Text and Analysis for in-progress Mountain Rain or Snow Manuscript

Keith Jennings

Sys.Date()

What is this?

This sentence you're reading is probably part of a rendered Markdown document on GitHub and it's based on an RMarkdown file I put together to analyze data for a Mountain Rain or Snow manuscript that explores the limits of rain-snow partitioning using near-surface meteorology.

What is this directory?

The directory, for which this document serves as the README, includes all of the R scripts for tuning and processing rain-snow partitioning models.

Who am I? Why I am here?

The necessary stuff

Import the packages we need.

```
# Load R packages
library(tidyverse)
library(tidymodels)
library(cowplot); theme_set(theme_cowplot())
```

Import the datasets we need.

```
# `cache=TRUE` keeps the data stored and prevents it from being read in anew each time

# Data directory prefix
# Specific to my computer, change before sharing
data_pre <- "../../../../data/"

# Load the results files
# These are local files (update document if moved)
# Also update nh scenario to be "nomix_onlymet"
# And add a source_label for plotting & tables
```

```r
summary_all <- readRDS(paste0(data_pre, "performance_summary_all.RDS")) %>%
    mutate(scenario = ifelse(source == "nh", "nomix_onlymet", scenario),
           source_lab = ifelse(source == "cs", "Crowdsourced", "Synoptic"))
summary_bytemp <- readRDS(paste0(data_pre, "performance_summary_bytemp.RDS")) %>%
    mutate(scenario = ifelse(source == "nh", "nomix_onlymet", scenario),
           source_lab = ifelse(source == "cs", "Crowdsourced", "Synoptic"))
# predictions_all <- readRDS(paste0(data_pre, "predictions_all.RDS"))

# Load the tuning data
# These are local files (update document if moved)
load(paste0(data_pre, "rf_tune_results.rdata"))
load(paste0(data_pre, "xg_tune_results.rdata"))
```

Results

Rain-snow partitioning performance

Benchmarks

The benchmark methods exhibited variable performance in predicting rain versus snow in both the crowdsourced and synoptic datasets. Table 1 shows the accuracy, snow bias, and rain bias for each of the partitioning methods with a few key similarities and differences. The methods incorporating humidity (e.g., the binary logistic regression model and the wet bulb and dew point temperature thresholds) generally performed better than the methods using only air temperature. In the crowdsourced dataset, the top 6 methods, ranked by accuracy, all used humidity, while the bottom 3 relied on air temperature alone. The case was slightly different for the synoptic dataset where the two dew point temperature thresholds performed worse than the two static air temperature thresholds and the spatially varying air temperature threshold.

```r
# Calculate minimums, maximums, and ranges

# Crowdsourced
# Accuracy
cs_benchmark_accuracy_min = summary_all %>% filter(source == "cs", scenario == "nomix_onlymet" & !ppm %
cs_benchmark_accuracy_max = summary_all %>% filter(source == "cs", scenario == "nomix_onlymet" & !ppm %
cs_benchmark_accuracy_range = cs_benchmark_accuracy_max - cs_benchmark_accuracy_min
# Snow bias
cs_benchmark_snow_bias_min = summary_all %>% filter(source == "cs", scenario == "nomix_onlymet" & !ppm
cs_benchmark_snow_bias_max = summary_all %>% filter(source == "cs", scenario == "nomix_onlymet" & !ppm
cs_benchmark_snow_bias_range = cs_benchmark_snow_bias_max - cs_benchmark_snow_bias_min
# Rain bias
cs_benchmark_rain_bias_min = summary_all %>% filter(source == "cs", scenario == "nomix_onlymet" & !ppm
cs_benchmark_rain_bias_max = summary_all %>% filter(source == "cs", scenario == "nomix_onlymet" & !ppm
cs_benchmark_rain_bias_range = cs_benchmark_rain_bias_max - cs_benchmark_rain_bias_min


# Synoptic
nh_benchmark_accuracy_min = summary_all %>% filter(source == "nh", scenario == "nomix_onlymet" & !ppm %
nh_benchmark_accuracy_max = summary_all %>% filter(source == "nh", scenario == "nomix_onlymet" & !ppm %
nh_benchmark_accuracy_range = nh_benchmark_accuracy_max - nh_benchmark_accuracy_min
# Snow bias
nh_benchmark_snow_bias_min = summary_all %>% filter(source == "nh", scenario == "nomix_onlymet" & !ppm
```

```
nh_benchmark_snow_bias_max = summary_all %>% filter(source == "nh", scenario == "nomix_onlymet" & !ppm %
nh_benchmark_snow_bias_range = nh_benchmark_snow_bias_max - nh_benchmark_snow_bias_min
# Rain bias
nh_benchmark_rain_bias_min = summary_all %>% filter(source == "nh", scenario == "nomix_onlymet" & !ppm %
nh_benchmark_rain_bias_max = summary_all %>% filter(source == "nh", scenario == "nomix_onlymet" & !ppm %
nh_benchmark_rain_bias_range = nh_benchmark_rain_bias_max - nh_benchmark_rain_bias_min
```

