III. Model documentation and write-up

Information included in this section may be shared publicly with challenge results. You can respond to these questions in an e-mail or as an attached file. Please number your responses.

1. Who are you (mini-bio) and what do you do professionally? If you are on a team, please complete this block for each member of the team.

Filipp: I am a mathematician and data scientist. Since high school I have been interested in physical process modeling and in the optimization of the choice of parameters for numerical models. My research really took off after I was introduced to the PyTorch framework and started to use it. PyTorch allows us to optimize parameters of rather complex mathematical models. I have recently defended my PhD, a study of the structure of errors in numerical model forecasting.

Currently I work in a private company as a data scientist in the broad area of environmental sciences, mainly with the Earth system data. The questions which attract me most are those where machine learning can be used to understand physical processes. As I worked on the competition's problem, I realized that the Earth system data available to us is very heterogeneous and diverse. In the future, I would like to study heterogeneities of other parameters such as near-surface air temperature on a sub-kilometer scale.

Polina: I am a mathematician and a researcher, working in a broad area of dynamical systems. This is a branch of mathematics which aims to predict the future of physical systems using simplified, sometimes idealistic models. I particularly enjoy interdisciplinary problems where the methods and ideas from one field can be applied in another, seemingly unrelated at first sight. For instance, recently it became clear that the methods initially developed to tackle data science questions can be successfully applied to study quite abstract mathematical models.

2. What motivated you to compete in this challenge?

This challenge is related to the thoughtful consumption of renewable resources. I strongly believe that my solution will be useful for improving the water resource management. The given problem is very close to my scientific interests. I have a good theoretical background and practical skills in mathematics, statistics and data science.

3. High level summary of your approach: what did you do and why?

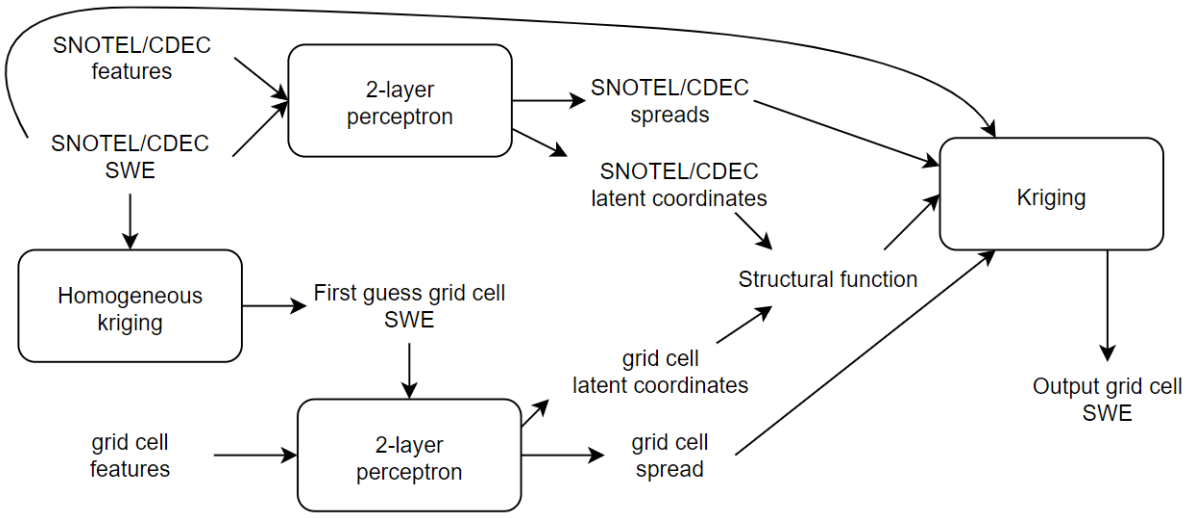
The problem is equivalent to the question of interpretation of the snow water equivalent (SWE) measurements in CDEC and SNOTEL stations to produce the SWE in grid cells. The kriging considers this problem as a regression problem with respect to nearest known SWE values and uses the estimates of the mean, the spread and the structural function. It follows from the reproducing kernel Hilbert space theory that any inhomogeneous structural function is equivalent to a homogeneous kernel in the latent space.

In my approach the neural networks (two layers perceptron's) used for estimations of the mean, the spread and the mapping in the latent space. This perceptron's takes into account all features.

