Lab 06

NN and PCA

2023-10-02

Table of contents

# 1. Intro

Today we are going to build a simple model to simulate precipitation, conditional on temperature fields. The model is fairly simple:

1. Input: a temperature field
2. Project the (high-dimensional) temperature field onto a low-dimensional space, using PCA
3. Find the nearest neighbors to the projected temperature field, using -NN
4. Sample the precipitation field from the nearest neighbors

We will use the [ERA5](https://www.ecmwf.int/en/forecasts/datasets/reanalysis-datasets/era5) reanalysis dataset, which is a global dataset of atmospheric variables, including temperature and precipitation, for temperature. We use the same precipitation dataset as Lab 05.

## 1.1 Setup

Remember to git clone to your machine, then activate and instantiate the environment, as we have done in previous labs.

using Dates  
using MultivariateStats  
using Plots  
using NCDatasets  
using Unitful

# 2. Data

## 2.1 Precipitation

We start by loading the precipitation data. Create variables called precip\_time, precip\_lon, precip\_lat, and precip. precip should be a 3D array with appropriate units added. Remember to close the file when you have read the data

precip\_ds = NCDataset("data/precip.nc")  
# ...

## 2.2 Temperature

Next we load the temperature data

Create variables called temp\_time, temp\_lon, temp\_lat, and temp. temp should have appropriate units.

Make sure that temp\_time and precip\_time are the same! If so, you can rename this variable to time.

## 2.3 Split

We will split the data into training and testing sets (we will cover this idea in more detail later). The idea is to use the training set to build the model, and the testing set to evaluate the model.

We will use the last 10 years of data as the testing set, and the rest as the training set. Create variables called precip\_train, precip\_test, temp\_train, and temp\_test, time\_train, and time\_test.

# 3. Principal components

See the documentation [here](https://juliastats.org/MultivariateStats.jl/dev/pca/).

## 3.1 Preprocessing

First, we need to convert the temperature data to a matrix. You can use the reshape function to do this.

## 3.2 Fitting

Following the instructions provided in the documentation, fit the PCA model to the training data. Choose how many dimensions you are using and explain why you chose that number.

## 3.3 Visualizing

1. Plot the spatial patterns associated with the first two principal components. You’ll need to reshape the data back to its original shape.
2. Plot the time series of the first two principal components. Use your time variable on the axis of the plots to match the actual dates.
3. Make a scatter plot of the first two principal components, with the points colored by the precipitation value at that time. Define “the precipitation value” however you like – it may be the precipitation at a particular point or a spatial mean.

# 4. -NN

nearest neighbors is a simple prediction algorithm. We will use a resampling-based version of KNN. Here is a general outline of the algorithm with generic notation:

1. We want to make predictions about some new value,
2. Find the nearest neighbors to in the training data and their distances
3. Sample from the nearest neighbors, with probability proportional to the inverse of the distance

Here’s an example:

1. Our dataset is .
2. Our new value is .
3. The 3 nearest neighbors to are .
4. The corresponding distances are .
5. Thus, we return a sample from with associated probabilities .
   1. Instead of returning , , or , it is often advantageous to return the corresponding indices: 7, 8, or 6.

Write a function in Julia to perform generic nearest neighbor prediction. As the function signature implies, the first argument should be the training dataa, the second should be the new point to predict, and the third should be the number of neighbors to use. The function should return a tuple containing the indices of the neighbors and the associated probabilities.

function knn(X::AbstractMatrix, X\_i::AbstractVector, K::Int)::Tuple{Vector{Int}, Vector{AbstractFloat}}  
 # ...  
end

# 5. Combining

If we put these two pieces together, we have a simple yet powerful simulation tool. If we want to explore the possible precipitation for some new temperature field, we can:

1. Project the temperature field onto the low-dimensional space
2. Find the nearest neighbors to the projected temperature field
3. Sample the precipitation field from the nearest neighbors

Create a function to implement this model, then make predictions for three randomly chosen days in the testing set. Create appropriate visualizations (you’ll need to think about how to visualize probabilistic precipitation!) Once you are happy with your model, compare to the actual precipitation values for those days. What do you learn?