The range in method performance also varied across the datesets. Accuracy ranged from 80.5% to 89.1%, a total spread of 8.6% for the crowdsourced dataset. That is larger than the performance spread of the synoptic dataset (2.4%), which had a minimum of 90.7% and a maximum of 93.1%. Snow and rain biases had larger ranges than accuracy in both datasets. Snow bias varied by 28.6% and 16.7%, while rain bias varied by 64.1% and 14.7% in the crowdsourced and synoptic datasets, respectively. In general, methods with better accuracy values had lower rain and snow biases.

```
summary_all %>%
    filter(scenario == "nomix_onlymet") %>%
    filter(!ppm %in% c("xg", "rf")) %>%
    ungroup() %>%
    arrange(source_lab, -accuracy_pct) %>%
    mutate(across(where(is.numeric), round, digits = 1)) %>%
    select(Source = source_lab,
           PPM = ppm,
           `Accuracy (%)` = accuracy_pct,
           `Snow bias (%)` = snow_bias_pct,
           `Rain bias (%)` = rain_bias_pct) %>%
    knitr::kable()
```

```
## Warning: There was 1 warning in `mutate()`.
## i In argument: `across(where(is.numeric), round, digits = 1)`.
## Caused by warning:
## ! The `...` argument of `across()` is deprecated as of dplyr 1.1.0.
## Supply arguments directly to `.fns` through an anonymous function instead.
##
##   # Previously
##   across(a:b, mean, na.rm = TRUE)
##
##   # Now
##   across(a:b, \(x) mean(x, na.rm = TRUE))
```

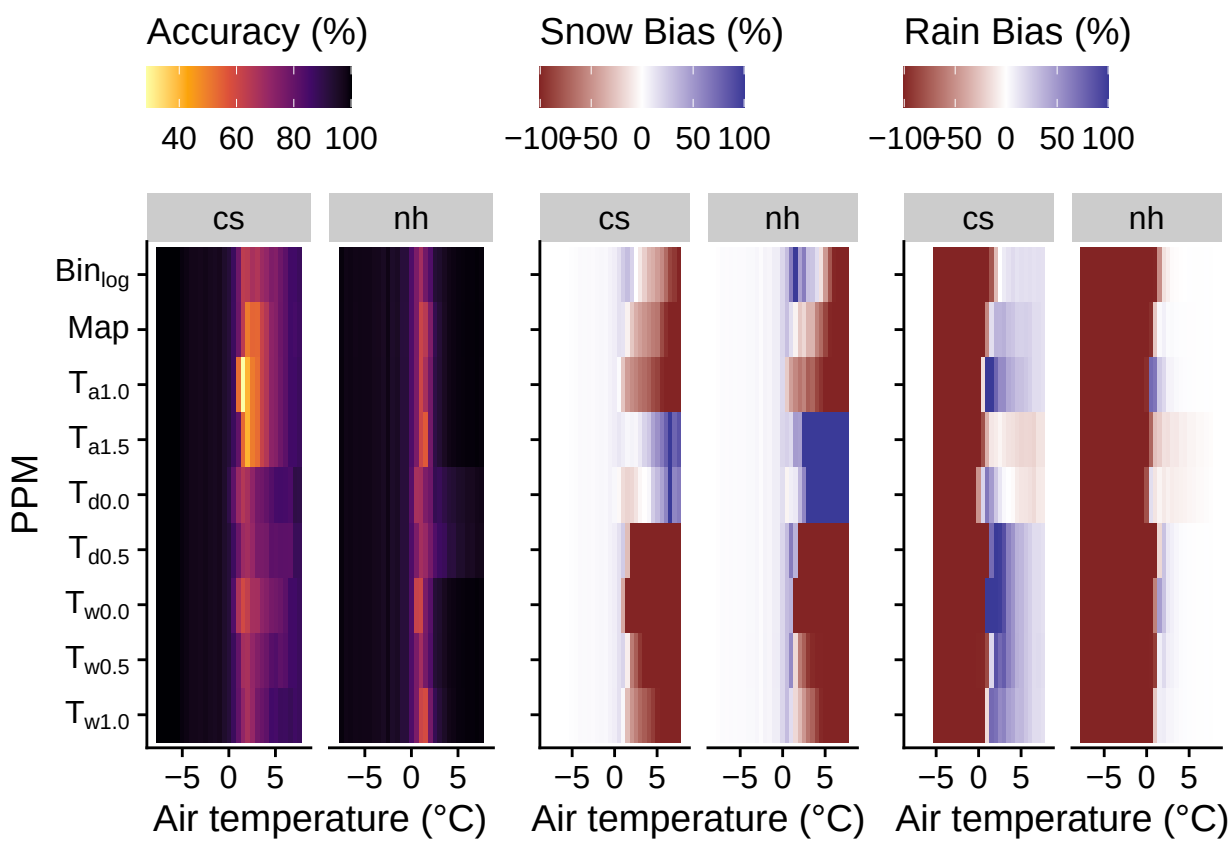
Table 1: Table 1. Performance metrics for the benchmark precipitation partitioning methods (PPMs) using the crowdsourced and synoptic datasets.

