Documentation

Katie Yi, katieyiowa@gmail.com, Doc Start Date: Jun 14, 2023

This document is a collection of my work from June 14 - July 31, 2023.
All work and thoughts are my own unless noted otherwise.
Additional work is found in write-ups. Final_XGB_Writeup

KEY

Yellow: Question Flag
Green: Assumption
Blue: Idea

Contents

Paths

Working Directory:
MyDrive/REU 2023 Team 1: Ice Bed Topography Prediction/Research/Yi_Work

Documentation:
MyDrive/REU 2023 Team 1: Ice Bed Topography Prediction/Research/Yi_Work

Background Research:
MyDrive/REU 2023 Team 1: Ice Bed Topography Prediction/Research/Yi_Work/Background/*

Data:
MyDrive/REU 2023 Team 1: Ice Bed Topography Prediction/Research/Yi_Work/Data/*

Scripts:
MyDrive/REU 2023 Team 1: Ice Bed Topography Prediction/Research/Yi_Work/Scripts/*

Road Map

Task	Status	Notes
Week 1 presentation	Completed	
Brainstorm	Completed	
Download and test Panoply	Completed	For data visualization
Understand the data	Completed	~~Waiting on answers from Dr. M~~
Exploration	Completed	
Merging dataset & minor cleaning	Completed	
Code & Model Review	Completed	Review all models & viz
Identify key areas to explore	Completed	Prob Models
GPR	Canceled	
STGP	Completed	
LSTM & Dense	Completed	
VAE	Completed	
XGBoost	Completed	
VAE & XGB Combined	Completed	
GAN	Canceled	towardsDataScience
Other GP	Canceled	1. MIT Press 2. GP Large Data Approx 3. GP Relation other models

Team Road Map

Task Tracker

Background

Background Reading Notes

Variable	Source	Meaning
surf_x	Raw	Coordinates of cell centers, meters
surf_y	Raw	Coordinates of cell centers, meters
surf_vx	Raw	Iceflow: meters per year
surf_vy	Raw	Iceflow: meters per year
surf_elv	Raw	Surface height: meters. Assuming from sea level (confirmed) Is this normalized by the rising sea level each year data was collected for? Not normalized and sea level rise is too small to change anything (~4 mm/yr, the uncertainty in the data is 30-50 m)
surf_dhdt	Raw	Ice thinning rates: meters per year
surf_SMB	Raw	Snow accumulation: meters per year Depth or difference? m/yr of *new snow* in “ice equivalent”
track_bed_x	Raw	Specific x coord inside cell, meters
track_bed_y	Raw	Specific y coord inside cell, meters
v_mag	Deriv…	Velocity magnitude of iceflow at each (x, y)
track_bed_target	Raw	Target: bed height
p	Deriv…	(Completed Previously) Row p to look for features according to real-world location - in conjunction with q (track_bed_x - surf_x[0,1])/150 = first element of the index
q	Deriv…	(Completed Previously) Column q to look for features according to real-world location - in conjunction with p (surf_y[-1,0] - track_bed_y)/150

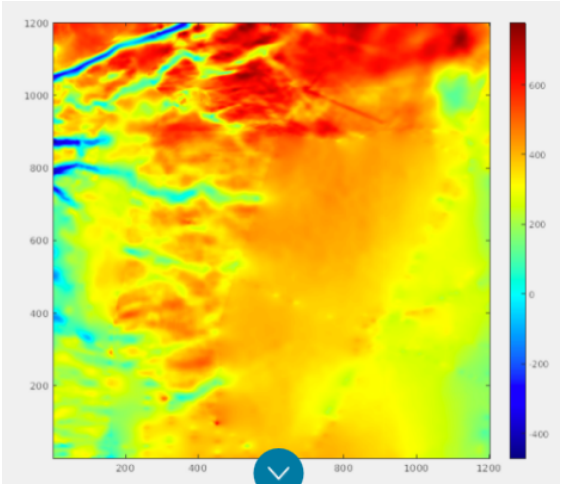
Goal: Estimating topography (beneath the ice) of greenland ice beds by predicting missing data in topographic maps. Improve bed mapping variation - in between levels.

10 features; 1201x1201=1,442,401 points
632,706 (43.86%) data known

Method:
Preprocessing 1:

- Use surf_x & surf_y centers to find the correct features for each section
  - Bilinear interpolation
  - Nearest Neighbors
- Merge cells to create csv

Prepare data (Preprocessing 2)
Predict height
Run accuracy tests



To Beat

Based on physics
See Physics Model Metrics.

Table 2: Performance Comparison

Model	RMSE	MAE	R^2
Gradient Boosting	136.93	100.63	0.113
XGBoost	135.18	97.06	0.135
Lasso Regressor	147.30	106.79	-0.026
Dense	387.98	364.02	-6.110
LSTM	556.09	493.71	-13.620
Dense+LSTM	**57.30**	**55.04**	**0.840**

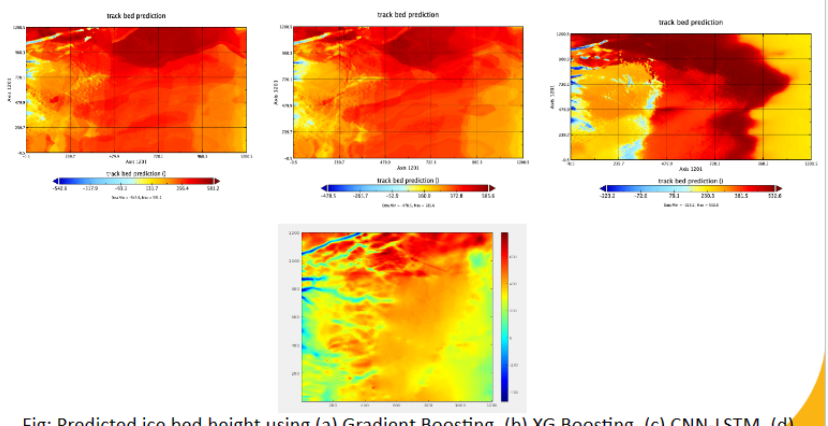


Fig: Predicted ice bed height using (a) Gradient Boosting, (b) XG Boosting, (c) CNN-LSTM, (d) physics

---

Exploration

Input Paths:
train_ = "/content/gdrive/MyDrive/REU 2023 Team 1: Ice Bed Topography Prediction/Research/Yi_Work/Data/train.csv"
y_test_ = "/content/gdrive/MyDrive/REU 2023 Team 1: Ice Bed Topography Prediction/Research/Yi_Work/Data/y_test.csv"
x_test_ = "/content/gdrive/MyDrive/REU 2023 Team 1: Ice Bed Topography Prediction/Research/Yi_Work/Data/test.csv"

Script: explore_KY_230614.ipynb
Output Paths: NA

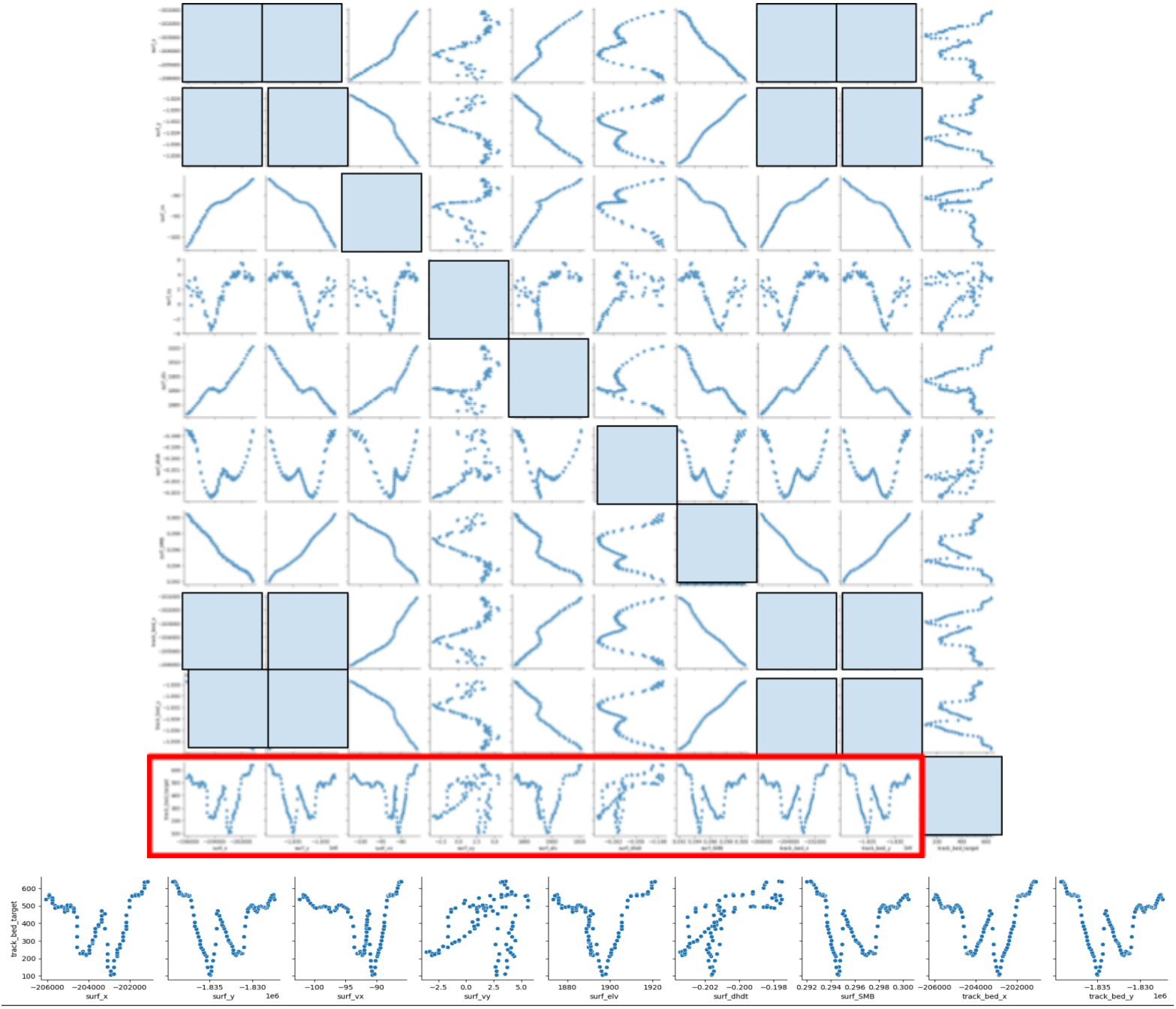
Notes & Thoughts:

- The accuracy could be improved by improving the distribution of the datasets.
    - Can we run our model on multiple random seeds for data splits?

Visualizations:

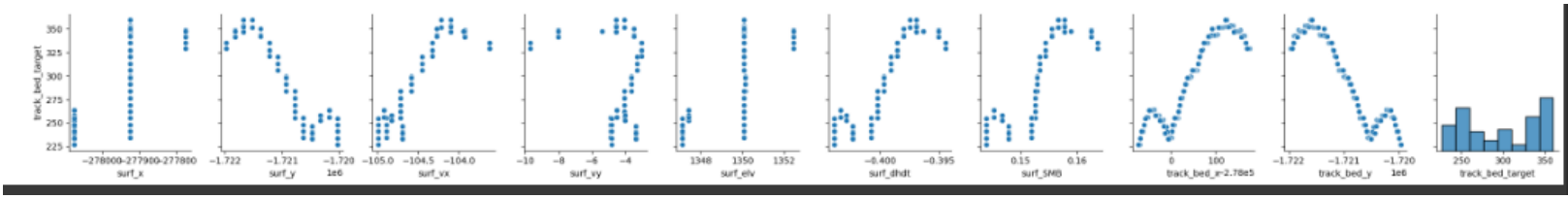
- Only plotting a subset of the training set due to the number of points in the overall dataset.

Pair Plot, [0:100] points in the training set



Note surf x&y and track_ x&y are reflections - this comes from the direction the plane was headed and the coordinates of the locations. They show the height (target) at the locations.

Pair Plot, [-100:] points in the training set



Notes the surf and target are VERY different from the other graph. The track_bed_x&y seem to more accurately reflect the actual topography.

Train Variable Ranges

surf_vx:
Max: -11.68596
Min: -497.34952
Range: 485.66355999999996

surf_vy:
Max: 154.84706
Min: -181.75491
Range: 336.60197

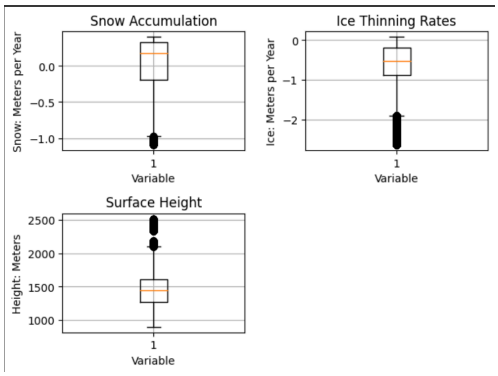
surf_elv:
Max: 2512.8472
Min: 896.68994
Range: 1616.1572600000002

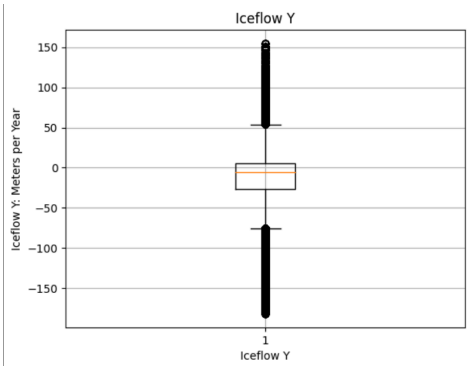
surf_dhdt:
Max: 0.07744038
Min: -2.6456227
Range: 2.72306308

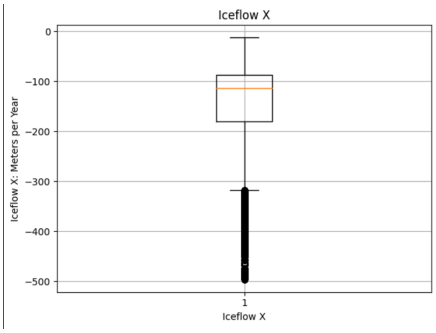
surf_SMB:
Max: 0.39795017
Min: -1.0943857
Range: 1.4923358699999998

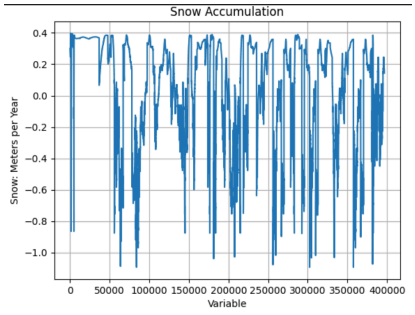
track_bed_target:
Max: 759.19482421875
Min: -671.1017456054688
Range: 1430.2965698242188

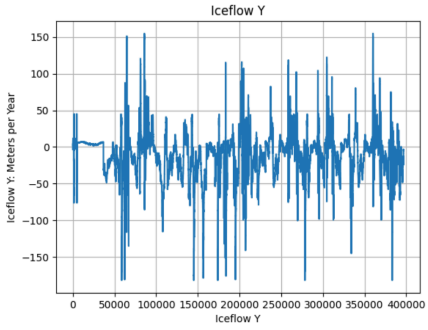
Train Variable Plots

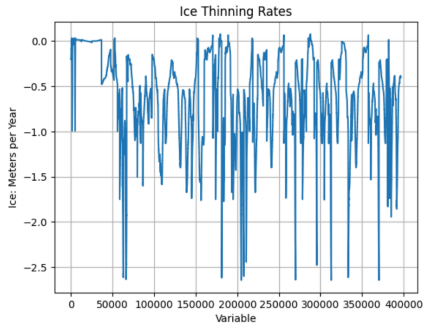


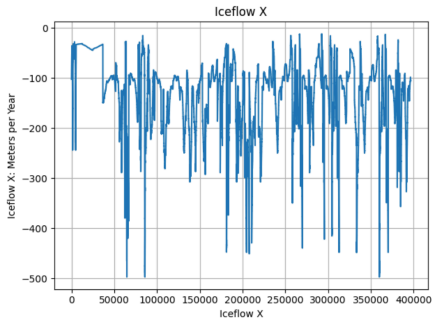


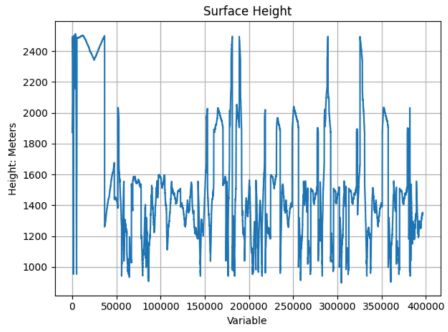












Notes: The boxplots show the height, snow, ice accumulation over the different cells. The box plots show the quartiles and medians of accumulation/thinning/height. According to the placement of the quartiles, the variables appear skewed.

Merging Data

Input Paths:
train_ = "/content/gdrive/MyDrive/REU 2023 Team 1: Ice Bed Topography Prediction/Research/Yi_Work/Data/train.csv"
y_test_ = "/content/gdrive/MyDrive/REU 2023 Team 1: Ice Bed Topography Prediction/Research/Yi_Work/Data/y_test.csv"
x_test_ = "/content/gdrive/MyDrive/REU 2023 Team 1: Ice Bed Topography Prediction/Research/Yi_Work/Data/test.csv"

Script: merging_KY_230615.ipynb
Output Path:
test_full.csv
- This is the merged test data (40% of known)

data_full.csv
- This is all of the training and test data
- This is all of the known data.

Purpose:
1. Rename the y_test column name to target
2. Merge the y_test file with the x_test file

Brainstorm Journey

Preprocessing

- Outlier Handling
  - Drop
  - IQR
  - Z-Score
- Duplicate value handling
  - Drop
- Missing value handling
  - Drop
  - Back/Forward Fill
  - Mean Fill
  - Mean Fill by region?
- Normalization/Standardization methods
- PCA (not applicable bc we have 5 features)

Processing

- Linear VS. Parallel

Dataset

- Trying alternative locations for the model to learn generalized characteristics

Modeling

1. Editing the current model
   - Apply addition layers
   - Process ||
2. Editing existing base-models
   - Adding additional layers
   - Altering parameters and hyperparameters
3. Designing a new model
   - Additional NN models
   - Combining ML & non-ML approaches
   - Adding bagging & boosting
   - Generative Adversarial Network (GAN)

Recommended by Dr. Wang & Homayra:

1. **Kriging**:
    - https://geostat-framework.readthedocs.io/projects/pykrige/en/stable/contents.html
    - https://stackoverflow.com/questions/45175201/how-can-i-interpolate-station-data-with-kriging-in-python
    - https://towardsdatascience.com/kriging-the-french-temperatures-f0389ca908dd
    - DeepKriging: Spatially Dependent Deep Neural Networks for Spatial Prediction
2. Other probabilistic models
    - https://github.com/big-data-lab-umbc/sea-ice-prediction/tree/main/probabilistic-modeling
    - Spatio Temporal Gaussian Process: https://github.com/AaltoML/spatio-temporal-GPs/tree/main
3. Deep learning models:
    - VAE, transformer

Note: it is ok to apply the normal assumption as long as it is documented.

