loss: 0.0069
Epoch [120/150], Loss: 0.0037
Epoch [130/150], Loss: 0.0021
Epoch [140/150], Loss: 0.0018
Epoch [150/150], Loss: 0.0017
```

（4）测试结果：



四、结果与比较

在四种训练完成的模型上对测试集进行预测，所得到的结果如图12-图 18 所示。在这些图中可以得到如下结论：

（1）四个模型对所给测试集给出的预测结果均与实际值较为贴合，有较好的预测效果。

（2）LSTM 模型与其他模型相比预测的误差较大，可能是该数据集不适合使用该模型。

```
Temperature Model - Epoch [110/600], Loss: 7.5007
Temperature Model - Epoch [120/600], Loss: 6.5491
Temperature Model - Epoch [130/600], Loss: 5.9117
Temperature Model - Epoch [140/600], Loss: 5.4225
Temperature Model - Epoch [150/600], Loss: 4.9704
Temperature Model - Epoch [160/600], Loss: 4.5748
Temperature Model - Epoch [170/600], Loss: 4.2146
Temperature Model - Epoch [180/600], Loss: 3.8823
Temperature Model - Epoch [190/600], Loss: 3.5750
Temperature Model - Epoch [200/600], Loss: 3.2898
Temperature Model - Epoch [210/600], Loss: 3.0264
Temperature Model - Epoch [220/600], Loss: 2.7833
Temperature Model - Epoch [230/600], Loss: 2.5574
Temperature Model - Epoch [240/600], Loss: 2.3476
Temperature Model - Epoch [250/600], Loss: 2.1542
...
   Predicted: High Temp: 28.81, Low Temp: 17.19
   Actual:    High Temp: 29.0, Low Temp: 17.0
```

图 12　线性回归模型训练过程中的损失

```
Epoch [100/300], Loss: 0.0045
Epoch [110/300], Loss: 0.0044
Epoch [120/300], Loss: 0.0043
Epoch [130/300], Loss: 0.0042
Epoch [140/300], Loss: 0.0041
Epoch [150/300], Loss: 0.0041
Epoch [160/300], Loss: 0.0040
Epoch [170/300], Loss: 0.0040