Source	PPM	Accuracy (%)	Snow bias (%)	Rain bias (%)
Crowdsourced	thresh_twet_1	89.1	4.2	-9.4
Crowdsourced	thresh_twet_0.5	88.4	-3.7	8.4
Crowdsourced	thresh_tdew_0.5	87.9	7.5	-16.8
Crowdsourced	thresh_tdew_0	87.3	0.0	0.0
Crowdsourced	binlog	86.7	-9.3	20.9
Crowdsourced	thresh_twet_0	85.8	-12.6	28.3
Crowdsourced	map	84.4	-9.4	21.3

Source	PPM	Accuracy (%)	Snow bias (%)	Rain bias (%)
Crowdsourced	thresh_tair_1.5	83.4	-13.0	29.2
Crowdsourced	thresh_tair_1	80.5	-21.1	47.3
Synoptic	thresh_twet_0.5	93.1	6.2	-5.5
Synoptic	binlog	93.1	5.6	-5.0
Synoptic	thresh_twet_0	92.2	-2.6	2.3
Synoptic	thresh_tair_1	92.1	4.5	-4.0
Synoptic	thresh_tair_1.5	91.9	8.2	-7.2
Synoptic	thresh_twet_1	91.9	14.1	-12.4
Synoptic	map	91.8	8.2	-7.6
Synoptic	thresh_tdew_0	90.9	9.8	-8.7
Synoptic	thresh_tdew_0.5	90.7	13.5	-11.9

All of the benchmark methods performed worse at air temperatures near and slightly above the freezing point in terms of accuracy, with most of them reaching their minimum accuracy values between 1.5°C and 2.0°C in the crowdsourced dataset and between 0.5°C and 1.5°C in the synoptic dataset (@ref(fig:benchmark_plot)). In both datasets, most methods had slightly positive snow biases and markedly negative rain biases at air temperatures < 1.0°C, which is consistent with their limited, or complete lack of, ability to predict rain at sub-freezing temperatures. Conversely, most methods, with the exception of the dew point temperature thresholds, had largely negative snow biases and slightly positive rain biases at air temperatures above 2°C. Notably, the crowdsourced dataset expressed larger positive rain biases at higher temperatures, indicating the benchmark methods failed to capture snowfall occurring during warm near-surface conditions.

```
## Warning: Removed 792 rows containing missing values (‘geom_raster()‘).
## Removed 792 rows containing missing values (‘geom_raster()‘).
## Removed 792 rows containing missing values (‘geom_raster()‘).
```




```
# Code for minimum accuracy
summary_bytemp %>% filter(source == "cs" & scenario == "nomix_onlymet" & !ppm %in% c("rf", "xg")) %>% g
```

```
## # A tibble: 9 x 16
## # Groups:   ppm [9]
##   ppm    scenario source tair_bin     n accuracy_pct snow_bias_pct rain_bias_pct
##   <chr>  <chr>    <chr>     <dbl> <int>        <dbl>         <dbl>         <dbl>
## 1 binlog nomix_o~ cs          1.5  1808         63.7         -29.8          73.5
## 2 map    nomix_o~ cs          2    1693         50.3         -59.4          87.0
## 3 thres~ nomix_o~ cs          1.5  1808         28.8        -100           247.
## 4 thres~ nomix_o~ cs          2    1702         40.7        -100           146.
## 5 thres~ nomix_o~ cs          1.5  1808         65.5         -19.5          48.2
## 6 thres~ nomix_o~ cs          2    1702         68.6           5.75         -8.37
## 7 thres~ nomix_o~ cs          1.5  1808         59.8         -44.4          110.
## 8 thres~ nomix_o~ cs          2    1702         67.7         -24.7          35.9
## 9 thres~ nomix_o~ cs          2    1702         67.8          20.8         -30.3
## # i 8 more variables: mixed_bias_pct <dbl>, snow_obs_pct <dbl>,
## #   rain_obs_pct <dbl>, mixed_obs_pct <dbl>, snow_pred_pct <dbl>,
## #   rain_pred_pct <dbl>, mixed_pred_pct <dbl>, source_lab <chr>
```

```
summary_bytemp %>% filter(source == "nh" & scenario == "nomix_onlymet" & !ppm %in% c("rf", "xg")) %>% g
```

```
## # A tibble: 9 x 16
## # Groups:   ppm [9]
##   ppm    scenario source tair_bin     n accuracy_pct snow_bias_pct rain_bias_pct
```

```
##   <chr> <chr>    <chr>     <dbl>  <int>        <dbl>         <dbl>         <dbl>
## 1 binl~ nomix_o~ nh          1   1.19e6         68.7          16.1          -25.1
## 2 map   nomix_o~ nh          1   1.08e6         61.3          56.8          -89.5
## 3 thre~ nomix_o~ nh          1   1.19e6         60.7          36.3          -56.7
## 4 thre~ nomix_o~ nh          1.5 2.42e5         56.1          37.1          -17.4
## 5 thre~ nomix_o~ nh          0.5 3.68e5         67.8          -4.82          15.3
## 6 thre~ nomix_o~ nh          1   1.19e6         68.4          17.8          -27.8
## 7 thre~ nomix_o~ nh          1   1.19e6         60.8         -43.1           67.3
## 8 thre~ nomix_o~ nh          1   1.19e6         68.7          15.6          -24.3
## 9 thre~ nomix_o~ nh          1.5 2.42e5         59.2         108.           -50.8
## # i 8 more variables: mixed_bias_pct <dbl>, snow_obs_pct <dbl>,
## #   rain_obs_pct <dbl>, mixed_obs_pct <dbl>, snow_pred_pct <dbl>,
## #   rain_pred_pct <dbl>, mixed_pred_pct <dbl>, source_lab <chr>
```