Topography Map Visualizations

Dense_LSTM_Model…ipynb
Save as hf format

Panoply Application (downloaded)

- File > Open > open H5 file > create

HDF5 View 3.1.4

Code Review

Successful Model

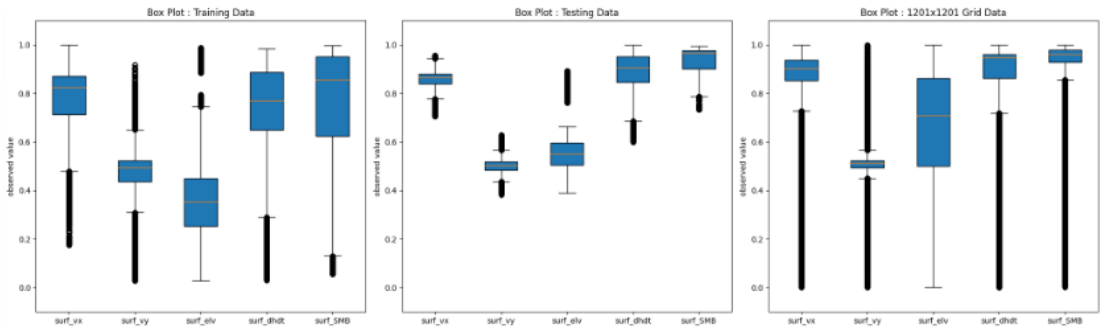
Script: Related_Work_Omar_Homayra/Copy of Dense_LSTM_Model_Ice_bed_prediction.ipynb
Resources:
https://chat.openai.com/share/f4773a57-285d-4f49-9a0c-c9664f22a697

Notes:

- rmspe: func to calculate root mean sq % error
    - Commonly used eval metric for regression task; measures average percentage difference between true and predicted.
        - It takes into account how much each prediction differs from the actual value and provides a single number that represents the overall accuracy.
    - Squares each element of relative difference
    - Gives more weight to larger errors
- rmspe_1: func to calculate root mean sq % error …. The equations are set up for rmsp residuals
    - The difference in parentheses placement means the squared relative difference is affected.
    - Used if expecting or wanting the root mean sq % residual
    - Essentially, RMSPR measures the average percentage deviation between the predicted and true values. A lower RMSPR indicates a better forecasting accuracy, as it signifies a smaller average deviation.
    - It's important to note that RMSPR is specifically *designed for time series* forecasting, where the focus is on predicting future values based on historical data.
    - BETTER TO USE RMSPE

Method:

1. Defines RMSP's
2. Read in data (train. X_test, y_test, data_1201)
3. Visualize
4. Scales using MinMaxScaler **
5. Reviz

a. 

6. Reshapes to (shape[0], 1,shape[1])
    a. Why 1 in the middle?
        i. (shape[0], 1, shape[1]): This reshaping represents each row of the topography data as a separate channel. It is commonly used when leveraging convolutional neural networks (CNNs) or models that expect input in a channel-first format. **Each row of the topography can be treated as an individual feature map, allowing the model to capture spatial relationships along the row dimension.**
        ii. By reshaping the topography data to (shape[0], 1, shape[1]), you can treat each row of the topography as a separate "channel" similar to the RGB channels in an image. This allows you to leverage the convolutional and pooling layers of a CNN for extracting spatial features from the topography data.
        iii. Topography data often has a 2D spatial structure, where each value represents the elevation or terrain height at a specific location. Reshaping the data to include the extra dimension of 1 can help preserve this spatial structure and enable models to **capture the relationships between neighboring locations.**
        iv. If your topography data represents a time series of elevation measurements or sequential observations, reshaping the data to (shape[0], 1, shape[1]) allows you to **treat each row as a separate time step.**
            1. Sequence bc plane flying gets one row after another.
7. Create model
    a. [more notes needed here]

```
input_batch = Input(shape = input_dims)

conv_model = TimeDistributed(Dense(128, activation="sigmoid"),
name='ConvL1')(input_batch)
conv_model = TimeDistributed(BatchNormalization())(conv_model)
conv_model = TimeDistributed(Dense(128, activation="sigmoid"), name='ConvL2'
)(conv_model)
conv_model = TimeDistributed(BatchNormalization())(conv_model)
conv_model = TimeDistributed(Dropout(0.5))(conv_model)
```

```
        conv_model = TimeDistributed(Dense(64, activation="sigmoid"), name='ConvL3'
)(conv_model)
        conv_model = TimeDistributed(BatchNormalization())(conv_model)
        conv_model = TimeDistributed(Dense(64, activation="sigmoid"), name='ConvL4'
)(conv_model)
        conv_model = TimeDistributed(BatchNormalization())(conv_model)
        conv_model = TimeDistributed(Dropout(0.5))(conv_model)
        conv_model = TimeDistributed(Dense(32, activation="sigmoid"), name='ConvL5'
)(conv_model)
        conv_model = TimeDistributed(BatchNormalization())(conv_model)
        conv_model = TimeDistributed(Dense(32, activation="sigmoid"), name='ConvL6'
)(conv_model)
        conv_model = TimeDistributed(BatchNormalization())(conv_model)
        conv_model = TimeDistributed(Dropout(0.5))(conv_model)

        lstm_network = LSTM(64, return_sequences=True)(conv_model)
        lstm_network = keras.layers.LSTM(32, return_sequences=False)(lstm_network)
        lstm_network = Dropout(0.5)(lstm_network)
        lstm_network = keras.layers.Dense(32, activation='sigmoid',
name='Dense1')(lstm_network)
        lstm_network = keras.layers.Dense(1, activation='linear',
name='Dense2')(lstm_network)
```

8. Get the model
9. Running model.summary()
10. Compile model with adam optimizer and loss = mean standard error
    a. Estimate the variability or uncertainty associated with the sample mean
    b. std / (sqrt(sample size))
11. Fit model on training data
    a. 200 epochs
    b. 5000 batch size
    c. 1 verbose
        i. What is this? Refers to the level of detail or amount of information that the model displays during the training process. Useful for monitoring and understanding the model's behavior and performance. Higher the value, the more information is displayed. Standard:
            1. verbose=0: No information is displayed during the training process.
            2. verbose=1: Progress bar or log messages are displayed, typically showing the training progress for each epoch.
            3. verbose=2: More detailed information is displayed, such as the loss and metrics for each batch or epoch.
        ii. KEEP VERBOSE

   d. 30% validation
   e. Shuffle = True
      i. What does this do? Common to randomly shuffle the training data to introduce randomness and prevent any inherent order or patterns in the data from influencing the model's learning process. Can improve generalization and reduce the risk of overfitting.
      ii. CONTINUE WITH SHUFFLE
   f. Callbacks=[callbacks.EarlyStopping(monitor='val_loss', patience=100, verbose=2, mode = 'auto)])
      i. What is the early stopping? Early stopping here is what it sounds like. If the model stops improving (according to monitor) at patience num, then the model stops training so reduce overfitting.
      ii. What is a monitor? Evaluated at every epoch. If this stops improving or deteriorates of a predefined number of epochs (patience), training is stopped early.
      iii. What is patience? Predefined number of epochs to stop at. See def of monitor above.
      iv. What is mode? Set to maximize, minimize or automatic where it infers max/min by the monitor type.
      v. CONTINUE USING EARLY STOPPING to prevent overfitting.
12. Use the model to predict on the training – y_pred_train
13. scaler_Y.inverse_transform(y_pred_train)
   a. What does this do? Commonly used with normalized data. This reverts the normalized values back to their original values.
14. Get the training error and r^2 score
15. Plot
16. Repeat predict, scale, get error, plot for test_pred
17. Predict 1201 data
18. Transform 1201 data with inverse_transform
19. Flatten
20. Reshape to 1201,1201
21. (Create dataset as hf file)
22. Plot 2D 1201 data set
   a. plt.imshow(np.transpose(2D_1201_pred), cmap= "seismic")
23. —------
24. New interpolation data…
   a. Repeat all

Probabilistic Models

1. https://github.com/big-data-lab-umbc/sea-ice-prediction/tree/main/probabilistic-modeling
2. Spatio Temporal Gaussian Process: https://github.com/AaltoML/spatio-temporal-GPs/tree/main

Gaussian Process Regression

Link: https://github.com/big-data-lab-umbc/sea-ice-prediction/tree/main/probabilistic-modeling
2022 Benchmarking Probabilistic Machine Learning Models for Arctic Sea Ice

Background: From Arctic sea ice prediction on sea-ice extent with lowest RSME.
Method:

- Averaging data, splitting then into model
- Model Baseline: MLR → LSTM (2 many to many and many to 1 including 1 dropout layer; adam optimizer)
- Model Probabilistic: HMM, GRP, GRNN

Models Explained

1. MLR: regression technique, used with more than 1 independent variable to prediction
    a. Not probabilistic
    b. Indirectly uncertainty
2. HMM: sequence prediction; capable of observing unknown facts using Markov
    a. Probabilistic
    b. Indirectly uncertainty
    c. Able to observe hidden events (features)
    d. Uses forward-backward prediction
    e. Number of hidden states = num features
3. *** GPR: regression technique, computes predictive distribution using bayesian
    a. Probabilistic
    b. Uncertain
    c. Uses bayes underneath the hood and optimize log-marginal-likelihood (LML)
    d. Must repeat to avoid local optima.
4. GRNN (general regression NN): regression technique, non-parametric to provide est of continuous variable and function approx
    a. Probabilistic
    b. Uncertain
    c. Uses prob density function (PDF) & extra layer of summation and weights between hidden and out layer
5. LSTM: sequence prediction; predicts future values of sequence data

Ideas

- Try MLR
- Try GMM
- Try GPR
- Try GRNN
- Try the HMM layer inside of the traditional model followed (a layer? then) by dropouts

Relevant Implement

```
import sklearn.gaussian_process as gp
from sklearn.metrics import r2_score

"kernel = gp.kernels.ConstantKernel(1.0, (1e-1, 1e3)) * gp.kernels.RBF(10.0, (1e-3, 1e3))"
"model = gp.GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=10, alpha=0.1,
normalize_y=True)"
"model.fit(x_train, y_train)\n",
"params = model.kernel_.get_params()"
"y_pred, std = model.predict(x_test, return_std=True)"
"y_train_pred, std = model.predict(x_train, return_std=True)"
      "%matplotlib inline\n",
      "#plot \n",
      "plt.plot(y_test, color='red')\n",
      "plt.plot(y_pred)\n",
      "\n",
      "plt.legend(['y_test','y_pred'])\n",
      "plt.show()\n"
rmse1 = np.sqrt(np.mean((inv_y_pred - inv_y_test) ** 2))
rmse1
nrmse1 = rmse1/(np.mean(inv_y_test))
nrmse1
r2_score1 = r2_score(inv_y_test, inv_y_pred)
```

Important Notes:

- Must implement with mini-batching because of time-cost ($$$$)
- Ran on subsets of the data because of the extremely high RAM costs

Slides

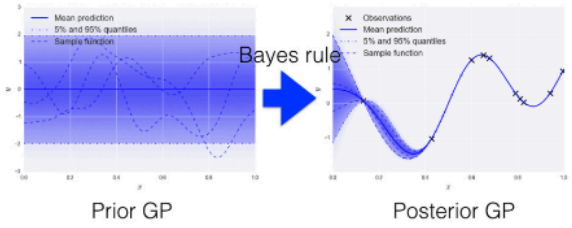
Katie Progress

Gaussian Process Regression (GPR)

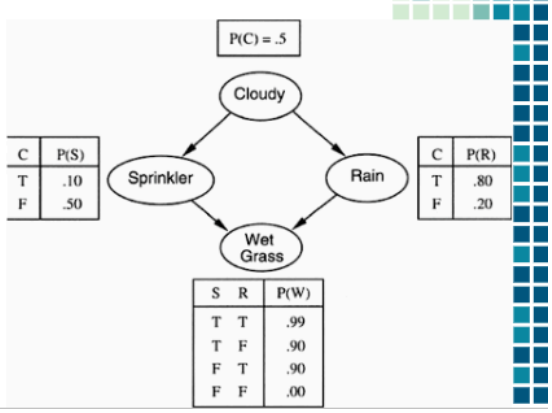
Regression technique

Computes predictive distributions using bayes with the bayesian network

Must be cautious of local optima



Prior GP Posterior GP



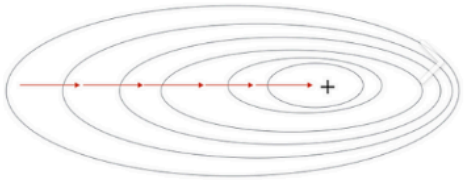
Katie Progress

We implemented GPR with mini-batch....

Mini-batching: training the model in small groups (batch)

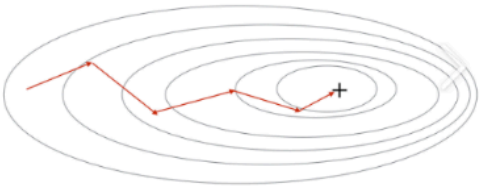
How data is processed to get to the target

Gradient Descent



$$$$

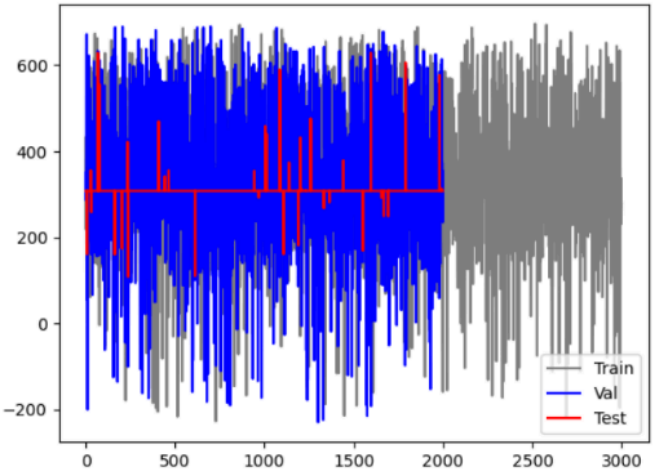
Mini-Batch Gradient Descent

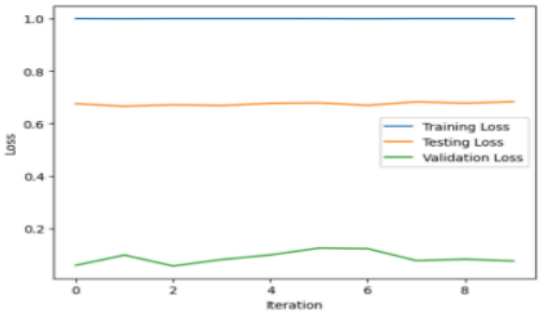


$-$$$

Katie Progress

GPR with subset 5000, 200 batch, 10 epochs





No loss = not improving

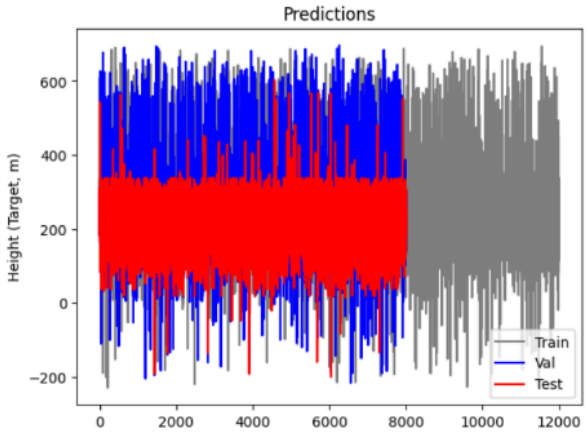
Great predictions from the validation (expected)
Poor prediction for the test :(

Testing Results

```
RMSE: 148.69296021013707
RMSE Percentage: 1134.5879179979088

Mean Absolute Error: 107.29178531006903
Mean Absolute Percentage Error: 1.0629395396306576
R^2 Score: 0.010190917861486959
```

GPR with subset 20,000, 2000 batch, 10 epoch



Testing Results

```
RMSE: 77.44198696844369
RMSE Percentage: 1051.910956806744
RMSE Percentage-1: 519.9708255978687
Mean Absolute Error: 30.65038559540605
Mean Absolute Percentage Error: 0.46369726423267804
R^2 Score: 0.4839325053385697
```

CPU times: user 2min 46s, sys: 34.3 s,
total: 3min 21s
Wall time: 2min 2s $

1. Great increase in the R^2 :)
2. Great decrease in the % error :)

… more data → better prediction results?

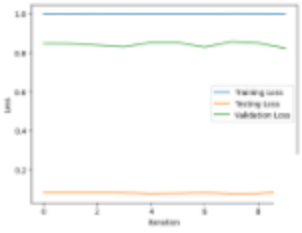
Katie Progress

GPR with subset 50,000, 5000 batch, 10 epochs

1. R^2 improves :)
2. Error decreases :)

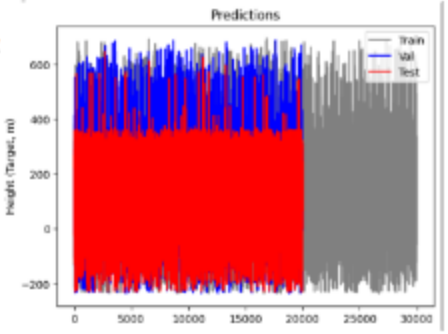
Just as good as LSTM+Dense for subset

3. Expensive @37 min :(
4. Model does not learn with more training (epochs) :(



CPU times: user 36min 12s, sys: 1min 39s, **total: 37min 51s**
Wall time: 23min 15s

$$$$




```
RMSE: 53.35848702533058
RMSE Percentage: 581.4732553302155

Mean Absolute Error: 16.74975621866551
Mean Absolute Percentage Error: 0.2576540431830953
R^2 Score: 0.8087017735706241
```

Katie Progress

GPR Summary

More data into the model → better predictions

More data = $$$$ ~ 9 hours

Model is not improving with more training... it is consistent

Thought: Model will have challenges predicting brand new areas because it is not learning patterns it is only adapting to what is it currently trained on.