My solution is based on Python 3.9 and uses PyTorch 1.9.1 for modeling. The final result is a mix of 63 inhomogeneous kriging models. The models have 8 different configurations and use different train-validation splitting.

4. Do you have any useful charts, graphs, or visualizations from the process?

   Yes, of course I do! On the chart below the two perceptron's are equivalent.



5. Copy and paste the 3 most impactful parts of your code and explain what each does and how it helped your model.

   A. I saved the model configuration in a single file with model weights. This is a good approach to managing the models' configurations and helps a lot to receive reproducible results, including after modifications in the model code.

```
# The model configuration defined by the dictionary kw. Any new feature of the model must add the new key in this dictionary.
kw = {'usee': True, 'biased': False, 'sigmed': True, 'densed': True, 'implicit': args['implicit'], 'edging': True, 'initgamma': None, 'norm': args['norm'], 'eps': -1.3, 'individual': args['individual'], 'nf': [24], 'nlatent': 3, 'embedding': args['embedding'], 'nmonths': nmonths, 'points': 20, 'gradloss': 0., 'style': 'stack', 'nstations': nstations, 'dropout': 0.2, 'calcsd': 0, 'mc': 3, 'rmaxe': 0.9, 'rmax': 0.7, 'rmax2': 0.5, 'relativer': args['relativer'], 'commonnet': True, 'calibr': args['calibr']}
netargin = [inputs[mode]['xinput'].shape[-1], inputs[mode]['xval'].shape[-1]]
#...<< Training code here >>
best_weights = m.state_dict()
best_weights.update(kw)
torch.save(best_weights, path.join(modelsdir, prefix+'_'+str(fold-1)+'.pt'))
# Loading the model from file fname
bw = torch.load(fname, map_location='cpu')
kw = {key: bw[key] for key in bw if not isinstance(bw[key], torch.Tensor)}
wg = {key: bw[key].to(models.calcdevice) for key in bw if key not in kw}
m = models.Model(inputs[mode]['xinput'].shape[-1], inputs[mode]['xval'].shape[-1], **kw)
```

```
m.load_state_dict(wg)
m.to(device=models.calcdevice).eval()
```

B. I used dictionaries of the PyTorch tensors. This makes the code cleaner, minimizes the potential errors and simplifies the model enhancement process. The dictionaries' keys with the first symbol 'x' correspond to the CDEC/SNOTEL stations. The keys with the first symbol 'y' correspond to the grid cells.

```
# nfunc=1 is a number of the interpreted function (SWE only). nlatent is the dimension of the latent space.
    self.outs = {'map': nlatent}
    if self.sigmed: # =True if model calculate spreads
      self.outs['sigma'] = nfunc
    if self.biased: # =True if model calculate bias
      self.outs['bias'] = nfunc
#<< Define other parts of the model >>
# The variable x is the dictionary of torch.Tensor's. The mapping x function calculates the bias, spread and latent space mapping. The label 'lab' may be 'x' or 'y'.
def mappingx (self, x, lab):
      y = [x[lab+'input']] # using the features
      if self.embedding > 0: # does the  model use embeddings?
        x[lab+'emba'] = self.emb(x[lab+'emb' if lab+'emb' in x else 'xemb']).mul_(self.embmult)
        y.append(x[lab+'emba'])
      if self.usee: # is the model nonlinear ?
        y.append(x[lab+'val'])
      if self.densed: # does the model use the density of the CDEC/SNOTEL stations in the nearest neighborhood ?
        y.append(torch.zeros(x['xval'].shape[:-1]+self.enet.eps.shape[:1],device=x['xval'].device) if lab=='x' else x['esd'])
      y = torch.cat(y,-1) if len(y)>1 else x[lab+'input']
      ok = torch.isfinite(y).all(-1)
      if ok.any():
        ymap = self.net (y[ok]) # Apply the perceptron
        i0 = 0
        for key in self.outs:
          i1 = i0+self.outs[key]
          s = ymap[..., i0:i1]; i0 = i1
          if key == 'sigma': # The default value of spread is equal to 1
            x[lab+key] = torch.ones (y.shape[:-1]+s.shape[-1:], dtype=ymap.dtype, device=ymap.device)
            s = functions.sigma_act.apply(s) # Applying the special positive activation function.
          else: # The default value of mean and latent space mapping is equal to 0
            x[lab+key] = torch.zeros (y.shape[:-1]+s.shape[-1:], dtype=ymap.dtype, device=ymap.device)
          x[lab+key][ok] = s
```