Machine learning methods

The two machine learning methods, random forest and XGBoost, generally provided small performance gains relative to the best benchmark methods and large, consistent improvements relative to the median benchmark methods. In the crowdourced dataset, XGBoost had a higher accuracy than $T_{w1.0}$, while random forest had a slightly lower accuracy than the threshold, which was the best performing method from the benchmark exercise. However, both XGBoost and random forest outperformed $T_{w0.5}$ and Bin_{log}, the best PPMs in the synoptic dataset as measured by the accuracy metric. When compared to the snow and rain biases produced by the best and median benchmark methods, the machine learning methods all provided consistent improvements in both datasets.

```
cols_to_get = c("source", "ppm", "accuracy_pct", "snow_bias_pct", "rain_bias_pct")


cs_best_benchmark = summary_all %>%
    filter(source == "cs" & !ppm %in% c("xg", "rf") & scenario == "nomix_onlymet") %>%
    ungroup() %>%
    slice_max(accuracy_pct, n = 1) %>%
    pull(ppm)

cs_median_benchmark = summary_all %>%
    filter(source == "cs" & !ppm %in% c("xg", "rf") & scenario == "nomix_onlymet") %>%
    ungroup() %>%
    arrange(-accuracy_pct) %>%
    slice(5) %>%
    pull(ppm)

cs_benchmark_diff <- summary_all %>%
    filter(source == "cs" & ppm %in% c("xg", "rf") & scenario == "nomix_onlymet") %>%
    select(cols_to_get) %>%
    bind_cols(.,summary_all %>%
                  filter(source == "cs" & ppm == cs_best_benchmark & scenario == "nomix_onlymet") %>%
                  select(cols_to_get)) %>%
    bind_rows(., summary_all %>%
    filter(source == "cs" & ppm %in% c("xg", "rf") & scenario == "nomix_onlymet") %>%
    select(cols_to_get) %>%
    bind_cols(.,summary_all %>%
                  filter(source == "cs" & ppm == cs_median_benchmark & scenario == "nomix_onlymet") %>%
                  select(cols_to_get)))
```

```
## Warning: Using an external vector in selections was deprecated in tidyselect 1.1.0.
## i Please use ‘all_of()‘ or ‘any_of()‘ instead.
##   # Was:
##   data %>% select(cols_to_get)
##
##   # Now:
##   data %>% select(all_of(cols_to_get))
##
## See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.

## Adding missing grouping variables: ‘scenario‘
## Adding missing grouping variables: ‘scenario‘
## New names:
## Adding missing grouping variables: ‘scenario‘
## Adding missing grouping variables: ‘scenario‘
## New names:
```

```
nh_best_benchmark = "binlog"

nh_median_benchmark = summary_all %>%
    filter(source == "nh" & !ppm %in% c("xg", "rf") & scenario == "nomix_onlymet") %>%
    ungroup() %>%
    arrange(-accuracy_pct) %>%
    slice(5) %>%
    pull(ppm)

nh_benchmark_diff <- summary_all %>%
    filter(source == "nh" & ppm %in% c("xg", "rf") & scenario == "nomix_onlymet") %>%
    select(cols_to_get) %>%
    bind_cols(.,summary_all %>%
                  filter(source == "nh" & ppm == nh_best_benchmark & scenario == "nomix_onlymet") %>%
                  select(cols_to_get)) %>%
    bind_rows(., summary_all %>%
    filter(source == "nh" & ppm %in% c("xg", "rf") & scenario == "nomix_onlymet") %>%
    select(cols_to_get) %>%
    bind_cols(.,summary_all %>%
                  filter(source == "nh" & ppm == nh_median_benchmark & scenario == "nomix_onlymet") %>%
                  select(cols_to_get)))
```

```
## Adding missing grouping variables: ‘scenario‘
## Adding missing grouping variables: ‘scenario‘
## New names:
## Adding missing grouping variables: ‘scenario‘
## Adding missing grouping variables: ‘scenario‘
## New names:
```

```
ml_benchmark_diff <- bind_rows(cs_benchmark_diff, nh_benchmark_diff) %>%
    mutate(comparison = rep(c("best", "best", "med", "med"), 2)) %>%
    select(-scenario...1,
           source = source...2,
```

```
            ppm = ppm...3,
            comparison,
            ml_accuracy_pct = accuracy_pct...4,
            ml_snow_bias_pct = snow_bias_pct...5,
            ml_rain_bias_pct = rain_bias_pct...6,
            bm_accuracy_pct = accuracy_pct...10,
            bm_snow_bias_pct = snow_bias_pct...11,
            bm_rain_bias_pct = rain_bias_pct...12,
            -scenario...7, -source...8, -ppm...9) %>%
    mutate(accuracy_pct_diff = ml_accuracy_pct - bm_accuracy_pct,
            snow_bias_abs_pct_diff = abs(ml_snow_bias_pct) - abs(bm_snow_bias_pct),
            rain_bias_abs_pct_diff = abs(ml_rain_bias_pct) - abs(bm_rain_bias_pct))

# Make a table
ml_benchmark_diff %>%
    select(source, ppm, comparison,
            ml_accuracy_pct, accuracy_pct_diff,
            ml_snow_bias_pct, snow_bias_abs_pct_diff,
            ml_rain_bias_pct, rain_bias_abs_pct_diff) %>%
    mutate(across(where(is.numeric), round, digits = 1)) %>%
    knitr::kable()
```