Next: run entire set on taki

Results: R^2 & RSME are promising for the subset
Next: need to run on taki with entire dataset

Result #1: 50,000 Subset, Statistics Only

June 26, 2023

File: gpr_fortaki_ky_230623.py
Slurm:

- 1 node
- Exclusive
- Hpcf2013
- Normal+ qos
- 4 hour time
- Batch partition

Parameters:

- 90-10 train-test
- 20% validation
- Dropping surf
- StandardScaler
- 5000 batch
- 10 epochs

Test Results (rounded 3 decimal places)

Run	Seed	RSME	R^2 Score	RSME %	Mean Abs Error	MAE%
7	151	47.767	0.850	122.987	15.470	0.159
10	206	48.023	0.846	655.445	14.923	0.230
6	345	48.279	0.845	1773.665	14.867	0.462
3	888	49.456	0.842	288.505	15.399	0.189
8	151	51.243	0.826	173.372	15.920	0.173
2	368	51.807	0.825	372.014	15.829	0.259
11	880	52.070	0.816	253.415	16.194	0.220
9	566	53.315	0.809	166.655	16.543	0.175
5	577	55.580	0.803	347.169	17.832	0.222
4	167	55.715	0.787	1280.023	16.918	0.382
1	471	57.070	0.783	220.510	17.916	0.217
AGG		51.848	0.821	513.978	16.165	0.244

Results #2: 50,000 Subset, Statistics Only

File: gpr_fortaki_ky_230623.py
Sbatch 6/28 @2:24PM, gpr_50, 12834503, node 113
Slurm:

- Batch
- 1 node
- Exclusive
- Hpcf2013
  - Each node has 64 GB of memory.
- Normal+ qos
- 4 hour time
- Batch partition

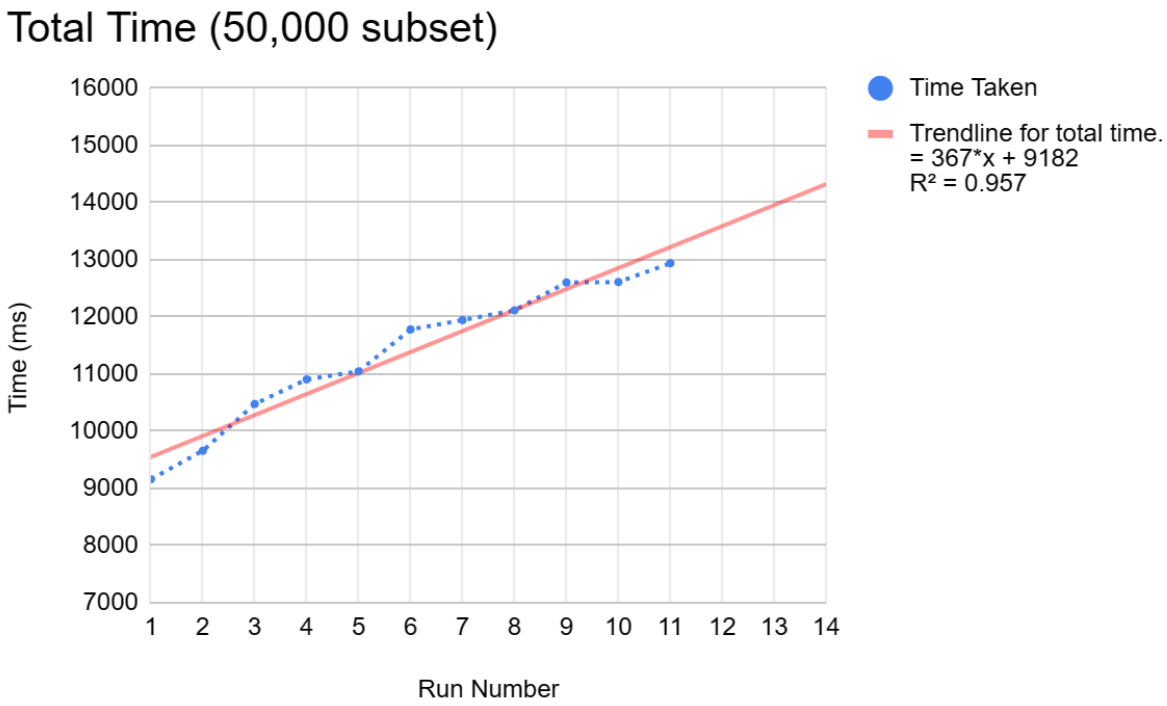
Parameters:

- 90-10 train-test
- 20% validation
- Dropping surf
- StandardScaler
- 5000 batch
- 10 epochs

Test Results (rounded 3 decimal places)

Run	Seed	Total Time (ms)	RSME	R^2 Score	RSME %	Mean Abs Error	MAE%
10	415	12609.934	48.198	0.845	391.309	14.801	0.219
1	918	9159.364	48.672	0.836	94.060	15.223	0.126
4	532	10906.501	49.074	0.834	639.076	15.973	0.309
9	287	12602.565	49.591	0.822	25949.884	15.593	3.910
5	99	11046.971	49.674	0.837	664.674	15.500	0.261
3	413	10476.788	50.942	0.818	454.212	15.086	0.271
8	42	12114.941	51.546	0.800	111.640	16.220	0.157
7	72	11943.837	52.169	0.811	199.725	16.167	0.181
11	214	12938.941	53.556	0.811	420.443	16.246	0.259
6	220	11781.133	53.611	0.809	102.215	16.914	0.147
2	161	9658.282	57.526	0.774	192.543	17.356	0.165
AGG	NA	11385.387	51.324	0.818	2656.344	15.916	0.546

11385.387ms = ~ 11 seconds …. 5-6 runs per minute. Should take abt 2 minutes to run.
Why is the time increasing for each run when it is running the exact same py file each time?


Total Time (50,000 subset)
16000
15000
14000
13000
12000
11000
10000
9000
8000
7000
Time (ms)
1
2
3
4
5
6
7
8
9
10
11
12
13
14
Run Number
Time Taken
Trendline for total time.
= 367*x + 9182
R² = 0.957

Results #3: full data train

File: gpr_fullTrainOnly_ky_230629.py
Sbatch 6/29 @4:13PM, gpr_TTOnly, 12897564, cnode[032-033]
Slurm:

- #SBATCH --partition=high_mem
  - Each node has 384 GB of memory
- #SBATCH --nodes=2
- #SBATCH --qos=normal+
- #SBATCH --time=4:00:00
- [not exclusive]

Params:

- 90-10
- Dropping surf
- StandardScaler
- 10000 batch
- 1 epochs

Sbatch 6/29 @4:31PM, gpr_TTOnly, 12897572, cnode[032-033]
Slurm:

- #SBATCH --partition=high_mem
  - Each node has 384 GB of memory
- #SBATCH --nodes=2
- #SBATCH --qos=normal+
- #SBATCH --time=4:00:00
- [not exclusive]
- CANCELS JOB IF ERRORS OUT OR GETS COMPLETED AND PRINTS A RETURN LINE

Params:

- 90-10
- Dropping surf
- StandardScaler
- 10000 batch
- 1 epochs

Never ran????

Sbatch: 12905304 high_mem gpr_TTOn kyi5 PD 0:00 2 (Priority) cnode[033,039]
#SBATCH --job-name=gpr_TTOnly
#SBATCH --partition=high_mem
#SBATCH --nodes=2
#SBATCH --qos=medium+
#SBATCH --time=06:00:00
Node sat open, code did not run?!?!?

Did not get any err or out again. Access node error again? Try running it on the batch nodes again??? (Not predicting 1201 so I should not need 100GB of storage allocated for 1201.)
Sbatch 7/5 @10:20AM, gpr_TTOnly,
12956002 batch gpr_TTOn kyi5 R 0:00 4 cnode[132-133,141-142]

- #SBATCH --job-name=gpr_TTOnly
- #SBATCH --partition=batch
- #SBATCH --nodes=4
- #SBATCH --qos=medium+
- #SBATCH --time=06:00:00

If I still get really poor output…
Calculate training R^2 and RSME
Then adjust…. Probably reduce the split ratio to reduce overfitting and improve the predictions?

Results #4: 50,000 train & 1201 data predicted to h5

File: gpr_for1201_ky_230627.py
Sbatch 6/27 @2:24PM, gpr_50h5, 12834504, node 149
Slurm:

- 1 node
- Hpcf2013
- Normal+ qos
- 4 hour time
- [Note] not exclusive due to testing file.
- Batch partition

Parameters:

- 90-10 train-test
- 20% validation
- Dropping surf
- StandardScaler
- 10000 batch
- 1 epochs

Test Results (rounded 3 decimal places)
Out of Space Error - I need more nodes

Sbatch 6/27 @7:23PM, gpr_50h5, 12834553, node cnode[128-131]
Slurm:

- 4 node
- Hpcf2013
- medium+ qos
- 4 hour time
- Batch partition
- [Note] not exclusive due to testing file.

Time limit reached. Node terminated. (populated in the error file, but the code never actually ran???)
Run this and predict 1201 data on more nodes on a longer time allocation.

Sbatch 6/28 @9:57AM, gpr_50h5_space+,12834812, node cnode[128-133]

- 6 node
- Hpcf2013
    - Each node has 64 GB of memory… I was running 6 nodes.
- medium+ qos
- 6 hour time
- Not exclusive
- Batch partition

BROKE! AHHHH! Err file was only loading in the py module and then a cancellation cuz time was reached.
Rerun it? Idk…. Break it down and test parts.

Results #5: full data & 1201 data predicted to h5

June 27, 2023

Sbatch @2:24PM ET, gpr_full, 12834505, node118
Slurm:

- 1 node
- Exclusive
- Hpcf2013
- medium+ qos
- 10 hour time
- Batch partition

Parameters:

- 90-10 train-test
- 20% validation
- Dropping surf
- StandardScaler
- 5000 batch
- 10 epochs

Test Results (rounded 3 decimal places)
[include time taken]
Node Killed. Out of Space.

[WAITING – pending results either tune hyper params OR abandon]
Expecting out of space error. Starting a second sbatch with more nodes.
Slurm: 7:30PM, 12834555, cnode[135-138]

- 4 node
- Exclusive
- Hpcf2013
- medium+ qos
- 10 hour time
- Batch partition

Node Killed. Out of Space Error. I think there was a slurm memory param that was defaulting. Try rerunning with no memory removed.

Slurm: 6/28 @10:00AM, 12834812, cnode[118-119,122-123]

- 4 node
- Exclusive
- Hpcf2013
- medium+ qos
- 10 hour time
- Batch partition

6 hours err file was stuck at ~200 size. At hour 6, jumped to 8705. The error is a known populating error which means the file is running. Maybe we will get an output!? 4 hours left to go on the nodes.
slurmstepd: error: *** JOB 12834817 ON cnode118 CANCELLED AT 2023-06-28T22:59:40 DUE TO TIME LIMIT ***
slurmstepd: error: Detected 244 oom-kill event(s) in step 12834817.batch cgroup. Some of your processes may have been killed by the cgroup out-of-memory handler.
I think the process couldn't finish cuz it didn't start till after 6 hours.

Slurm: 6/28 @8:05PM (entered queue)... sat in the queue 5 hours... didn't start running till 6/29 @1:43AM, 12848209, nodes cnode[118-119,123,130,133,140]

- 6 node (Priority)
- Exclusive
    - Common for exclusive nodes to have access to the full memory capacity available on the node
- Hpcf2013
- medium+ qos
- 12 hour time
- Batch partition

numpy.core._exceptions.MemoryError: Unable to allocate 101. GiB for an array with shape (1442401, 9435) and data type float64

—----------

Code needs to be rewritten to predict in batches, concatenated, and thennnnn converted to h5 and exported.

—----------

File:
Slurm XX/XX @XX:XXAM/PM, runName, jobID, nodes
Slurm:

- High_mem
    - Each node has 384 GB of memory

Parameters:

Slides 2

GPR

Brought to you by the absurd amount of coffee I have consumed this week & dogs.

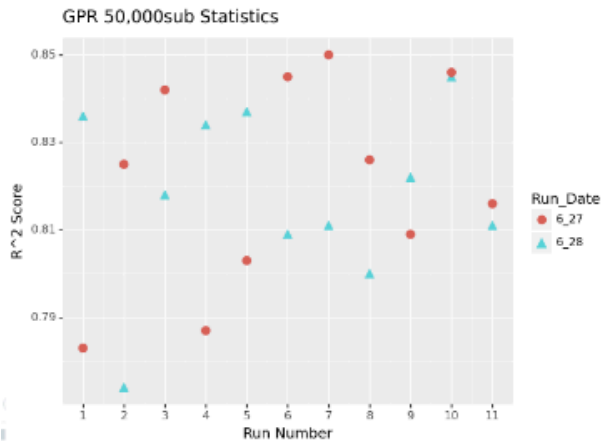
GPR on Taki: 50,000 sub

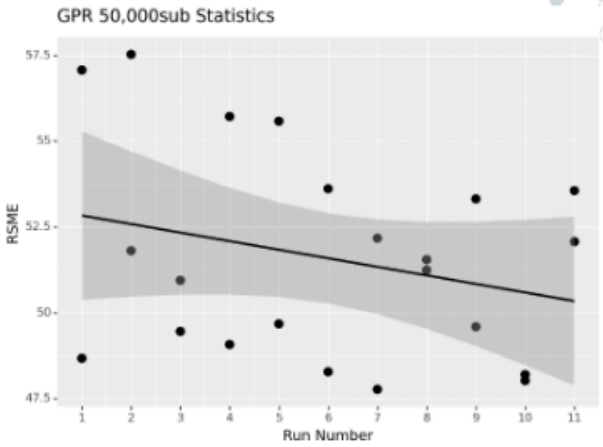
Run_Date	Run	Seed	Total Time (ms)	RSME	R^2 Score	RSME %	Mean Abs Error	MAE%
AGG	NA	Random Generate	11385.387	51.586	0.820	1585.161	16.040	0.395

GPR on Taki

Summary: Consistently scoring & performing well on subset

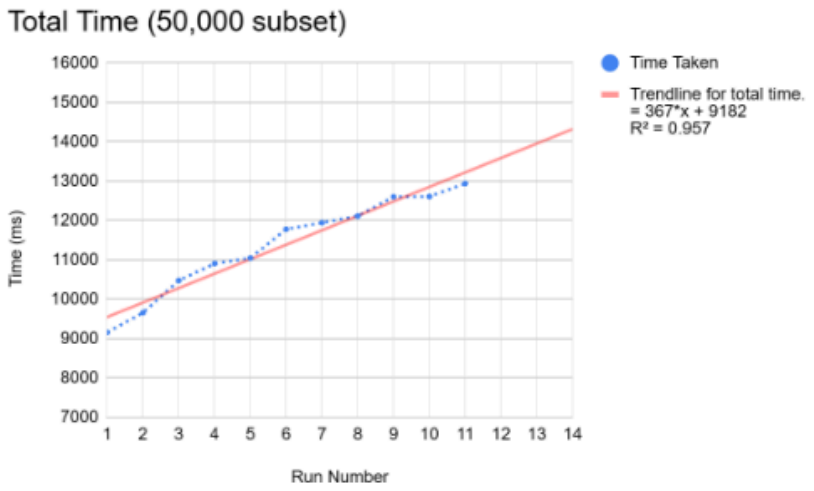
Tooling: Generated with plotnine ggplot





Total Time to Run on taki

11385.387ms = ~ 11 seconds + 5 runs per minute = Should take about 2 minutes to run 50,000 training and testing.



Next Steps

Get the full model working,
predict unseen data, visualize

GPR Next Steps

Get entire data set working on taki
Train full unseen data
Convert to h5 files
Visualize them in Panoply

ipped in slideshow mode

Spatio Temporal Gaussian Process

Link: 2111.01732.pdf (arxiv.org)

Abstract

We introduce a scalable approach to Gaussian process inference that combines **spatio-temporal filtering with natural gradient variational inference**, resulting in a non-conjugate GP method for multivariate data that **scales linearly** with respect to time. Our natural gradient approach enables application of parallel filtering and smoothing, further reducing the temporal span complexity to be logarithmic in the number of time steps. We **derive a sparse approximation that constructs a state-space model over a reduced set of spatial inducing points**, and show that for separable Markov kernels the full and sparse cases exactly recover the standard variational GP, whilst exhibiting *favorable computational properties*. To further improve the spatial scaling we propose a ***mean-field assumption of independence*** between spatial locations which, when coupled with sparsity and parallelisation, leads to an efficient and accurate method for large spatio-temporal problems.

Definitions

Spatiotemporal: Spatiotemporal, or spatial temporal, relates to space and time. Spatial refers to space and temporal refers to time. It describes a phenomenon in a certain location and time — for example, shipping movements across a geographic area over time (see above example image). A person uses spatial-temporal reasoning to solve multi-step problems by envisioning how objects move in space and time.

- What is Spatial Temporal? Definition and Related FAQs | HEAVY.AI
- Is spatiotemporal very relevant given we are not trying to predict the future landscape? We are just trying to predict at the moment.

Variational inference (VI): a popular method in machine learning that uses optimization techniques to estimate complex probability densities. Converges faster than classical methods. Works by choosing a family of probability density functions and then finding the one closest to the actual probability density -- often using the Kullback-Leibler (KL) divergence as the optimization metric.

- [2108.13083] An Introduction to Variational Inference (arxiv.org)

My thoughts

- GP's combined with variational inference to use GP's but have a better processing time.

WAITING TO CONTINUE

Implement

Inputs:

data_full_ = mainPath + '/Data/data_full.csv' #training and test combined
data_1201_ = mainPath + '/Data/df_1201_validation_data.csv'

Script: Scripts/GP_KY_230621.ipynb
Outputs:

Notes:

- Running the GPR must use batches
    - Takes forever to run even for 1 epoch
    - Also note the setup is running at least an O(n^2) time.
    - Running 10000 batch with 1 epoch takes: ____ time.
    - 632706 training size
- Continued running with subsets of the training
    - Continue this tomorrow

https://github.com/AaltoML/spatio-temporal-GPs/blob/main/experiments/nyc_crime/models/m_gpflow.py

https://www.statology.org/content-validity/
CVI stands for Content Validity Index and is used to calculate the content validity of a test or questionnaire12.
The CVI is the mean content validity ratio of all questions on a test, and the closer it is to 1, the higher the overall content validity of a test

In each situation, content validity can help determine if a test covers all aspects of the construct that it sets out to measure.