Epoch [180/300], Loss: 0.0039
Epoch [190/300], Loss: 0.0038
Epoch [200/300], Loss: 0.0038
Epoch [210/300], Loss: 0.0037
Epoch [220/300], Loss: 0.0037
Epoch [230/300], Loss: 0.0036
Epoch [240/300], Loss: 0.0036
Epoch [250/300], Loss: 0.0036
```

图 13　RNN 训练过程中的损失

```
Epoch [10/150], Loss: 0.3958
Epoch [20/150], Loss: 0.2267
Epoch [30/150], Loss: 0.0378
Epoch [40/150], Loss: 0.0455
Epoch [50/150], Loss: 0.0389
Epoch [60/150], Loss: 0.0293
Epoch [70/150], Loss: 0.0264
Epoch [80/150], Loss: 0.0225
Epoch [90/150], Loss: 0.0184
Epoch [100/150], Loss: 0.0144
Epoch [110/150], Loss: 0.0108
Epoch [120/150], Loss: 0.0075
Epoch [130/150], Loss: 0.0050
Epoch [140/150], Loss: 0.0033
Epoch [150/150], Loss: 0.0022
```

图 14　LSTM 训练过程中的损失

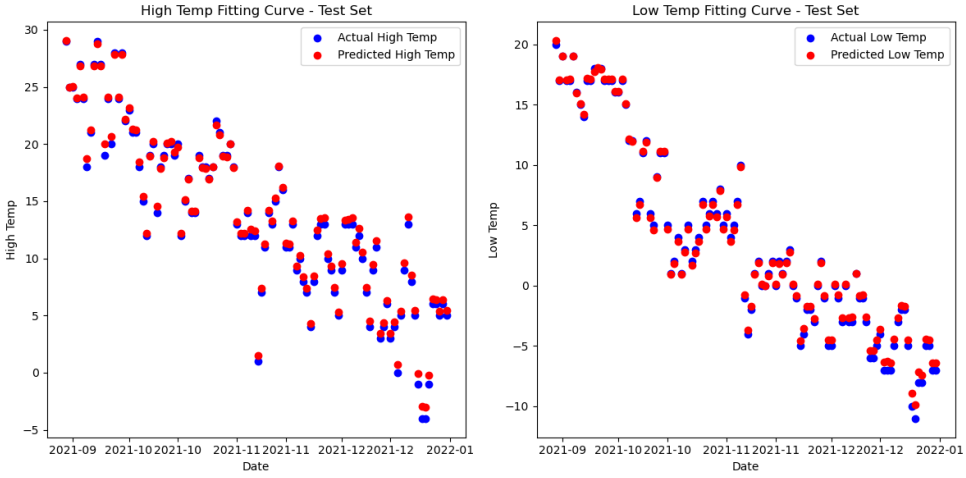


图 15　线性回归模型的预测值与真实值的比较

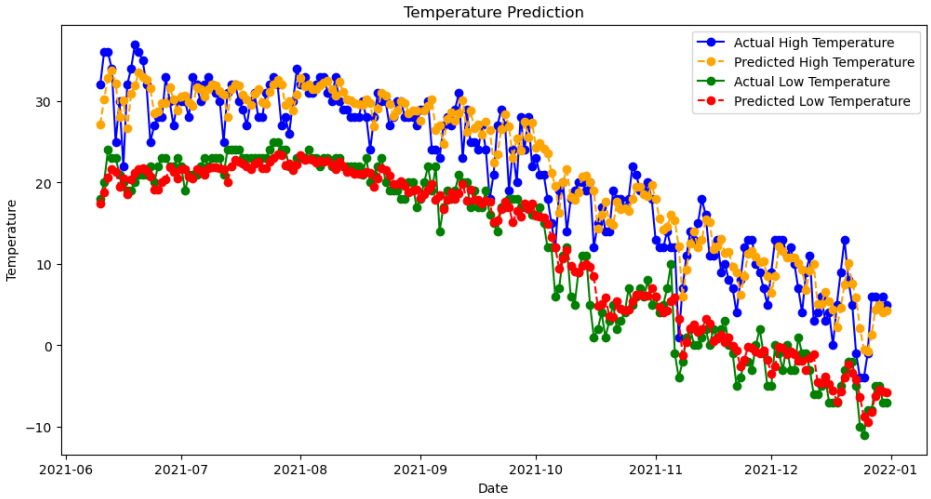


图 16　RNN 模型的预测值与真实值的比较

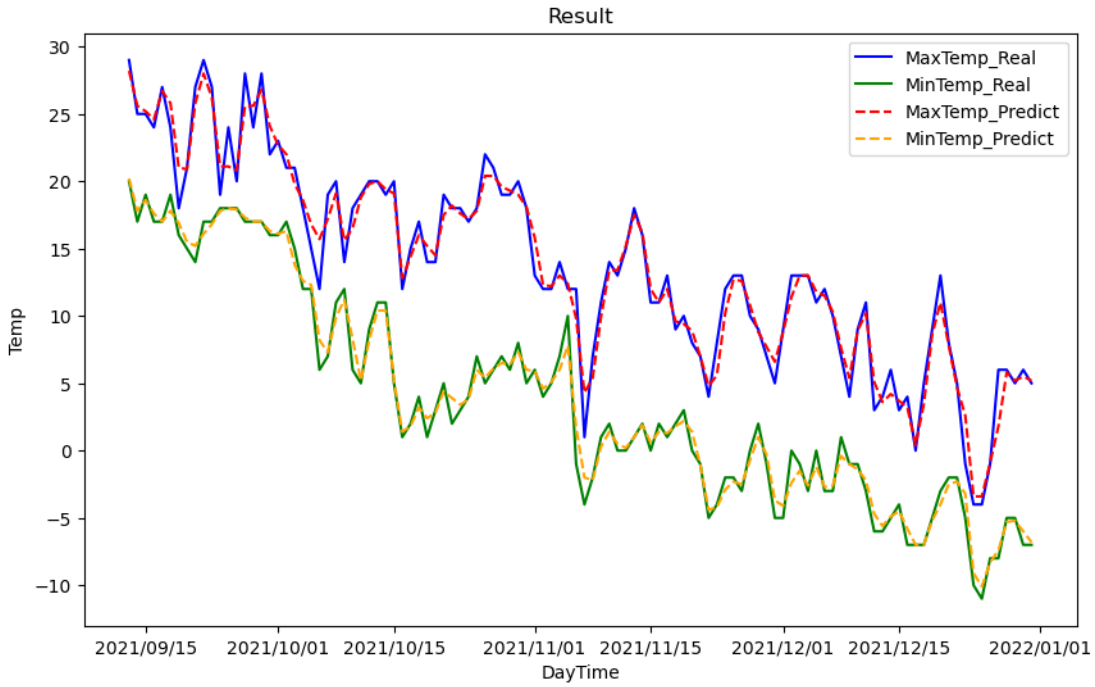


图 17　RFR 模型的预测值与真实值的比较

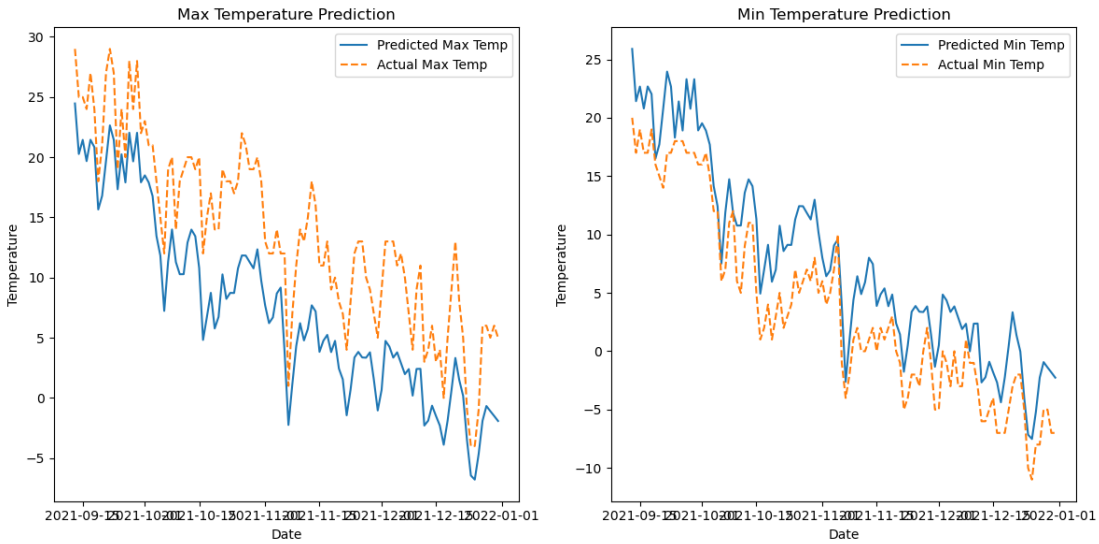


图 18　LSTM 模型的预测值与真实值的比较

我们还将线性回归模型和 RNN 模型集成在网页上，通过模型训练和模型调用可以得到模型训练的结果和预测的结果，以 RNN 模型为例，详见图 19 和图 20。模型训练结果中有预测值与实际值的对比图，已在图 16 中展示。

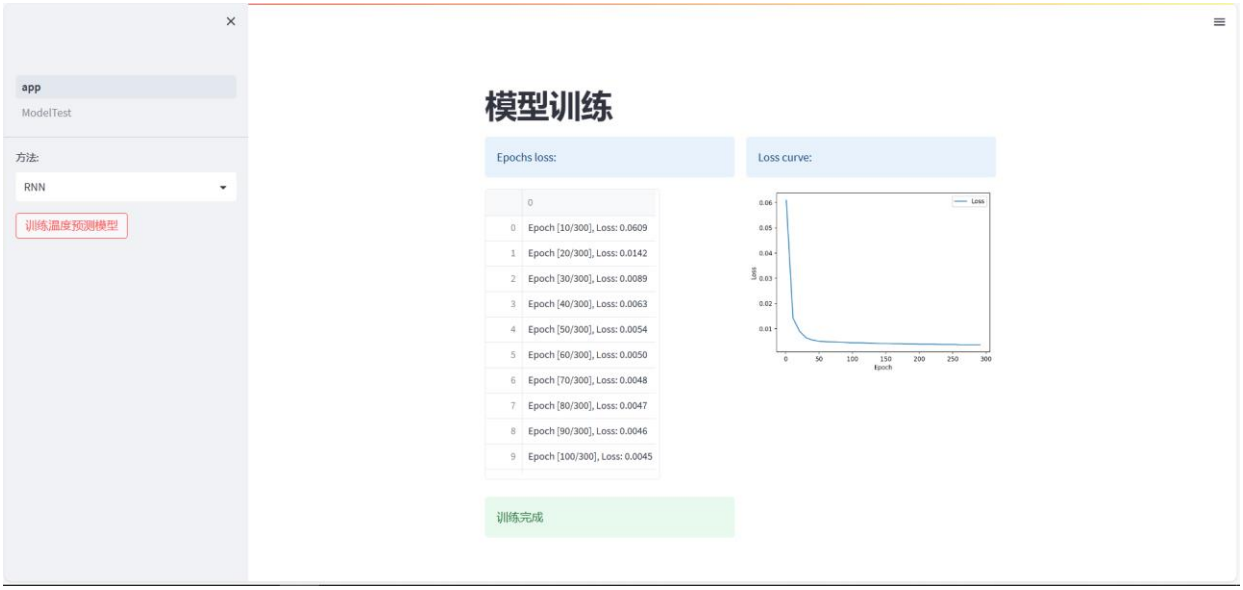


图 19　RNN 模型训练结果

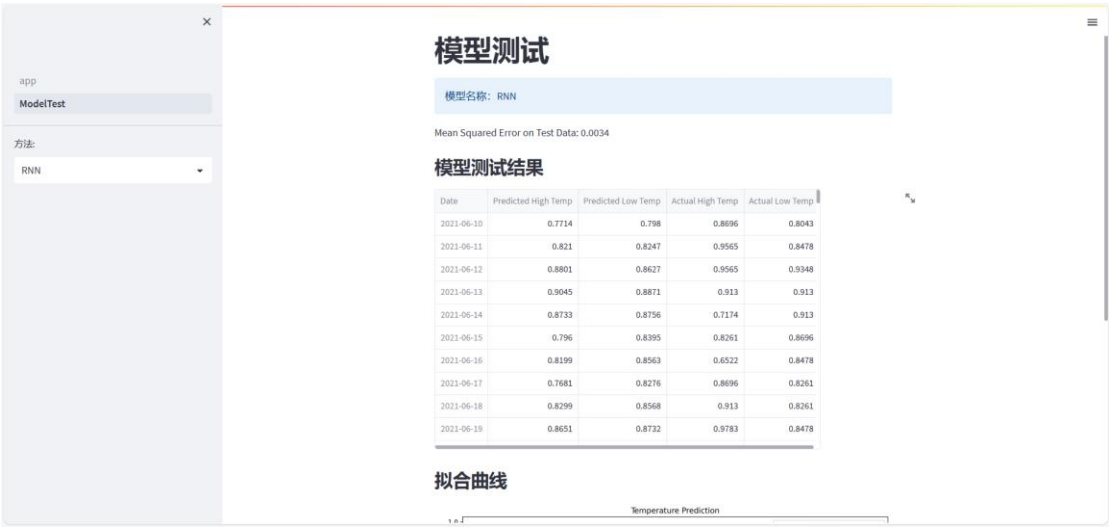


图 20　RNN 模型预测部分结果

五、总结

对于气温这样的参数来说，主要影响因素多种多样，仅靠日期来进行预测在实际应用时误差较大。如果要进一步开展工作，需要加入其他的参数（如所在地经纬度、气压、湿度等）重新训练模型并进行预测。而在模型选取上，我们选择的四种模型虽然已经可以代表几类典型的神经网络，但是还有补充的空间。也可以再选取几种别的模型进行比较。