source	ppm	comparison	ml_accuracy_pct	accuracy_pct_diff	ml_snow_bias_pct	snow_bias_abs_pct_diff	ml_rain_bias_pct	rain_bias_abs_pct_diff
cs	rf	best	88.8	-0.3	3.0	-1.1	-6.8	-2.5
cs	xg	best	89.6	0.5	3.8	-0.4	-8.5	-0.8
cs	rf	med	88.8	2.2	3.0	-6.3	-6.8	-14.0
cs	xg	med	89.6	3.0	3.8	-5.5	-8.5	-12.3
nh	rf	best	93.7	0.6	3.7	-1.9	-3.3	-1.7
nh	xg	best	93.3	0.2	5.5	-0.1	-4.8	-0.1
nh	rf	med	93.7	1.8	3.7	-4.5	-3.3	-4.0
nh	xg	med	93.3	1.4	5.5	-2.7	-4.8	-2.4

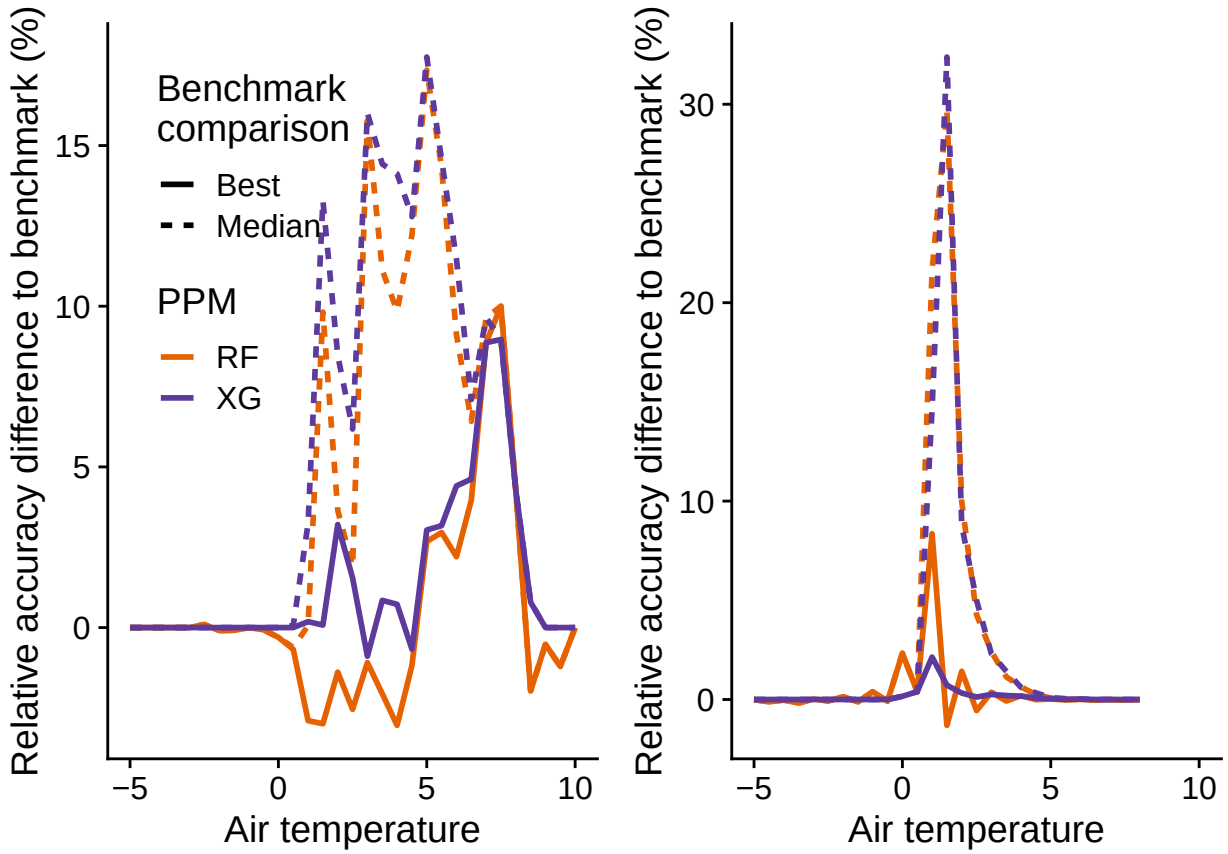
The two machine learning methods exhibited similar accuracy patterns by air temperature as the best and median benchmark methods, with performance degrading at air temperatures slightly above freezing. Notably, when examining relative differences in accuracy by air temperature, random forest and XGBoost provided larger performance increases in the temperature ranges where the benchmark methods performed worst. In the crowdsourced dataset, this increase was at most worth nearly 10% compared to the best and nearly 15% compared to the median benchmark. In the synoptic dataset, the machine learning relative performance increase compared to the median benchmark was worth approximately 30% at 1.5°C.

```
plot_grid(
  summary_bytemp %>%
  filter(source == "cs" & scenario == "nomix_onlymet") %>%
    pivot_wider(names_from = ppm, values_from = accuracy_pct, id_cols = tair_bin) %>%
  mutate(rf_best = (rf-thresh_twet_1)/thresh_twet_1*100,
         rf_med = (rf-binlog)/binlog*100,
         xg_best = (xg-thresh_twet_1)/thresh_twet_1*100,
         xg_med = (xg-binlog)/binlog*100) %>%
  select(tair_bin, rf_best:xg_med) %>%
  pivot_longer(cols = rf_best:xg_med,
               names_to = c("ppm", "benchmark"),
```

```
              names_pattern = "(.*)_(.*)",
              values_to = "accuracy_diff_rel_pct") %>%
  ggplot(aes(tair_bin, accuracy_diff_rel_pct, color = ppm, lty = benchmark)) +
  geom_line(lwd = 1) +
  scale_color_manual(values =  c("#E66100",
                                 "#5D3A9B"),
                     labels = c("RF", "XG"),
                     name = "PPM")+
  scale_linetype_discrete(labels = c("Best", "Median"),
                          name = "Benchmark\ncomparison") +
  labs(x = "Air temperature",
       y = "Relative accuracy difference to benchmark (%)") +
  xlim(c(-5,10)) +
    theme(legend.position = c(0.1,0.7)),
  summary_bytemp %>%
  filter(source == "nh" & scenario == "nomix_onlymet") %>%
    pivot_wider(names_from = ppm, values_from = accuracy_pct, id_cols = tair_bin) %>%
  mutate(rf_best = (rf-binlog)/binlog*100,
         rf_med = (rf-thresh_twet_1)/thresh_twet_1*100,
         xg_best = (xg-binlog)/binlog*100,
         xg_med = (xg-thresh_twet_1)/thresh_twet_1*100) %>%
  select(tair_bin, rf_best:xg_med) %>%
  pivot_longer(cols = rf_best:xg_med,
              names_to = c("ppm", "benchmark"),
              names_pattern = "(.*)_(.*)",
              values_to = "accuracy_diff_rel_pct") %>%
  ggplot(aes(tair_bin, accuracy_diff_rel_pct, color = ppm, lty = benchmark)) +
  geom_line(lwd = 1) +
  scale_color_manual(values =  c("#E66100",
                                 "#5D3A9B"),
                     labels = c("RF", "XG"),
                     name = "PPM")+
  scale_linetype_discrete(labels = c("Best", "Median"),
                          name = "Benchmark\ncomparison") +
  labs(x = "Air temperature",
       y = "Relative accuracy difference to benchmark (%)") +
  xlim(c(-5,10)) +
    theme(legend.position = "none"),
  ncol = 2
)
```

```
## Warning: Removed 344 rows containing missing values (‘geom_line()‘).
```

```
## Warning: Removed 24 rows containing missing values (‘geom_line()‘).
```