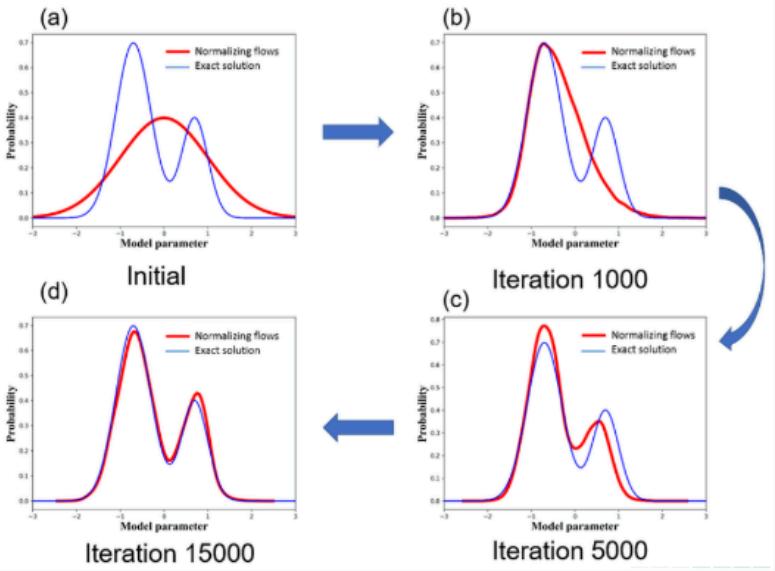
Content Validity Ratio = (ne – N/2) / (N/2)
ne: The number of subject matter experts indicating “essential”
N: The total number of SME panelists
subject matter expert (SME)

Variational Inference (VI)

Estimates complex probability densities

Converges faster than traditional methods

Chooses family of pdfs are find the closest to true pdf

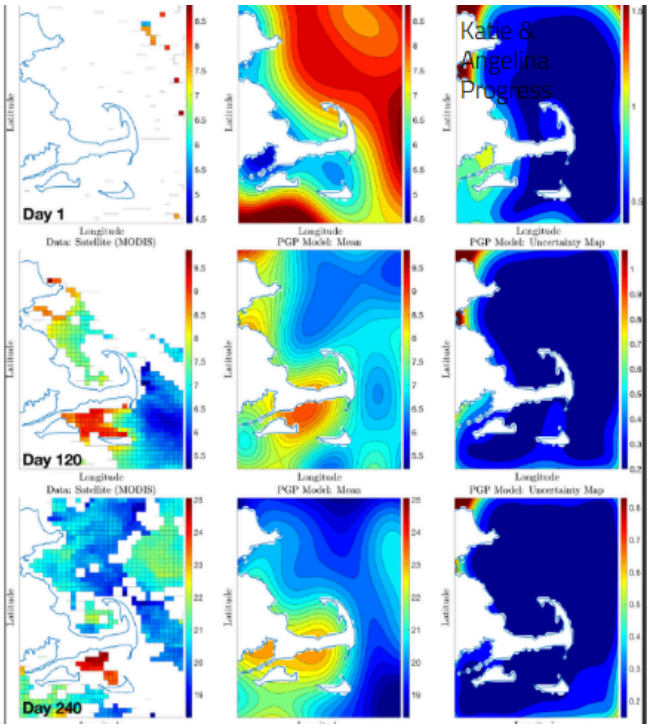


Spatio Temporal Gaussian Process

Uses VI

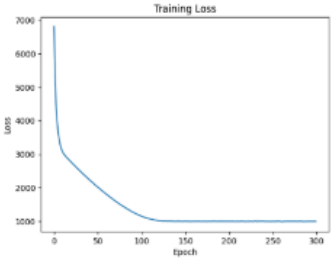
Relates to time & space

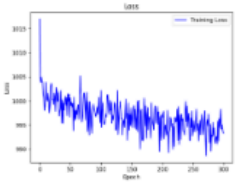
Used to solve multi-step problems by envisioning movement

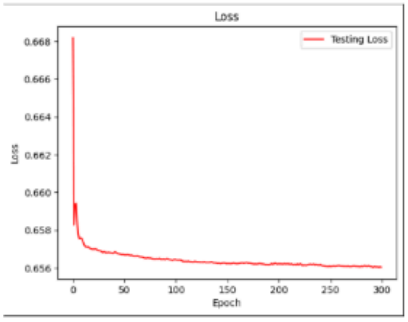


Katie & Angelina Progress

STGP with full data, 1500 batch, 300 epochs; MinMax Scaling; 90-10 train-test split







While Scaled

```
RMSE: 0.06460067396209421
RMSE Percentage: 45.44139455654972
RMSE Percentage-1: 13.962888819203153
Mean Absolute Error: 0.04733973503110621
Mean Absolute Percentage Error: 0.150952606436013
R^2 Score: 0.25118865550540714
```

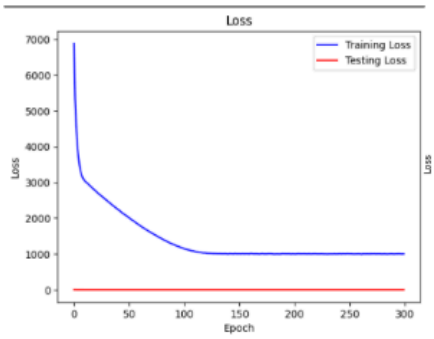
Reverted to Normal Scale

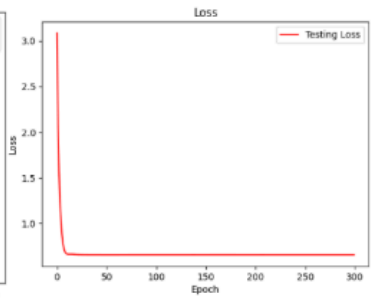
```
RMSE: 155.46955360199044
RMSE Percentage: 21709.58350863823
RMSE Percentage-1: nan
Mean Absolute Percentage Error: 113.9289580359666
Mean Absolute Percentage Error: 2.9532354060676666
R^2 Score: 0.25118865550540737
```

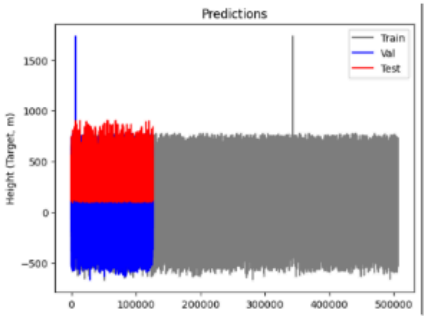
Model is learning during training although severely overfitting
total time: ~7 min

Katie & Angelina Progress

STGP with full data, 1500 batch, 300 epochs; MinMax Scaling; 80-20 train-test split



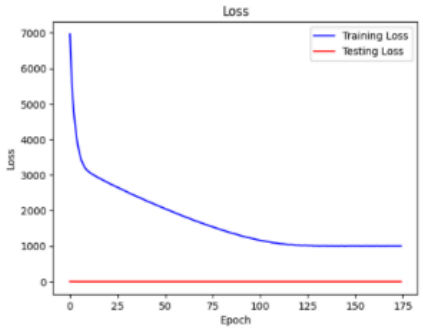


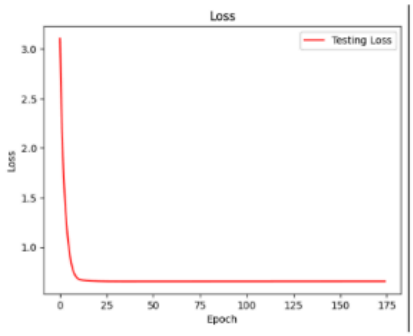


Total time: 12 min

Katie & Angelina Progress

STGP with full data, 1500 batch, 175 epochs; MinMax Scaling; 60-40 train-test split





Total time: 13 min

```
RMSE: 0.06403802292479678
RMSE Percentage: 46.60758433763885
RMSE Percentage-1: 13.87349744763906
Mean Absolute Error: 0.046900032760228076
Mean Absolute Percentage Error: 0.1503264823794649
R^2 Score: 0.2647378376542263
```

Katie & Angelina Progress

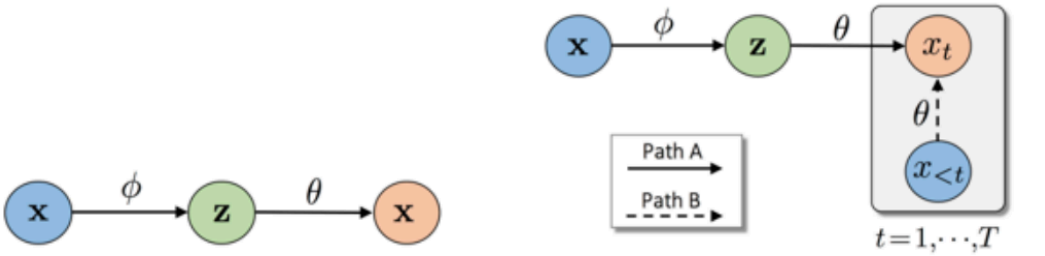
VAE

Deep Learning Methodology
Deep generative model

Background

Microsoft

Less pain, more gain: A simple method for VAE training with less of that KL-vanishing agony - Microsoft Research

- Learn prob representations (z) of natural languages (x) by (1) reconstructing features z and (2) KL regularization to leverage prior knowledge
- $$\mathcal{L}_\beta = \underbrace{\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\right]}_{\text{Reconstruction}} - \beta \underbrace{\mathrm{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}))}_{\text{KL Regularization}}$$
- 

(a) Standard VAE (b) VAE with an auto-regressive decoder
- Provide latent z at beginning - trained under full VAE
  - Learn using cyclical schedule which repeats the monotonic schedule
    - Monotonic: standard in training text VAE
    - Cyclical schedule improves performance each cycle
  - Trained to reconstruct x

Towards Data Science

*Understanding Variational Autoencoders (VAEs) | by Joseph Rocca | Towards Data Science

- Deep Generative Model
- “A variational autoencoder can be defined as being an autoencoder whose training is regularized to avoid overfitting and ensure that the latent space has good properties that enable generative process.”
- Encoding is regularized in training to help produce better data
- Variational coming from relation of regularization and variational inference (VI)
- Z is random var; p(z) is prob of random variable

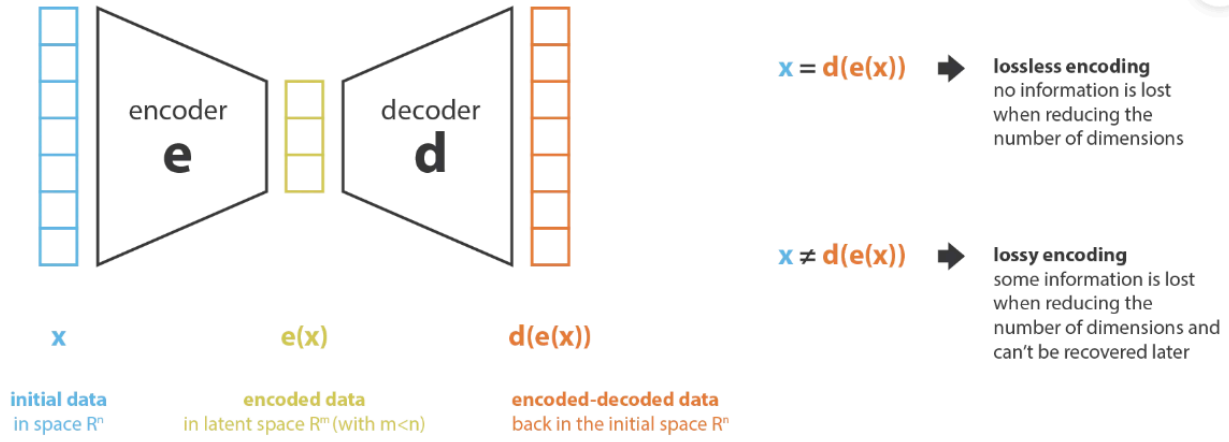
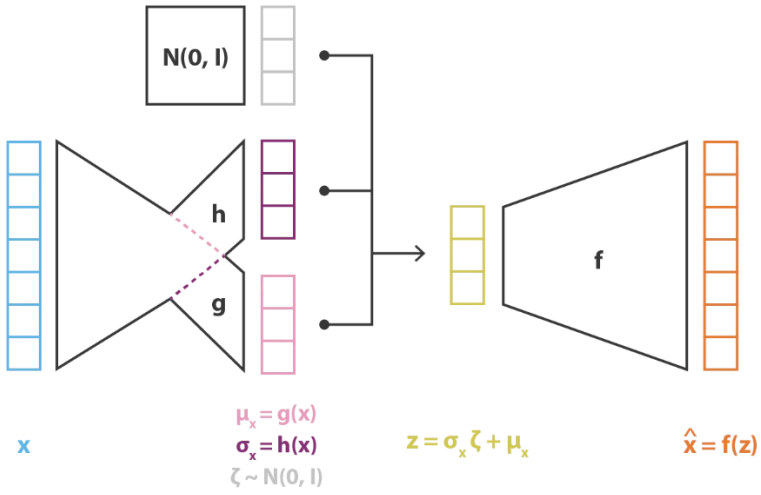


Illustration of the dimensionality reduction principle with encoder and decoder.

-
    - Maximize info kept in encoding
    - Minimum reconstruction error decoding
- More complex architecture → can process higher dim data with lower error
- Must regularize to keep from overfitting

Directly from Towards Data Science

1. Input is encoded as distribution over the latent space
2. A point from the latent space is sampled from that distribution
3. The sampled point is decoded and the reconstruction error can be computed
4. The reconstruction error is back propagated through the network



$$\text{loss} = C \,||\, x - \hat{x} \,||^2 + KL[\, N(\mu_x, \sigma_x), N(0, I) \,] = C \,||\, x - f(z) \,||^2 + KL[\, N(g(x), h(x)), N(0, I) \,]$$

Variational Autoencoders representation.

Keras

Variational AutoEncoder (keras.io)

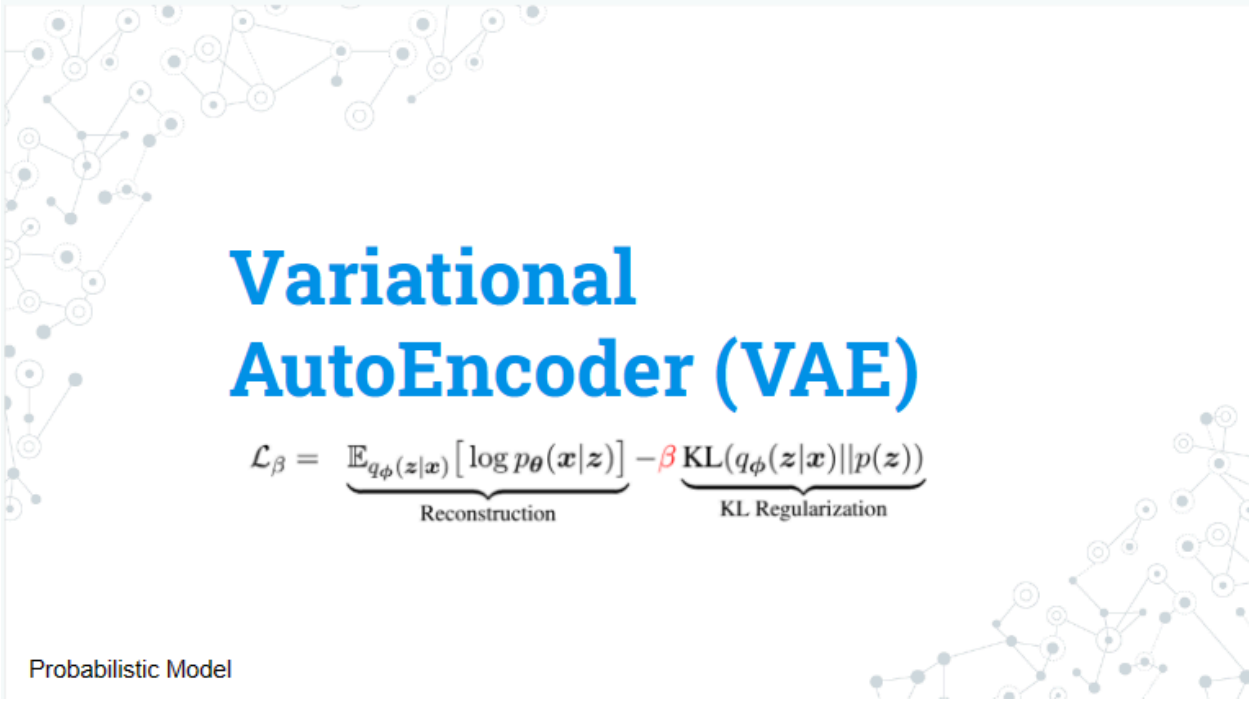
- Example code and differentiation into classes
- Local Saved Example Code

GPT

VAEs are generative models and not specifically designed for regression tasks, R^2 might not be the most appropriate metric to evaluate its performance.

