Lab 06

NN and PCA

2023-10-13

Table of contents

# 1. Intro

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

## 1.1 Why might this be useful?

Precipitation is notoriously difficult to forecast at high spatial and temporal resolution. Temperature, by contrast, is much smoother and often easier to forecast. We might imagine seeking to generate high-resolution precipitation fields from climate models that have substantial biases in representing precipitation.

## 1.2 Algorithm

For your reference, here is the full algorithm we will implement today. We’ll break this into bite-sized pieces for you, and then at the end we’ll implement the whole thing as a function in Julia.

1. Inputs
   * Past temperature fields
   * Current temperature field for which we are making predictions
   * Number of nearest neighbors
   * Number of principal components
2. Output
   * Sampled precipitation field
3. Steps
   1. Preprocessing:
      * Preprocess the temperature fields and as deemed necessary.
      * Document and explain the preprocessing steps and assumptions made.
   2. Dimensionality Reduction using PCA:
      * Apply Principal Component Analysis (PCA) to the past temperature fields .
      * Retain principal components. The choice of should be made and defended by the student.
   3. Find Nearest Neighbors:
      * Using the Euclidean distance metric, find the nearest neighbors to the projected current temperature field from step 2.
   4. Sample Precipitation Field:
      * For each of the nearest neighbors, compute the inverse distance as where is the distance to the neighbor.
        + This is sometimes called the “kernel” and there are other valid choices (from simple – like distance squared – to more complex)
      * Normalize the weights such that they sum to 1: .
      * Sample a precipitation field from the nearest neighbors with probability proportional to the normalized inverse distances .

## 1.3 Data

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. This data is available to enrolled students on Canvas as precip.nc and temp.nc.

## 1.4 Setup

Remember to git clone to your machine, then activate and instantiate the environment, as we have done in previous labs. **Do not use additional packages for this assignment,** though you are welcome to look at the documentation or implementation of other packages for nearest-neighbors methods.

using Dates  
using MultivariateStats  
using Plots  
using NCDatasets  
using StatsBase  
using Unitful  
  
Plots.default(; margin=4Plots.mm, size=(700, 400), linewidth=2)

# 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

|  |
| --- |
| Hint |
| Review the lab 05 solutions posted on Canvas to make sure you have the right syntax. Additionally, if the latitudes are flipped, you need to reverse them (also in lab 5 solutions!) |

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 create a variable called time that is equal to temp\_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.

### 2.3.1 Preprocess

Review the slides on PCA and decide whether you would like to preprocess the temperature data before applying PCA. Explain what you have done and why.

Next, we need to convert the temperature data to a matrix. You can use the reshape function to do this.[[1]](#footnote-31) Call your matrix temp\_mat.

Put all these steps into a function called preprocess that takes in:

* the raw field of temperature data (indexed lon, lat, time)
* the raw field of temperature data (indexed lon, lat, time) corresponding to the “reference period” over which climatology is returned
* any other inputs you would like (e.g., a vector of times for the reference and target periods)

and returns the matrix of preprocessed temperature data.

function preprocess(temp::Array{T, 3}, temp\_reference::Array{T, 3})::AbstractMatrix where T  
 # ...  
end

# 3. Principal components

We will use the implementation in MultivariateStats.jl to perform PCA. The key functions to know are:

1. **fit:** Use the syntax like M = fit(PCA, X\_train). The model will automatically choose how many prinicpal components to use but you can modify it using the principalratio and maxoutdim arguments.
   * To make plots of variance explained, the principalvars(M), var(M), and cumsum functions may be helpful 😉
   * Note: the documentation says: let (d, n) = size(X) be respectively the input dimension and the number of observations: this is the opposite of what we have been doing, so you will need to transpose the data.
2. **predict:** use the syntax X\_pca = predict(M, X). This will project the data X onto the PCA basis.
3. **projection:** returns a matrix where each column is a principal component loading vector and each row is a grid cell. . Use the syntax X\_pca = projection(M).

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

## 3.1 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. Create any plots to help inform this choice.

## 3.2 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.
   * You do not need to add map features. The best package for mapping is [GeoMakie](https://github.com/MakieOrg/GeoMakie.jl), which has a bit of a new plotting syntax and is still a little rough around the edges, though improving rapidly and great in many ways.
2. Plot the time series of the first two principal components. (We have the actual times – called time – so the axis should show the actual time)
3. Make a scatter plot of the first two principal components, where each day is a day. Color the points 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.
   * Use the syntax scatter(x, y, zcolor=z) to plot the points, where x and y are the first two principal components and z is the precipitation value.

# 4. -NN

nearest neighbors is a simple prediction algorithm. We will use a resampling-based version of KNN.

We have described the algorithm above; 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{Int,AbstractVector}  
 # ...  
end

In our example, X would be the principal component loading matrix for the training data, X\_i would be the principal component loading vector for the point we are trying to predict, and K would be the number of neighbors to use.

|  |
| --- |
| Hint |
| You may find the sample function from StatsBase to be helpful. You can define weights and use it for weighted sampling:  w = Weights(rand(10)) sample(1:10, w) |

# 5. Combining

If we put these two pieces together, we have a simple yet powerful simulation tool. Now we will implement the full algorithm as a function.

"""  
KNN resampling algorithm  
  
temp\_train: the training temperature data. This should be the raw field, not the prinipal components. Inside the function, convert to principal components using `n\_pca` principal components.   
temp\_test: the temperature data to predict. This should be the raw field, not the prinipal components. Inside the function, convert to principal components using `n\_pca` principal components.  
precip\_train: the training precipitation data.  
"""  
function predict\_knn(temp\_train, temp\_test, precip\_train; n\_pca::Int)  
 temp\_train = preprocess(temp\_train) # you have defined this function above  
 temp\_test = preprocess(temp\_test) # adjust arguments as needed  
  
 # fit the PCA model to the training data  
  
 # project the test data onto the PCA basis  
  
 # use the `knn` function for each point in the test data  
  
 # return a matrix of predictions  
  
end

Now, you can run this function on several days of data and plot the results. How can you visualize your results?

1. Access documentation by typing ?reshape in the REPL. [↑](#footnote-ref-31)