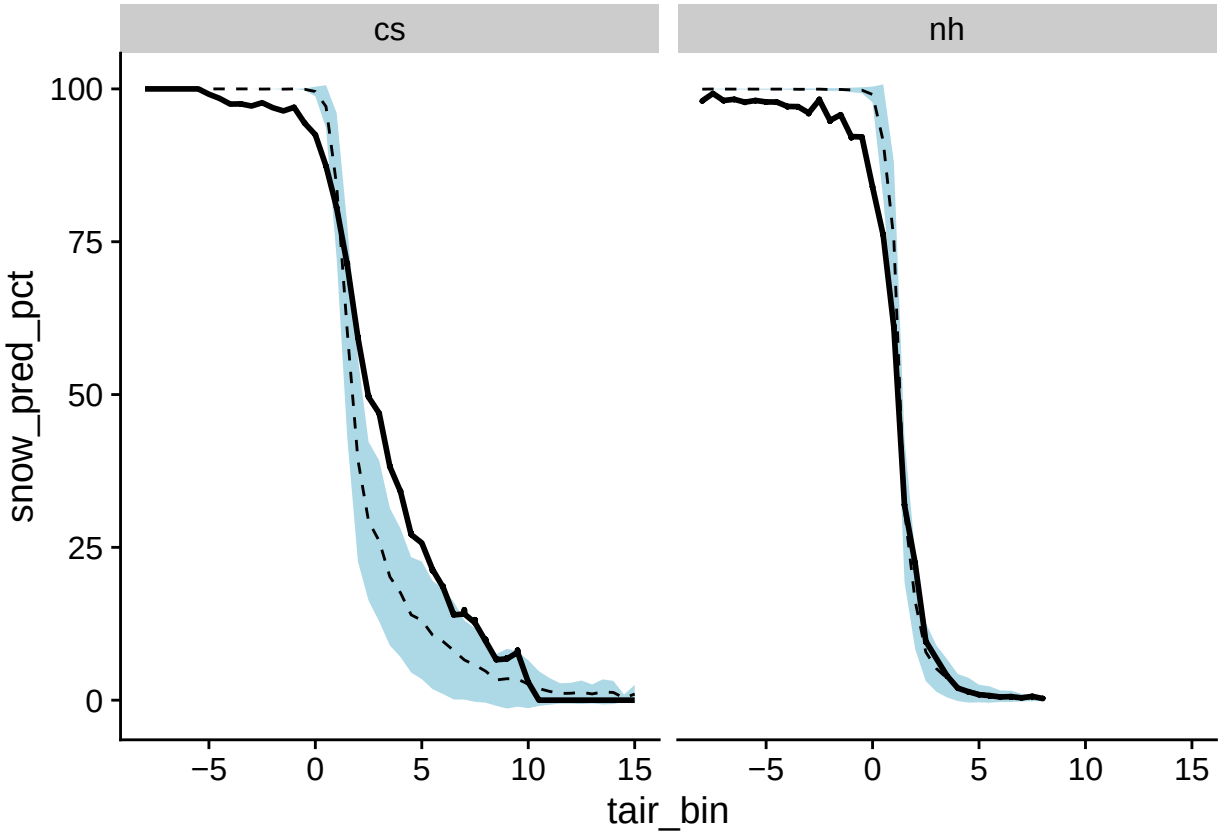
Meteorology and snow frequency analysis

```
plot_data <- summary_bytemp %>%
    filter(scenario == "nomix_onlymet")

ggplot(plot_data, aes(tair_bin, snow_pred_pct)) +
    stat_summary(geom="ribbon", fun.data=mean_cl_normal,
                 fun.args=list(conf.int=0.95), fill="lightblue") +
      stat_summary(geom="line", fun.y=mean, linetype="dashed")+
    xlim(-8, 15) +
    geom_line(aes(tair_bin, snow_obs_pct), color = "black", lwd = 1) +
    facet_wrap(~source, scales = "free_x")
```

```
## Warning: The ‘fun.y‘ argument of ‘stat_summary()‘ is deprecated as of ggplot2 3.3.0.
## i Please use the ‘fun‘ argument instead.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```