Beginner guide to Variational Autoencoders (VAE) with PyTorch Lightning | by Reo Neo | Towards Data Science
Variational Autoencoders (VAEs) for Dummies-Step By Step Tutorial | Towards Data Science

Slides



AutoEncoder

"Autoencoders is pretty simple and consists in setting an encoder and a decoder as neural networks and to learn the best encoding-decoding scheme using an iterative optimisation process"

– Towards Data Science

Uses backpropagation of error to update weights

Backpropagation

Purdue CS373 &

Research Gate

Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-b)}$$

$$b = \sum_{j=0}^{D} \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$$

Backward

$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

$$\frac{dJ}{db} = \frac{dJ}{dy}\frac{dy}{db}, \frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$$

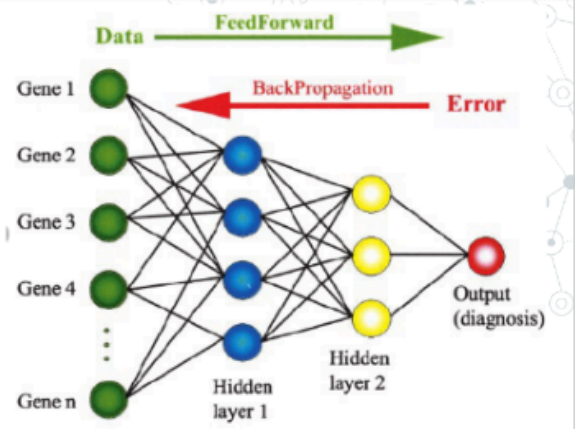
$$\frac{dJ}{d\beta_j} = \frac{dJ}{db}\frac{db}{d\beta_j}, \frac{db}{d\beta_j} = z_j$$

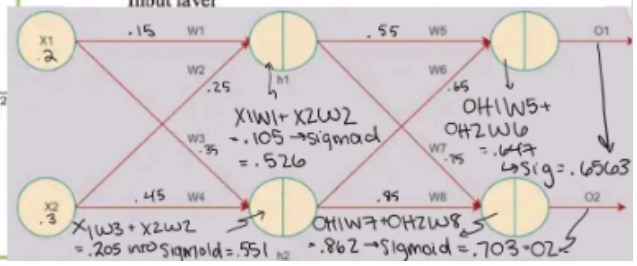
$$\frac{dJ}{dz_j} = \frac{dJ}{db}\frac{db}{dz_j}, \frac{db}{dz_j} = \beta_j$$

$$\frac{dJ}{da_j} = \frac{dJ}{dz_j}\frac{dz_j}{da_j}, \frac{dz_j}{da_j} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2}$$

$$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j}\frac{da_j}{d\alpha_{ji}}, \frac{da_j}{d\alpha_{ji}} = x_i$$

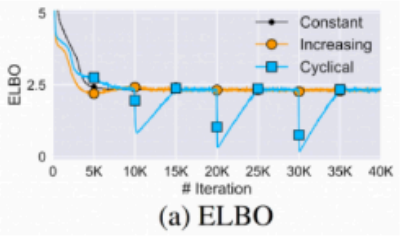
$$\frac{dJ}{dx_i} = \frac{dJ}{da_j}\frac{da_j}{dx_i}, \frac{da_j}{dx_i} = \sum_{j=0}^{D} \alpha_{ji}$$



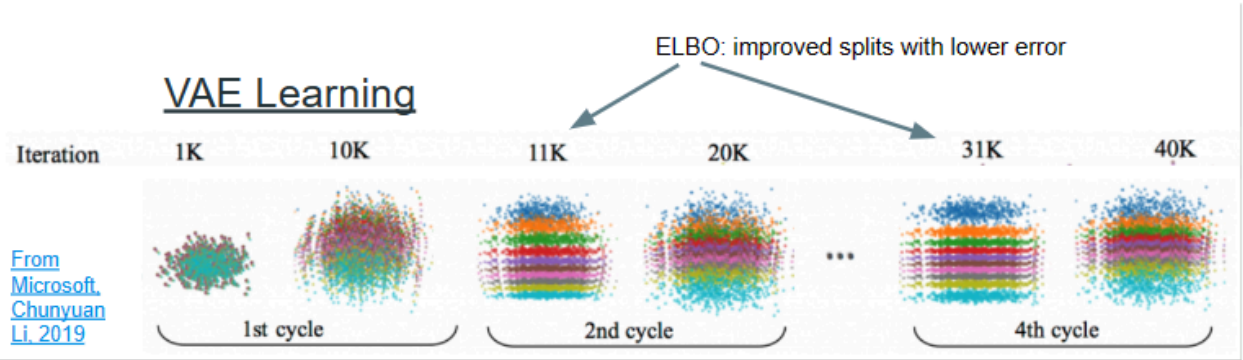


What is VAE?

Learns probability representation of data by (1) reconstructing features and (2) using prior knowledge through a learning cycle.



(a) ELBO



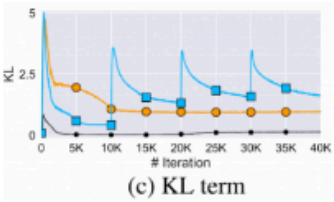
From Microsoft, Chunyuan Li, 2019

Why do we need to regularize (KL)?

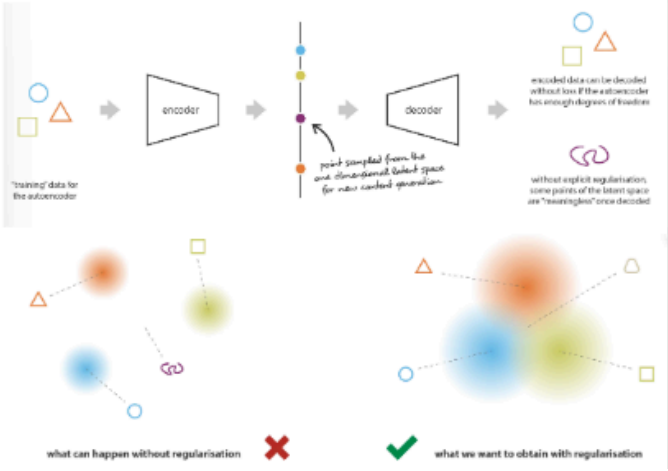
Expressed as Kullback-Leibler divergence (KL)

Regularization from returned distribution & standard Gaussian

Why is regularization important? This keeps the model from overfitting and outputting gibberish.



(c) KL term



Regularization Assumptions

1. Continuity: 2 points close to each other should not give 2 completely different contents once encoded

   Aka: similar points should produce similar encodings

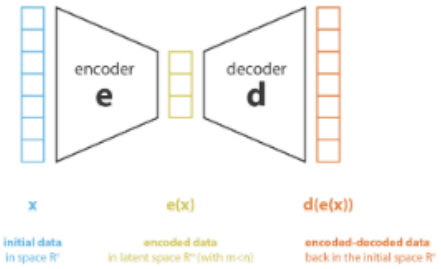
2. Completeness: a point from the space should give meaningful content

   Aka: Don't feed the space garbage

How does it work?

Directly from Towards Data Science

1. Input is encoded as distribution over the latent space
2. A point from the latent space is sampled from that distribution
3. The sampled point is decoded and the reconstruction error can be computed
4. The reconstruction error is back propagated through the network



Results

VAE Results

VAE works better with simpler model.

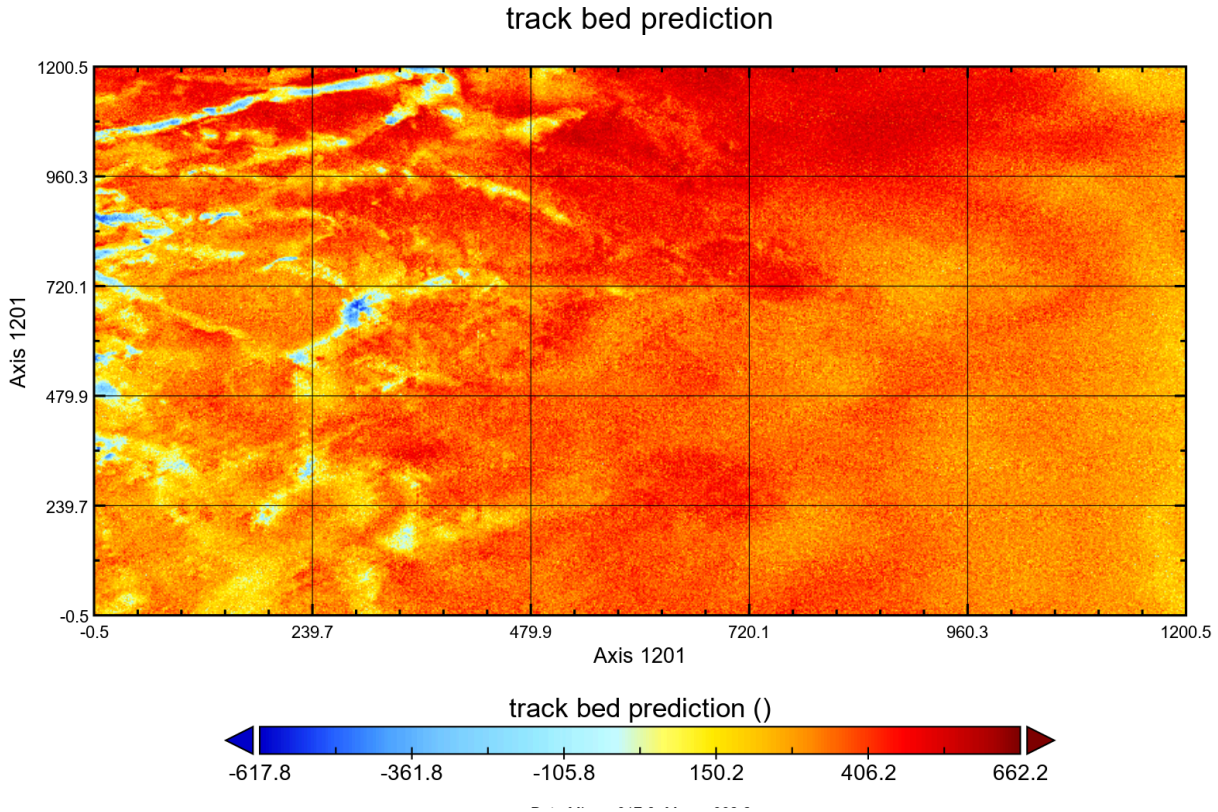
Next:

1. Combine this with known modeling in LSTM+Dense.
2. Find ELBO from KL & Error

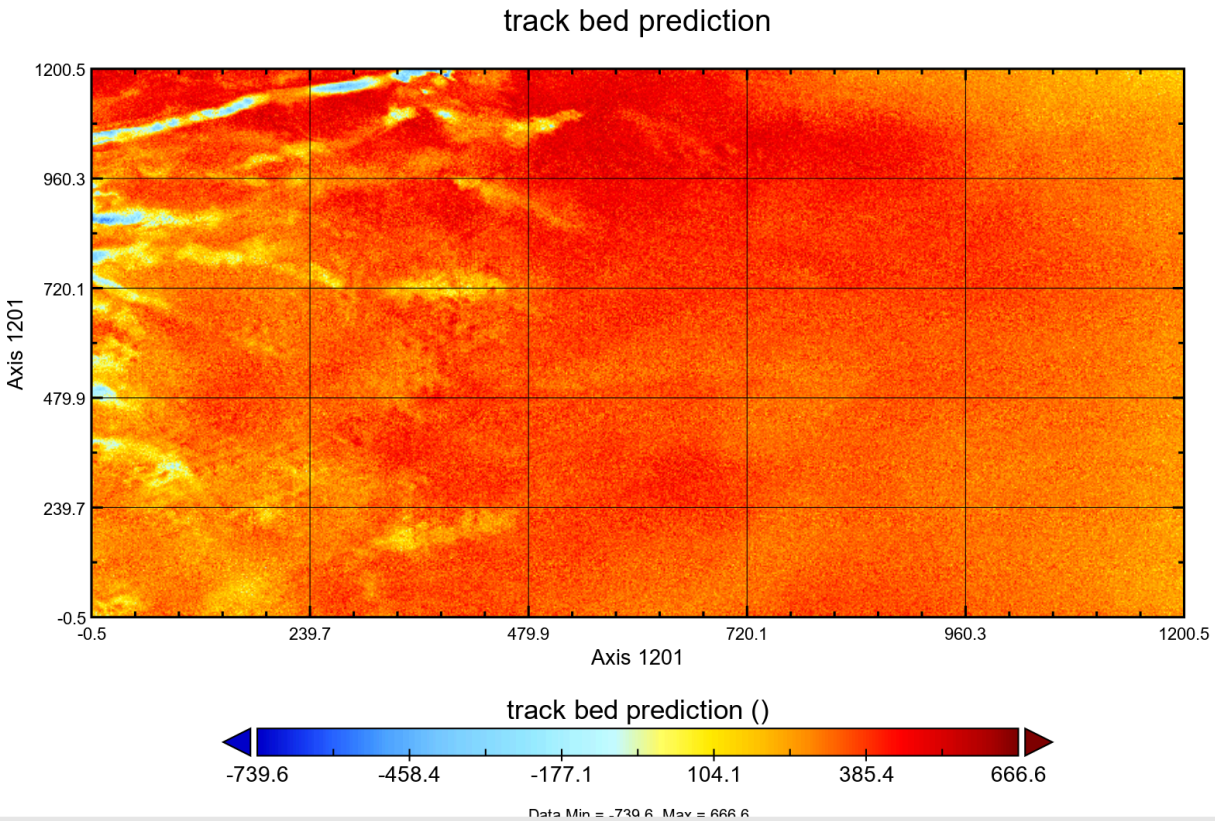
Raw VAE Prediction Run

300 epochs

```
Test stats
RMSE: 106.14681003031008
RMSE Percentage: 3638.4876861273283
Mean Absolute Error: 83.100289925444
Mean Absolute Percentage Error: 1.1337143532253442
R^2 Score: 0.6522880949191128
```



VAE with vMag




```
Validation stats
RMSE: 124.30374437064424
RMSE Percentage: 8585.135884416632
Mean Absolute Error: 96.3654566898057
Mean Absolute Percentage Error: 1.8295566326888744
R^2 Score: 0.5244135728747972
```

GAN

Background

Understanding Generative Adversarial Networks (GANs) | by Joseph Rocca | Towards Data Science

Understanding Variational Autoencoders (VAEs) | by Joseph Rocca | Towards Data Science
Connection between VAE and GAN
“At this point, a natural question that comes in mind is “what is the link between autoencoders and content generation?”. Indeed, once the autoencoder has been trained, we have both an encoder and a decoder but still no real way to produce any new content. At first sight, we could be tempted to think that, if the latent space is regular enough (well “organized” by the encoder during the training process), we could take a point randomly from that latent space and decode it to get a new content. The decoder would then act more or less like the generator of a Generative Adversarial Network”

XGBoost

Background

XGBoost (eXtreme Gradient Boosting)

- Gradient boosting framework
- Supervised learning tasks
    - regression and classification problems.
- Combines multiple weak prediction models (typically decision trees) into a strong predictive model.

Here's a high-level overview of how XGBoost works:

- Decision trees as its base learners.
    - Correct the mistakes made by the previous trees iteratively
    - Considers the reduction in loss function
- Defines an objective function that needs to be optimized during the training process
    - loss function and a regularization term

- Gradient Boosting
    - starts with an initial prediction (e.g., the mean of the target variable) and computes the gradient of the loss function with respect to this initial prediction.
        - calculates the gradients using the chain rule of derivatives.
        - update the model's predictions to minimize the loss function.
    - Trained to fit the negative gradient
        - Reduce the residuals or errors made by the previous predictions.

- Regularization
    - Applies regularization techniques - similar to ridge and lasso regression)
    - To discourage overly complex trees and prevent overfitting.

- Pruning
    - performs pruning
    - Control the model complexity and improve generalization.

- Prediction
    - Aggregates the predictions from all the trees
        - weighting each tree's contribution based on its performance.
        - prediction is obtained by summing the predictions from all the trees.

Implementation

File: Scripts/XGBoost_ky_230629.ipynb

Parameters

XGBoost Parameters — xgboost 2.0.0-dev documentation
Verbosity: printing messages
Use_rmm: true or false to use RAPIDS Memory Manager to allocate GPU memory - must be compiled with plugin enabled

General params: relates to which boosters to do boosting tree vs linear

- Booster: default to gbtree; gbtree, gblinear, dart
    - For dart:
        - `preds = bst.predict(dtest, iteration_range=(0, num_round))`
        - Sample_type: uniform or weighted
        - Normalization_type (default tree):
            - Tree: new trees have same weight of dropped trees
                - Weight new trees (1 / (k + lr))
                - Weight dropped trees (k / (k + lr))
            - Forest: new trees have same weight of sum of dropped trees
                - Weight new trees (1 / (1 + lr))
                - Weight dropped trees (1 / (1 + lr))
        - Rate_drop: dropout rate for trees [0,1]
        - One_drop: 1/0 to always drop a tree
        - Skip_drop
    - For gblinear
        - Lambda
        - Alpha
        - Updater
            - Shotgun: parallel coordinate descent alg
            - Coord_descent: deterministic descent alg
        - Feature_selector… don't use this here.
- Verbosity
- Validate parameters: true/false to perform validation of input params used or not
- Nthread: num parallel threads to run
- Disable_default_eval_metric: default False
- Num_feature: feat dims used in boosting (automatically set)

Booster params: depends on booster using
Tree

- **Eta**: aka learning rate, [0,1], shrinks feature weights to make boosting more conservative and prevent overfitting.
    - Large lr ~> overfitting
    - Small lr ~> underfitting
- Gamma: [0,:] min loss reduction required to make next leaf partition

    - Larger gamma is simpler model
- **Max_depth**: of tree
- **Min_child_weight**: [0,:], weight needed to continue partitioning
    - Larger min_child_weight is simpler model
- Max_delta_step: [0,:], to help in logreg.
    - Positive val is updates to simpler model
- **Subsample**: [0,1] Ratio of the training instances, helps with overfitting
- Sampling_method: to sample training instances
    - [default] Uniform, with subsample >= .5
    - Gradient_based: selection prob each instances proportional to regularized abs val of gradients, with subsample low as .1 without loss of accuracy
        - Only use with tree_method = gpu_hist
    - More for subsample columns… does not apply to our data since we only have 5 ind vars
- Lambda aka reg_lambda: regularizer L2
    - Increased is simpler model
- Alpha aka reg_alpha: regularizer L1
    - Increased is simpler model
- Tree_method (auto default): tree construction alg
    - … additional supporting funcs
- & more, but they don't seem to carry the same significance to tweak.