C. Smart mixing of the models. This mixing of the models has significantly improved scores with respect to simple averaging.

```
# Considering only data from SNOTEL/CDEC, e.g. with a key with 'x' in first position.
      x = {key: inputs[key].detach() for key in inputs if key[0] == 'x'}
      x['yid'] = x['xid'] = torch.zeros_like(x['xlo'])
      x['target'] = x['xval'][...,:1].clone()
      trg = x['target'].clone()
      okx = torch.isfinite(trg)
# And calculate one-leave-out results of interpolation to SNOTEL/CDEC
      rets = applymodels (x, models, average=False, device=device)*okx
# The models weights are inverse proportional to MSE of one-leave-out interpolation. The L2 = 1 is regularization
      w = rets - torch.nan_to_num (trg)
```

```
        w = 1./ ((w*w).sum(1)/okx.sum(1) + L2*L2)
        w.div_(w.sum(-1,keepdim=True))
        r = (rets*w[:,None]).sum(-1,keepdim=True)
        w.mul_ ((r*torch.nan_to_num (trg)).sum(1)/(r*r).sum(1)) # The linear regression coefficient
        w.mul_ (0.875) # Magic constant
        del x, rets, okx # Free memory
        gc.collect()
        torch.cuda.empty_cache()
        w = w[:,None]
# The results on CDEC/SNOTEL points are calculated based only on models with embeddings of CDEC/SNOTEL id's.
        if noemb and 'yemb' in inputs:
          ws = w.sum(-1,keepdim=True)
          w = w.expand(-1,inputs['yval'].shape[1],-1)
          noembmodels = torch.tensor([models[m].embedding<=0 for m in models], device=w.device)
          w = w*((inputs['yemb'][0,:,None]>0)|noembmodels)
          w = w/w.sum(-1,keepdim=True)*ws
# Calculate the results on grid cells with smart mixing averaging
        result = applymodels (inputs, models, average=w, device=device, calcsd=calcsd).clamp_(min=0.)
```

6. Please provide the machine specs and time you used to run your model.

   - CPU (model): AMD Threadripper 2950X
   - GPU (model or N/A): Nvidia RTX 2070 8Gb
   - Memory (GB): 64
   - OS: Windows 10
   - Train duration: ~120 hours
   - Inference duration: 1-4 min if MODIS-based features are prepared and ~10 hours if not. The MODIS-based features can be prepared in advance for future days.

7. Anything we should watch out for or be aware of in using your model (e.g. code quirks, memory requirements, numerical stability issues, etc.)?

   The preparation of the DEM-based features used more than 50 Gb of memory. Other blocks are more economical and can work on a system with 32Gb of memory.

   The training process is stochastic. I think this happens because the rounding sometimes leads to different lists of CDEC/SNOTEL stations nearest to the grid cell.

8. Did you use any tools for data preparation or exploratory data analysis that aren't listed in your code submission?

   Yes. I used HRRR initially but not in the final solution.

9. How did you evaluate performance of the model other than the provided metric, if at all?

   I used the custom loss function (x-y)^2/(x+y+1) for the model training

10. What are some other things you tried that didn't necessarily make it into the final workflow (quick overview)?

I tried to use the HRRR forecasts and current MODIS data. In the solution I used only previous years month averaged MODIS data.

11. If you were to continue working on this problem for the next year, what methods or techniques might you try in order to build on your work so far? Are there other fields or features you felt would have been very helpful to have?

I have an improved version of my model. This version can also return the (based on random fields theory) estimations of the standard deviation (and confidence intervals) of the SWE. I believe this information is very helpful to estimate the risks of data-driven decision-making. For example, it can suggest the best locations of the new CDEC/SNOTEL stations which takes into account all used features (DEM, GlobCover, soil types, etc.). Putting the stations in those locations will most likely improve the results in the considered region.
Limited experiments that I ran after the deadline showed that the gaussian assumption is not best for this problem: the distribution of SWE values have the «fat tails». The Pareto-II distribution demonstrates some lower RMSE.

I am guessing that taking into account the radar-based precipitation and cloudiness composites can also improve the model.