```
## Warning: Removed 770 rows containing non-finite values (‘stat_summary()‘).
## Removed 770 rows containing non-finite values (‘stat_summary()‘).
```



Variable importance

The importance of wet bulb temperature is further underscored by variable importance scores from the random forest and XGBoost analyses. Regardless of the machine learning method or scenario, the most important variable was consistently wet bulb temperature.

test <- readRDS("../../data/rf_predict_CSdataset_full.RDS") test[[1]][[2]]%>% vip::vip()

Supplemental Information

Hyperparameter tuning

Random Forest

For the random forest model, we found that overall performance was insensitive to the choice of hyperparameters. First, we randomly selected 1% of the Jennings et al. dataset (178108 observations) and used those to run a hyperparameter tuning workflow in the tidymodels R package. We used two performance metrics to evaluate the models: accuracy and area under the curve (AUC). The former represents the proportion of observations correctly predicted and the latter corresponds to the area under the receiver operating characteristic curve, which plots the true positive rate against the false positive rate. Both metrics range from 0 to 1, where 0 equals no observations correctly predicted and 1 equals all observations correctly predicted.

In this tuning workflow, we provided three mtry values (3, 4, and 5) and three tree values (100, 300, and 500), giving a total of 9 unique hyperparameter combinations, which we tested across ten folds of the data.

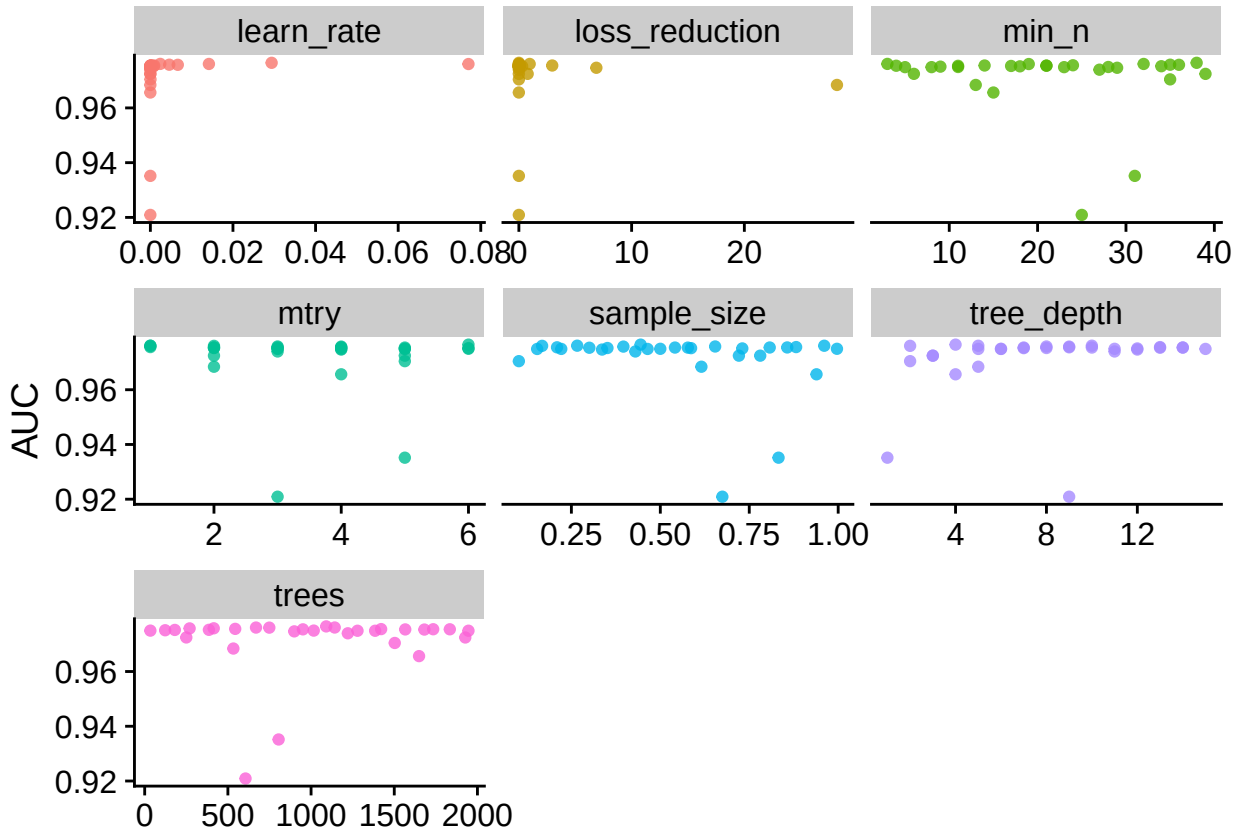
Within this hyperparameter space, we found almost no variation, with mean accuracy ranging from 0.924 to 0.924 and mean AUC similarly ranging from 0.969 to 0.97. The best-performing hyperparameters were mtry = 5 and trees = 500.

Next, we performed the tuning using all entries from the crowdsourced precipitation phase dataset in four different scenarios: 1) include mixed precipitation and only use meteorological variables, 2) exclude mixed precipitation and only use meteorological variables, 3) include mixed precipitation and use meteorology plus EPA Level III ecoregions, and 4) exclude mixed precipitation and use meteorology plus EPA Level III ecoregions. In each of these four scenarios we found little sensitivity in the mean accuracy and mean AUC, confirming the results of the tuning exercise on the northern hemisphere dataset. Mean accuracy ranged from 0.771 to 0.774 for scenario 1, 0.886 to 0.887 for scenario 2, 0.785 to 0.786 for scenario 3, and 0.897 to 0.898 for scenario 4. Similarly mean AUC ranged from 0.801 to 0.802 for scenario 1, 0.941 to 0.943 for scenario 2, 0.822 to 0.824 for scenario 3, and 0.949 to 0.95 for scenario 4.

XGBoost

We also performed hyperparameter tuning on the XGBoost models, which have a greater number of hyperparameters and thus required a slightly different tuning approach. Here, instead of prescribing the hyperparameter values and combinations, we created a 30 element latin hypercube of the hyperparameter space across ten folds of the data. We again started with a 1% sample of the Jennings et al. (2018) dataset and found, at first glance, that output from the XGBoost model appeared to be more sensitive to hyperparameter values than the random forest (FIG S#). However, a closer look at the data showed much of the sensitivity was a result of low learn rate values. For example, if we removed the lowest learn rate value (1.12e-10), we found the range in mean accuracy shifted from 0.529–0.932 to 0.931–0.932 and mean AUC shifted from 0.921–0.976 to 0.935–0.976.