Leaning task params: based on learning scenario

Initial Results

- 60-40 Train-Test
- 20 val
- max_depth=4,
- n_estimators=64,
- min_child_weight=0.5,
- colsample_bytree=0.5,
- subsample=0.8,
- eta=0.1,
- seed=42
- eval_set=[(x_val, y_val)],
- early_stopping_rounds = 20

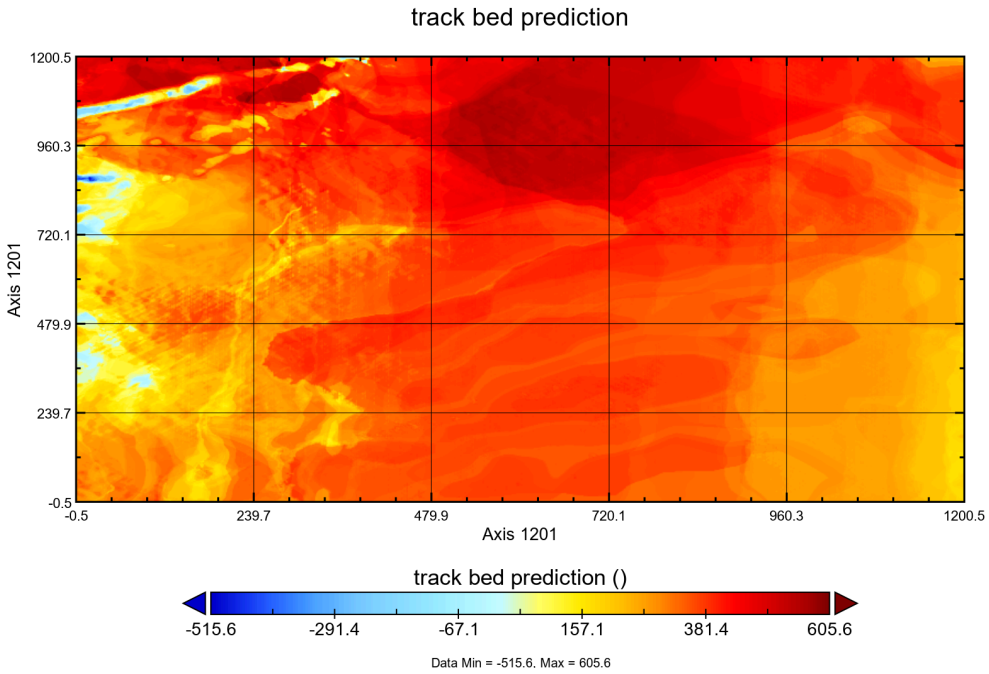
Print testing stats statements.
RMSE: **107.79185571844279**
RMSE Percentage: 16159.701260894735
RMSE Percentage-1: 717.5027663032877
Mean Absolute Error: 79.0251128522477
Mean Absolute Percentage Error: 2.0042163751365094
R^2 Score: **0.6400529428083894**

End Train-Testing Time.
Training-Testing Time: **265.875ms**

Trial 2

- 60-40 Train-Test
- 20 val
- max_depth=4,
- n_estimators=64,
- min_child_weight=0.5,
- colsample_bytree=0.5,
- subsample=0.8,
- eta=0.1,
- seed=663
- eval_set=[(x_val, y_val)],
- early_stopping_rounds = 20

Predicting 1201
Predicting 1201 Complete.
Time taken: 3.572ms



Results notes for next steps:

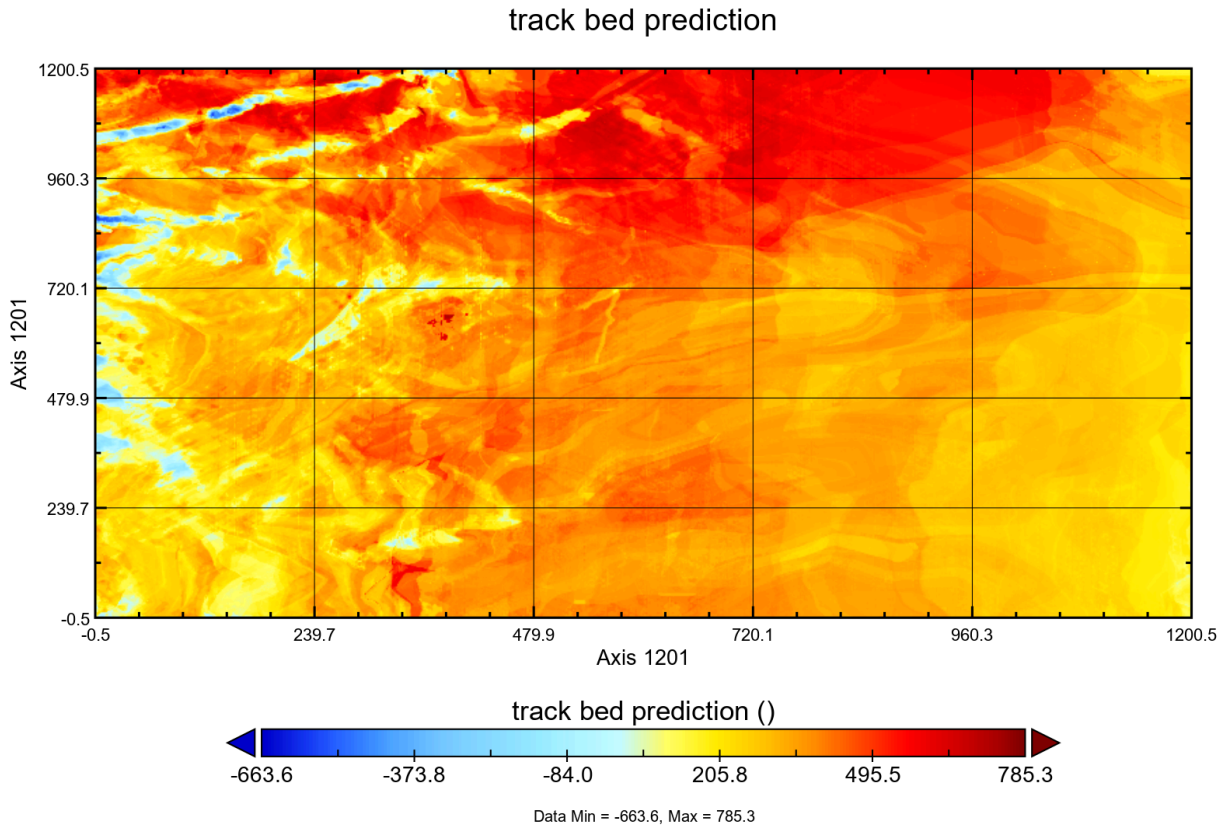
- Needs more tuning, not learning enough (reduce early stopping)
- We need a more defined track bed.

Testing Tuning Trials

Trials 1

Test 1
File: XGBoost_trial1_ky_230705.ipynb
Generated random split for train-test: 372
Take out early stopping
CPU times: user 1min, sys: 226 ms, total: 1min
Wall time: 37.6 s

```
Print testing stats statements.
RMSE: 48.137608096868824
RMSE Percentage: 2319.4466936816457
RMSE Percentage-1: 277.4653378489872
Mean Absolute Error: 33.76626173687076
Mean Absolute Percentage Error: 0.5507752294273501
R^2 Score: 0.9287581639522663
```



Test 2

```
Print testing stats statements.
RMSE: 107.67504923753457
RMSE Percentage: 14988.689709915698
RMSE Percentage-1: 973.1038333438803
Mean Absolute Error: 78.81016330840954
Mean Absolute Percentage Error: 1.8511860322087568
R^2 Score: 0.6432781196840035
```

Why is the same model getting different results? Hyperparams not being hard set.

Trials 2, train-test split

File: XGBoost_trial2_ky_230705.ipynb
Take out early stopping & train-test split

- 80-20

```
Print testing stats statements.
RMSE: 109.34076088151062
RMSE Percentage: 7450.503204349277
RMSE Percentage-1: 859.2303902319493
Mean Absolute Error: 80.073870322669
Mean Absolute Percentage Error: 1.5349940559256754
R^2 Score: 0.631600033669407
```

- 70-30

```
Print testing stats statements.
RMSE: 107.86097662823194
RMSE Percentage: 9611.388737490213
RMSE Percentage-1: nan
Mean Absolute Error: 78.74428437475697
Mean Absolute Percentage Error: 1.7415654677074202
R^2 Score: 0.6395814885166864
```

- **50-50**

```
Print testing stats statements.
RMSE: 108.57209534090231
RMSE Percentage: 13835.798930910283
RMSE Percentage-1: 888.231454827075
Mean Absolute Error: 79.38252620382842
Mean Absolute Percentage Error: 1.7727373441887622
R^2 Score: 0.6364232541625041
```

FAIL, stick with 60-40

Trials 3, Overfit the Model

File: XGBoost_trial3_ky_230705.ipynb

Test 1, remove subsample and eta

- 60-40 Train-Test
- max_depth=4,
- n_estimators=64,
- min_child_weight=0.5,
- colsample_bytree=0.5,

```
Print testing stats statements.
RMSE: 90.12126695809387
RMSE Percentage: 6444.970393635607
RMSE Percentage-1: 267.7191448698207
Mean Absolute Error: 64.45517842209895
Mean Absolute Percentage Error: 1.1968676330464527
R^2 Score: 0.7486007829612584
```

Test 2, add back

- max_depth=4,
- n_estimators=64,
- min_child_weight=0.5,
- colsample_bytree=0.5,
- subsample=0.8,
- eta=0.1

```
Print testing stats statements.
RMSE: 108.5520606820532
RMSE Percentage: 8753.06899562133
RMSE Percentage-1: 660.8399614701437
Mean Absolute Error: 79.71073341325902
Mean Absolute Percentage Error: 1.6707912021117646
R^2 Score: 0.6365910154972956
```

Test 3, alter eta to increase complexity

- max_depth=4,
- n_estimators=64,
- min_child_weight=0.25,
- colsample_bytree=0.5,
- subsample=0.8,
- eta=0.5,
- seed=42

```
Print testing stats statements.
RMSE: 81.76917015096122
RMSE Percentage: 11877.270539973197
RMSE Percentage-1: 654.9021505284147
Mean Absolute Error: 58.144194759998655
Mean Absolute Percentage Error: 1.3232592316082776
R^2 Score: 0.7929091358804393
```

Test 4, alter seed for subsampling to generated same as train-test split seed

- max_depth=4,
- n_estimators=64,
- min_child_weight=0.25,

- colsample_bytree=0.5,
- subsample=0.8,
- eta=0.5,
- seed=generated

```
Print testing stats statements.
RMSE: 82.55635681901127
RMSE Percentage: 9481.611397339344
Mean Absolute Error: 58.13191631499442
Mean Absolute Percentage Error: 1.070351357301785
R^2 Score: 0.7899951101915788
```

Test 5, increase estimators

- max_depth=4,
- n_estimators=80,
- min_child_weight=0.25,
- colsample_bytree=0.5,
- subsample=0.8,
- eta=0.5,
- seed=generated

```
Print testing stats statements.
RMSE: 79.18955105435855
RMSE Percentage: 9352.195077382039
Mean Absolute Error: 55.57193237658979
Mean Absolute Percentage Error: 1.0313257005380831
R^2 Score: 0.8067746378845333
```

Test 6, increase depth of trees

- max_depth=5,
- n_estimators=64,
- min_child_weight=0.25,
- colsample_bytree=0.5,
- subsample=0.8,
- eta=0.5,
- seed=generated

```
Print testing stats statements.
RMSE: 69.55754341116916
RMSE Percentage: 4832.469905242502
Mean Absolute Error: 48.75192487868631
Mean Absolute Percentage Error: 0.8309025705601868
R^2 Score: 0.8509208736106908
```

Test 7 , increase subsample

- max_depth=5,
- n_estimators=64,
- min_child_weight=0.25,
- colsample_bytree=0.5,
- subsample=0.9,
- eta=0.5,
- seed=generated

```
Print testing stats statements.
RMSE: 67.93429027262377
RMSE Percentage: 9000.502924359862
Mean Absolute Error: 47.89941907939945
Mean Absolute Percentage Error: 0.8949060427451626
R^2 Score: 0.857797754836275
```

Test 8, increase depth

- max_depth=6,
- n_estimators=64,
- min_child_weight=0.25,
- colsample_bytree=0.5,
- subsample=0.8,
- eta=0.5,
- seed=generated

```
Print testing stats statements.
RMSE: 57.43279670211823
RMSE Percentage: 2646.4818671364087
Mean Absolute Error: 40.109954748973294
Mean Absolute Percentage Error: 0.6346382173281788
R^2 Score: 0.8983638283839795
```

Test 9, increase number of estimators

- max_depth=6,
- n_estimators=80,
- min_child_weight=0.25,
- colsample_bytree=0.5,
- subsample=0.8,
- eta=0.5,
- seed=generated

```
Print testing stats statements.
RMSE: 53.612796594493
RMSE Percentage: 2325.662515616943
Mean Absolute Error: 37.22337238066469
Mean Absolute Percentage Error: 0.5631668713184012
R^2 Score: 0.9114343545880209
```

Test 10, increase num estimators

- max_depth=6,
- n_estimators=100,
- min_child_weight=0.25,
- colsample_bytree=0.5,
- subsample=0.8,
- eta=0.5,
- seed=generated

```
Print testing stats statements.
RMSE: 50.06622424962592
RMSE Percentage: 2274.8693378077132
Mean Absolute Error: 34.7666218754609
Mean Absolute Percentage Error: 0.522117602505479
R^2 Score: 0.9227643068093137
```

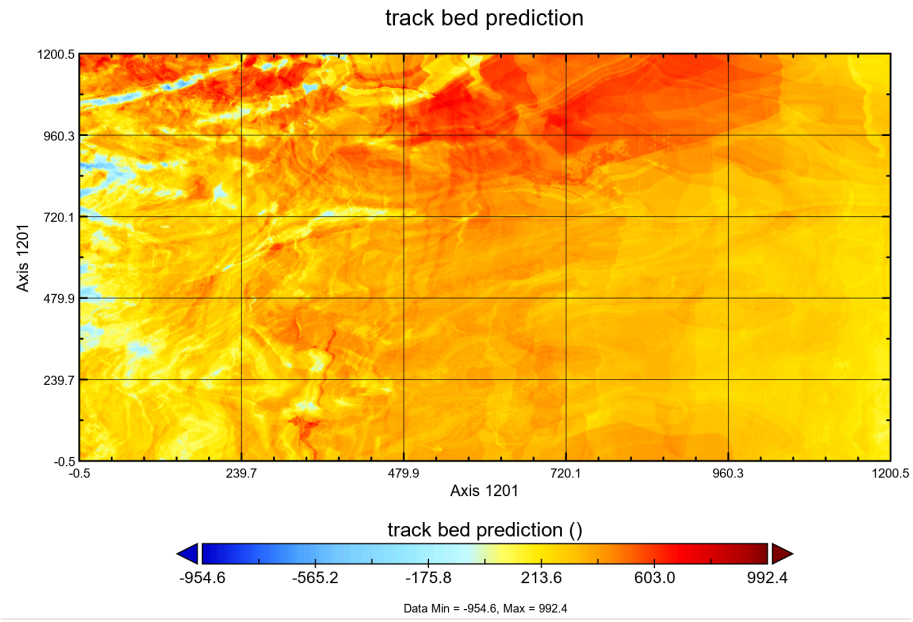
Test 11, increase number estimators

- `Seed 323`
- max_depth=6,
- n_estimators=150,
- min_child_weight=0.25,
- colsample_bytree=0.5,
- subsample=0.8,
- eta=0.5,
- seed=generated

```
Print testing stats statements.
RMSE: 44.661359124415085
RMSE Percentage: 1893.497750879121
Mean Absolute Error: 30.565779518582204
Mean Absolute Percentage Error: 0.450097642585426
R^2 Score: 0.938540044885473
```

All run again test 11

```
Print testing stats statements.
RMSE: 42.77482930872207
RMSE Percentage: 2974.941172344448
Mean Absolute Error: 29.272512695828894
Mean Absolute Percentage Error: 0.4907277541818431
R^2 Score: 0.9433040604176197
```



Trials 4, Reduce Overfitting for better h5

Test 1, reduce max depth

- max_depth=4,
- n_estimators=150,
- min_child_weight=0.25,
- colsample_bytree=0.5,
- subsample=0.8,
- eta=0.5,
- seed=generated

```
Print testing stats statements.
RMSE: 65.97007713090495
RMSE Percentage: 2477.155691919344
Mean Absolute Error: 45.81803874575685
Mean Absolute Percentage Error: 0.731728198202718
R^2 Score: 0.8651833131457364
```

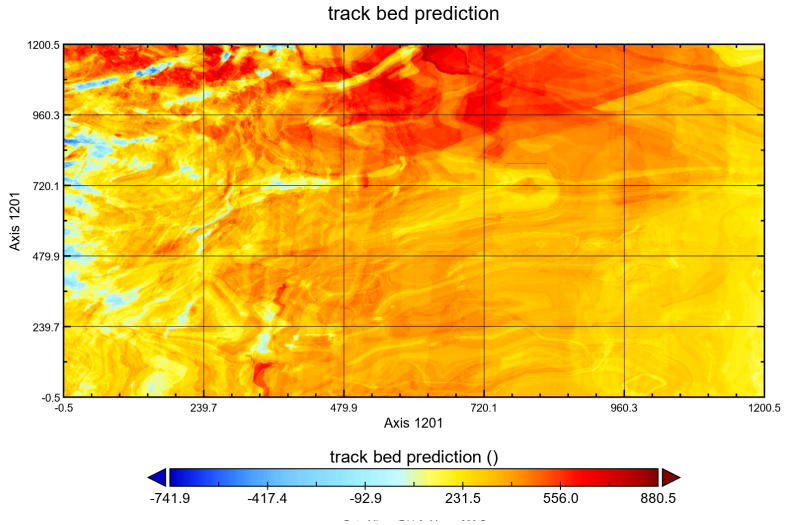
Lost 8% for a much more simplified model

Test 2, implement early stopping

- max_depth=6,
- n_estimators=150,
- min_child_weight=0.25,
- colsample_bytree=0.5,
- subsample=0.8,

- eta=0.5,
- seed=generated)
- eval_metric="rmse",
- eval_set=[(x_val, y_val)],
- verbose=True,
- early_stopping_rounds = 20

```
Print testing stats statements.
RMSE: 43.74988849129311
RMSE Percentage: 1451.9893010970964
Mean Absolute Error: 29.610765532243946
Mean Absolute Percentage Error: 0.4504389667818035
R^2 Score: 0.9407069805178303
```