```
nh_nomix_metonly_xg_tune_results %>%
    filter(.metric == "roc_auc") %>%
    select(mean, mtry:sample_size) %>%
    pivot_longer(mtry:sample_size,
                 values_to = "value",
                 names_to = "parameter"
    ) %>%
    ggplot(aes(value, mean, color = parameter)) +
    geom_point(alpha = 0.8, show.legend = FALSE) +
    facet_wrap(~parameter, scales = "free_x") +
    labs(x = NULL, y = "AUC")
```



Finally, we performed the XGBoost tuning using the crowdsourced precipitation phase dataset in the four different scenarios we detailed in the random forest subsection above. We again used a 30 element latin hypercube of the hyperparameter space across ten folds of the data. Mean accuracy ranged from 0.119 to 0.788 for scenario 1, 0.863 to 0.895 for scenario 2, 0.119 to 0.796 for scenario 3, and 0.302 to 0.9 for scenario 4. Mean AUC ranged from 0.5 to 0.824 for scenario 1, 0.897 to 0.951 for scenario 2, 0.5 to 0.838 for scenario 3, and 0.5 to 0.955 for scenario 4.

Notably, low learn rate values drove again drove the wide range in performance for the XGBoost models. In this case, removing the 8 lowest learn rate values—out of 30—from the latin hypercube produced much narrower ranges in mean accuracy and AUC. When only considering learn rate values greater than 1.67e-8, mean accuracy ranged from 0.78 to 0.788 for scenario 1, 0.889 to 0.895 for scenario 2, 0.119 to 0.796 for scenario 3, and 0.698 to 0.9 for scenario 4. Similarly mean AUC ranged from 0.779 to 0.824 for scenario 1, 0.897 to 0.951 for scenario 2, 0.5 to 0.838 for scenario 3, and 0.935 to 0.955 for scenario 4.