Test 2, max_depth compromise

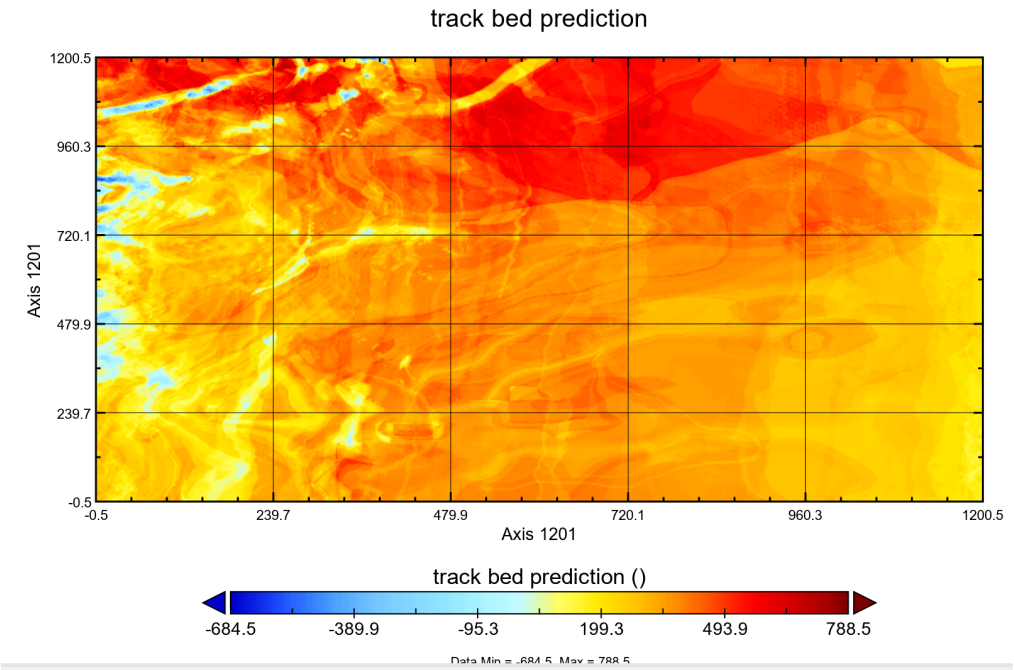
- max_depth=5,
- n_estimators=150,
- min_child_weight=0.25,
- colsample_bytree=0.5,
- subsample=0.8,
- eta=0.5
- seed=generated)
- eval_metric="rmse",
- eval_set=[(x_val, y_val)],
- verbose=True,
- early_stopping_rounds = 20

```
Print testing stats statements.
RMSE: 53.206368579916294
RMSE Percentage: 2449.6709844900683
Mean Absolute Error: 36.60438667999406
Mean Absolute Percentage Error: 0.5514097415594482
R^2 Score: 0.9123045817676008
```

Test 3, reduce depth, eta, and early stop

- max_depth=4,
- n_estimators=150,
- min_child_weight=0.25,
- colsample_bytree=0.5,
- subsample=0.8,
- eta=0.3,
- seed=generated
- eval_metric="rmse",
- eval_set=[(x_val, y_val)],
- verbose=True,
- early_stopping_rounds = 20

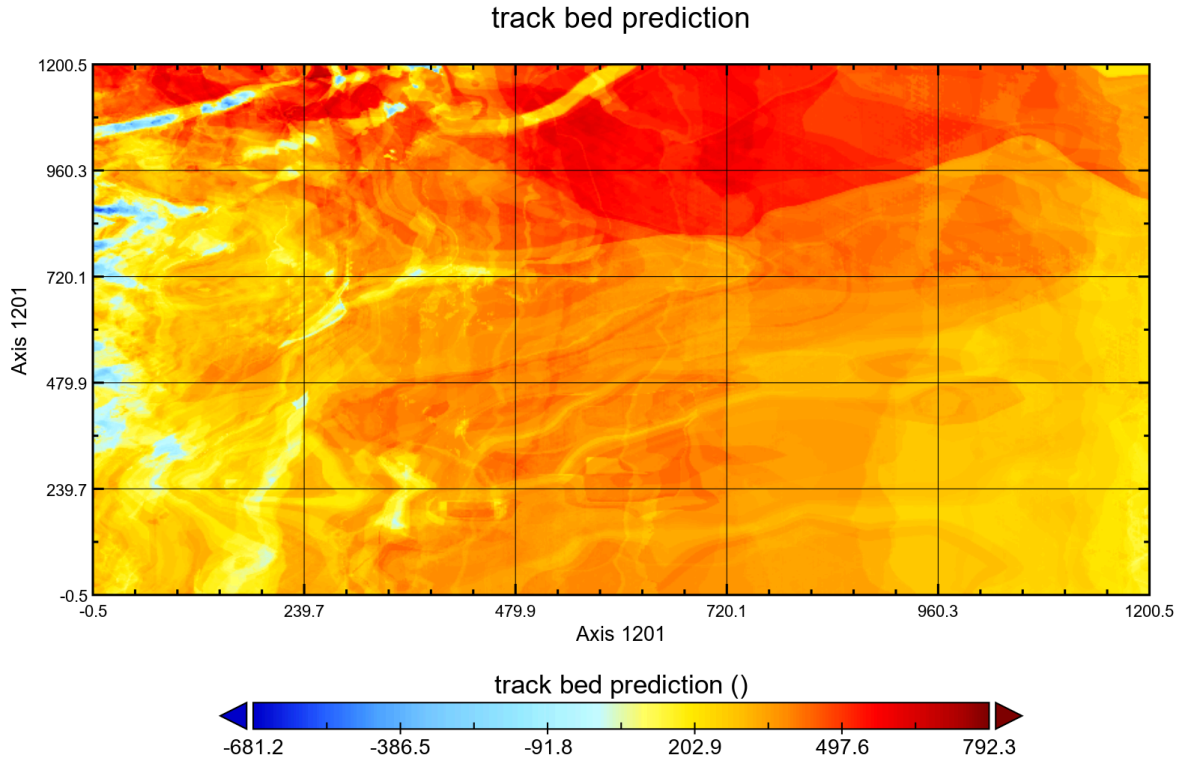
```
Print testing stats statements.
RMSE: 74.59615939556805
RMSE Percentage: 2659.2834651384323
Mean Absolute Error: 51.90482882590285
Mean Absolute Percentage Error: 0.8292780679563827
R^2 Score: 0.8276216962127394
```



Test 4,

- max_depth=4,
- n_estimators=100,
- min_child_weight=0.25,
- colsample_bytree=0.5,
- subsample=0.8,
- eta=0.3,
- seed=generated
- eval_metric="rmse",
- eval_set=[(x_val, y_val)],
- verbose=True,
- early_stopping_rounds = 20

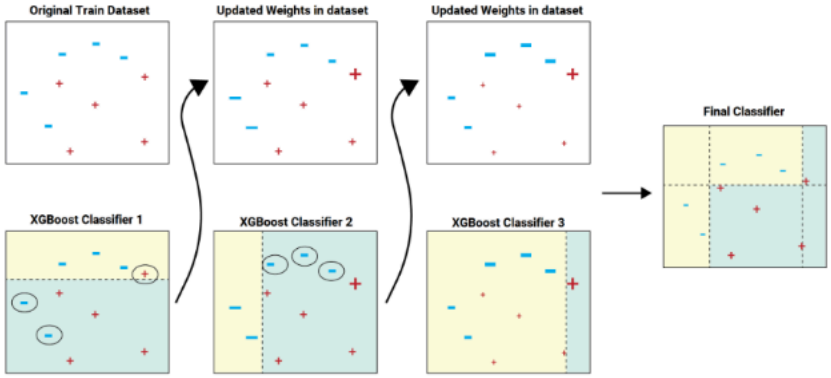
```
Print testing stats statements.
RMSE: 82.35751040596074
RMSE Percentage: 3000.0883770972428
Mean Absolute Error: 57.96975966984643
Mean Absolute Percentage Error: 0.9410723651308998
R^2 Score: 0.789885470058859
```



Slides

Issues with GPR with entire dataset →
Idea: make many gprs to run on small subsets and then aggregate results…. = core idea of…

XGBoosting



How does it work?

Combines multiple weak prediction models (DT) into model by correcting previous mistakes

&

Applies regularization techniques

To discourage overly complex trees

and prevent overfitting

… aggregates tree results

Initial Results (pre tuning)

Results from last year vs. Results running it yesterday

Table 2: Performance Comparison

Model	RMSE	MAE	R^2
Gradient Boosting	136.93	100.63	0.113
XGBoost	135.18	97.06	0.135
Lasso Regressor	147.30	106.79	-0.026
Dense	387.98	364.02	-6.110
LSTM	556.09	493.71	-13.620
Dense+LSTM	**57.30**	**55.04**	**0.840**

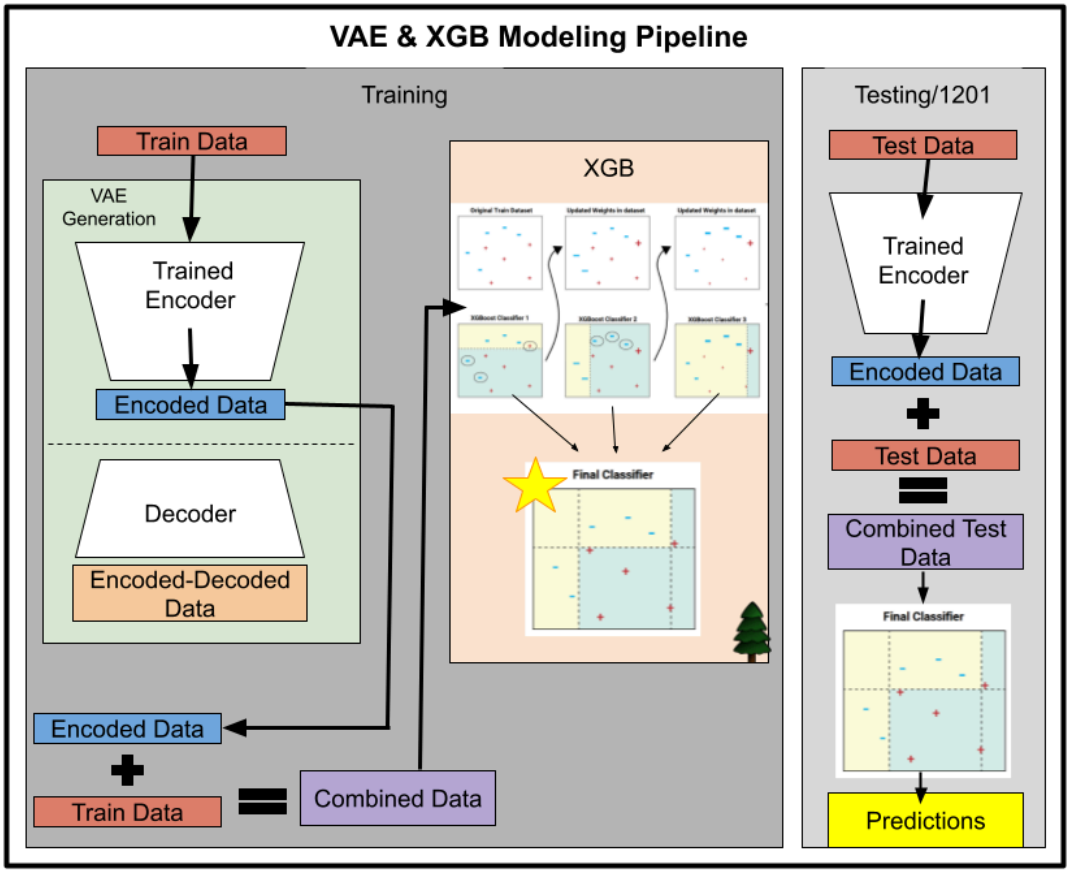
RMSE: **107.79185571844279**
RMSE Percentage: 16159.701260894735
RMSE Percentage-1: 717.5027663032877
Mean Absolute Error: 79.0251128522477
Mean Absolute Percentage Error: 2.0042163751365094
R^2 Score: **0.6400529428083894**
End Train-Testing Time.
Training-Testing Time: **265.875ms**

Next Steps

Tune XGBoost

Combined VAE & XGBoost

Pipeline



Base

File: VAE-XGB_ky_070523.ipynb

Parameters:
60-40 TT split rand generated
XGB

- max_depth=4,
- n_estimators=100,
- min_child_weight=0.25,
- colsample_bytree=0.5,
- subsample=0.8,
- eta=0.3,
- seed=generated

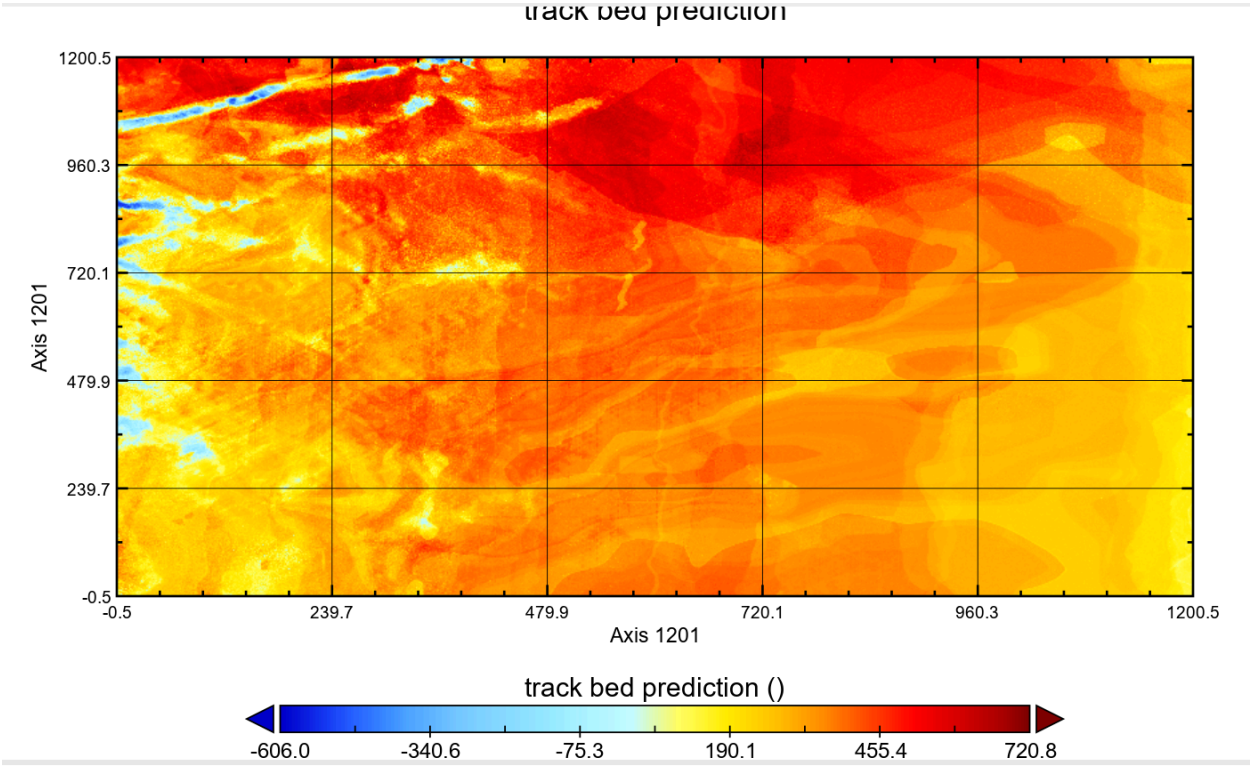
VAE

- num_columns = 5
- vae.compile(optimizer=keras.optimizers.Adam())
- vae.fit(x_train, y_train,
- validation_data=(x_val, y_val),
- epochs=10,
- batch_size=712)

XGB

- xgb_model.fit(combined_features, y_train,
- eval_metric="rmse",
- eval_set=[(x_val, y_val)],
- verbose=True

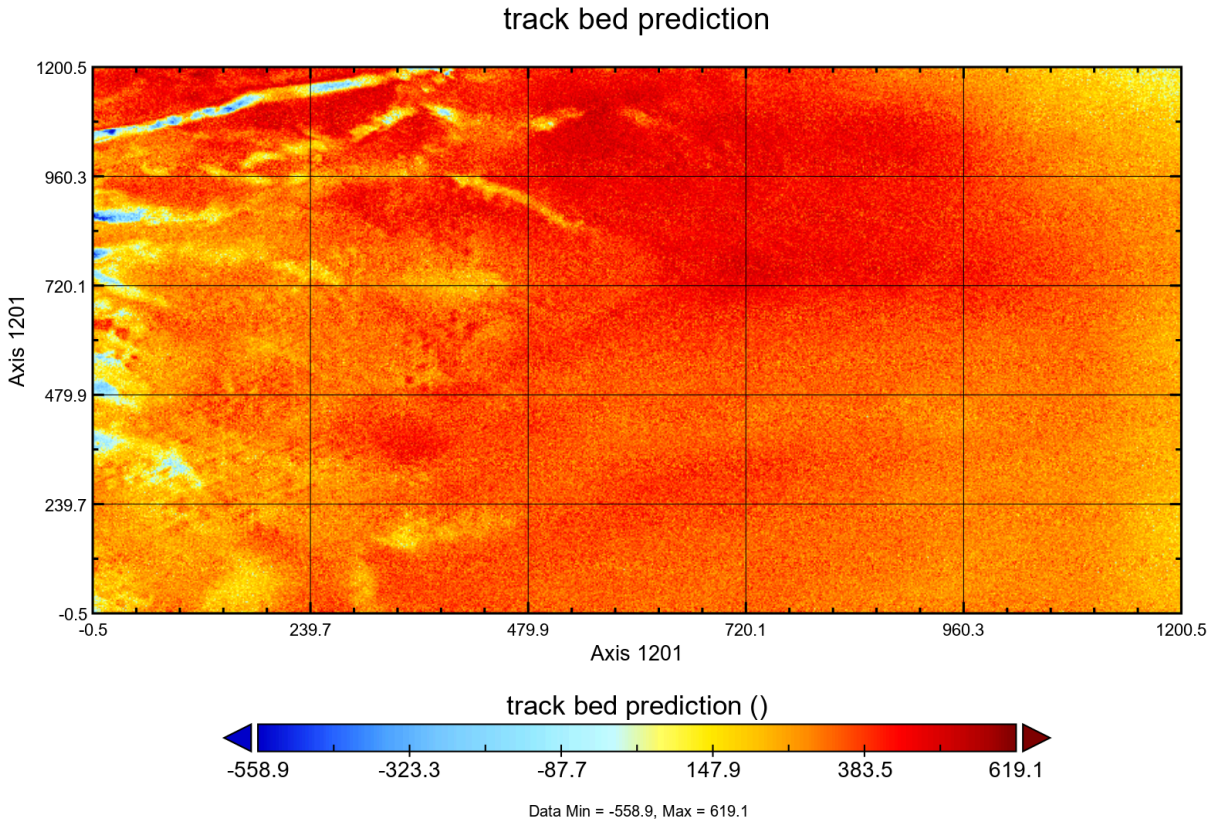
```
Print testing stats statements.
RMSE: 82.1822374628812
RMSE Percentage: 11925.942671398769
Mean Absolute Error: 59.20048193873395
Mean Absolute Percentage Error: 1.4597567365332504
R^2 Score: 0.7920981701323279
CPU times: user 22.1 s, sys: 565 ms, total: 22.7 s
Wall time: 22 s
```



VAE direct XGB

Putting VAE predictions right into the XGB to train the model
File: VAE-XGB_direct_ky_230706.ipynb

```
Print testing stats statements.
RMSE: 126.32427890573162
RMSE Percentage: 6854.068011505286
Mean Absolute Error: 97.19687454009363
Mean Absolute Percentage Error: 1.700128758980 22
R^2 Score: 0.5069772564869249
CPU times: user 15.3 s, sys: 557 ms, total: 15.9 s
```



LSTM+Dense Model

Results

Trial 1

File: LSTM+Dense_trial1_ky_230705.ipynb

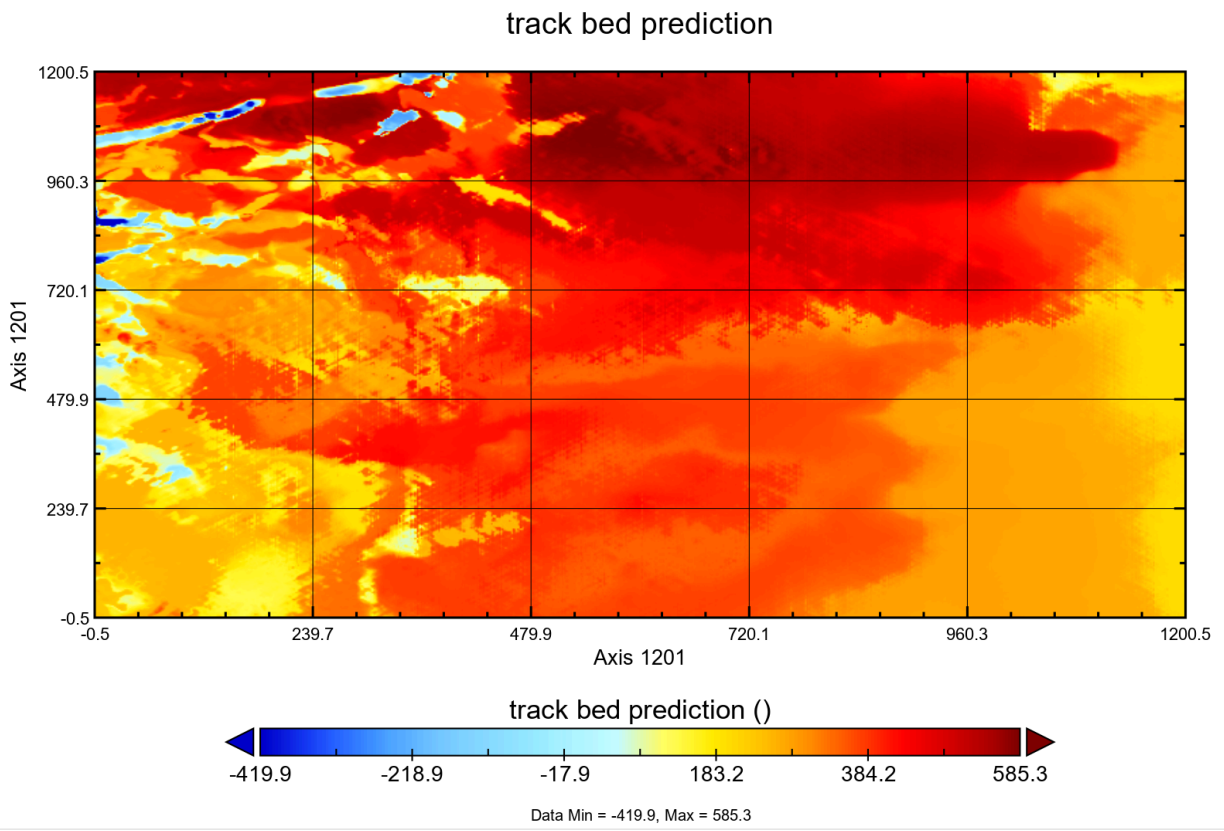
- 60-40 split
- 209 generated random split
- StandardScaling
- 200 epochs
- 5000 batch
- 30% validation on mse loss
- Shuffled
- Callback 100 patience with val_loss monitoring

Results:
Still not performing as well as reported.
35 minutes to train
1.5 minutes to predict train and test
21 seconds to predict validation

```
Print train stats statements.
RMSE: 86.6668684661375
RMSE Percentage: 7779.690052151989
RMSE Percentage-1: 449.1202267531169
Mean Absolute Error: 62.20041905380454
Mean Absolute Percentage Error: 1.3114575011385199
R^2 Score: 0.7676912371646578
```

```
Print validation stats statements.
RMSE: 87.01648392407351
RMSE Percentage: 8208.596579710864
RMSE Percentage-1: 491.96300156519845
Mean Absolute Error: 62.51190295982683
Mean Absolute Percentage Error: 1.4475801222597322
R^2 Score: 0.768103346621412
```

```
Print testing stats statements.
RMSE: 87.07688416960245
RMSE Percentage: 11859.440831198983
RMSE Percentage-1: 289.40036706596175
Mean Absolute Error: 62.38122365981117
Mean Absolute Percentage Error: 1.5184609657812962
R^2 Score: 0.7658959943069319
```

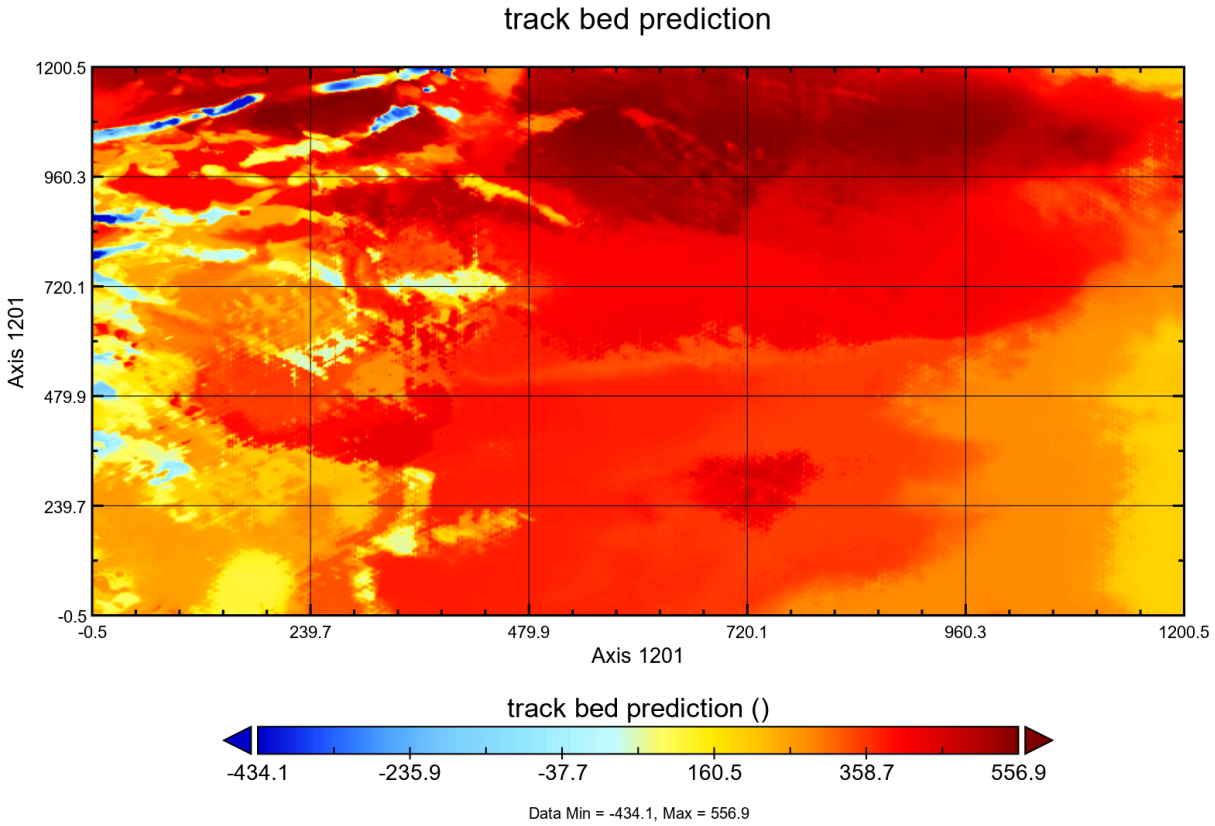

track bed prediction
1200.5
960.3
720.1
479.9
239.7
-0.5
Axis 1201
-0.5
239.7
479.9
720.1
960.3
1200.5
Axis 1201
track bed prediction ()
-419.9
-218.9
-17.9
183.2
384.2
585.3
Data Min = -419.9, Max = 585.3

Trial 2

File: LSTM+Dense_trial2_ky_230630.ipynb

1. Different generated seed

1201 data predicted and visualized



82.549	0.789	LSTM+Dense

Predicting 1201 Complete.

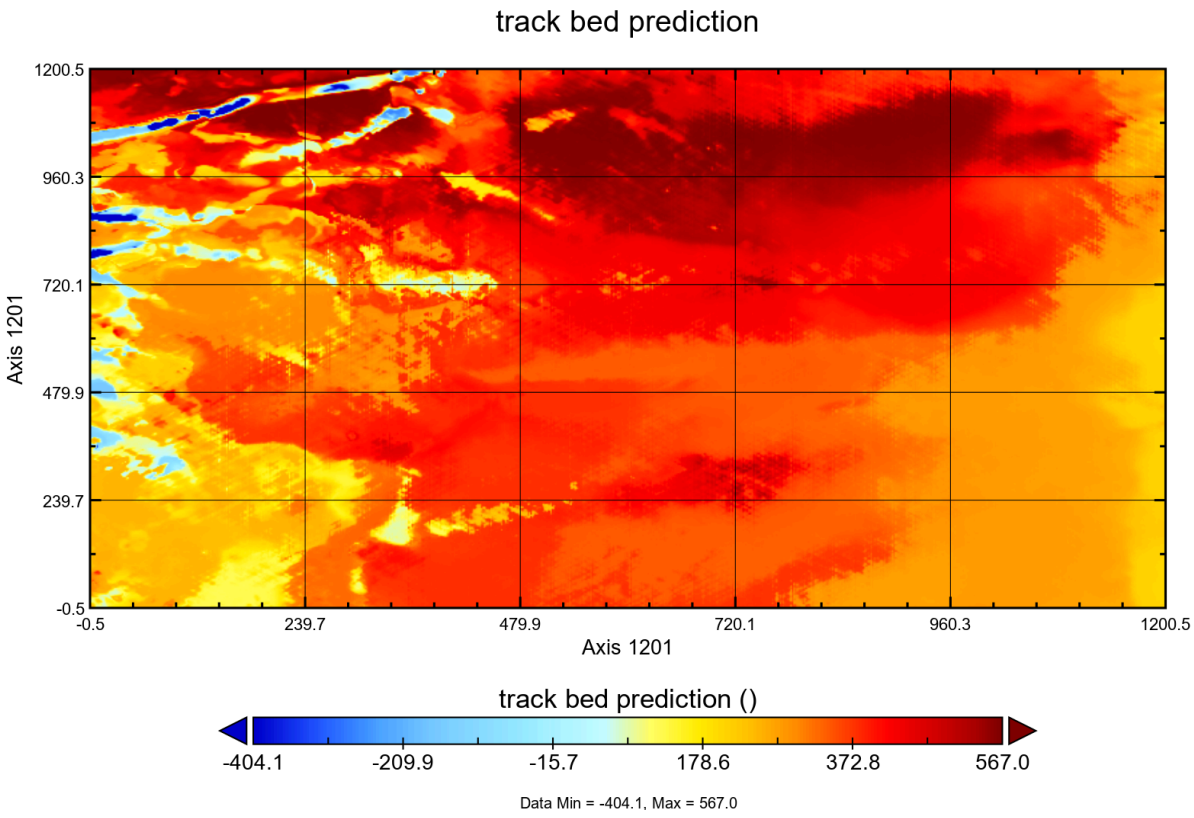
Time taken: 161.602ms

Trial 3

File: LSTM+Dense_trial3_ky_230705.ipynb

2. Try with assigned split 42

1201 data predicted and visualized
81.96 0.794 LSTM+Dense
Predicting 1201 Complete.
Time taken: 202.049ms



Trial 4 (Canceled)

3. Try with chunked data

Trial 5 (Canceled)

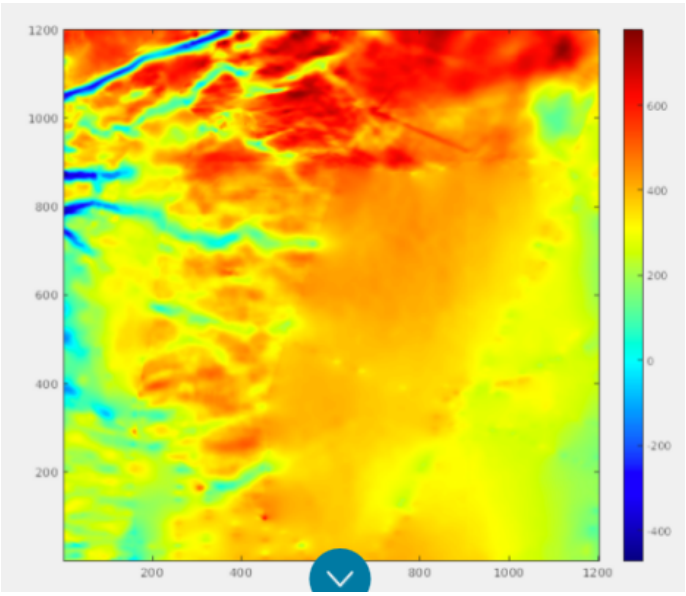
4. Report loss and R^2 each epoch to see when they are min & maximized during training/prediction

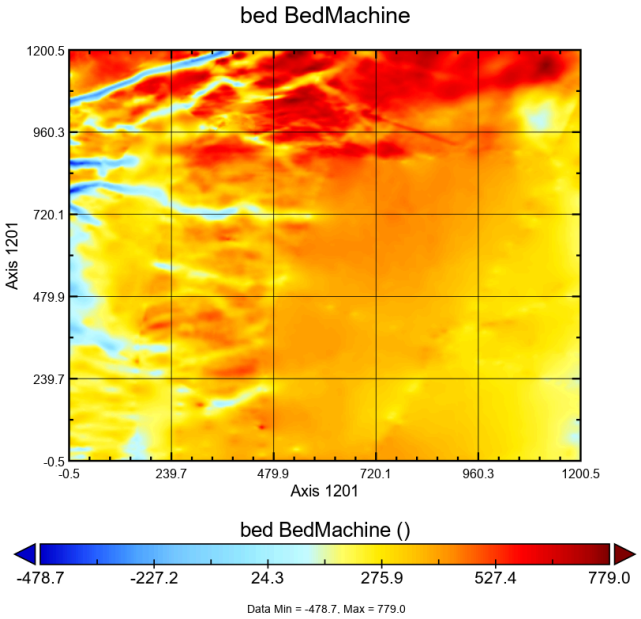
Trial 6 (Canceled)

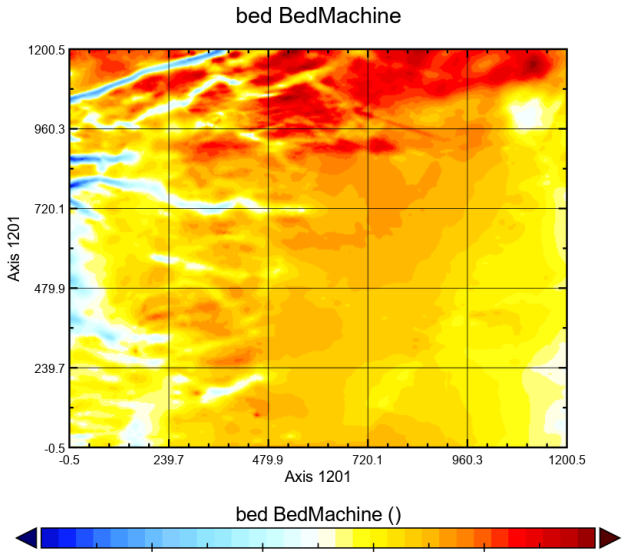
5. Run optimization on the model

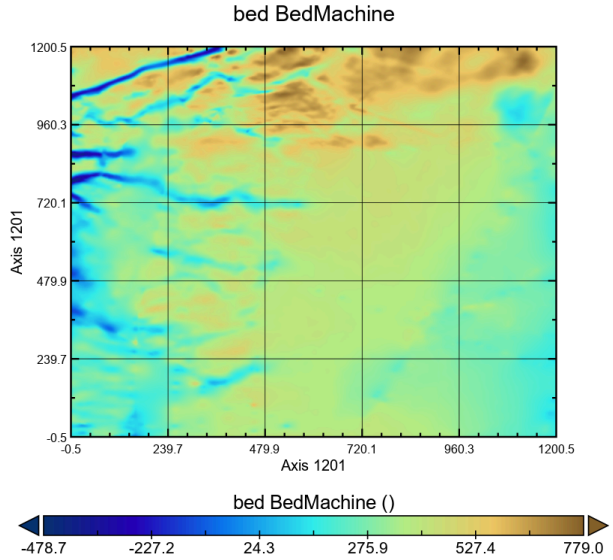
a. Adjust epochs
b. Batch_size
c. Validation_split
d. Callback early stopping patience

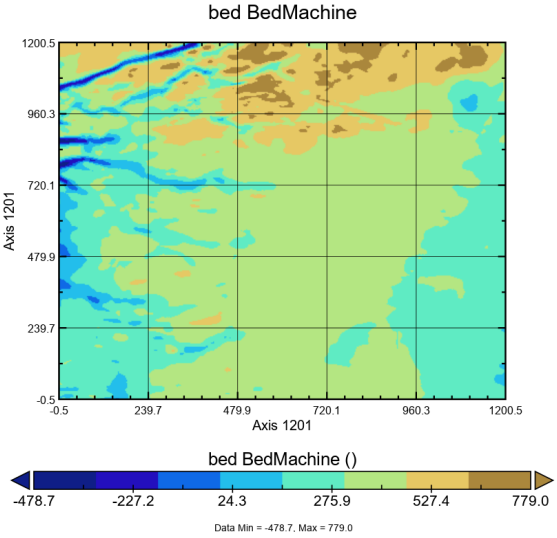
To Beat Rereviewed











Paths with Derived Velocity Work

Working Directory:
MyDrive/REU 2023 Team 1: Ice Bed Topography Prediction/Research/Yi_Work

Documentation (current):
MyDrive/REU 2023 Team 1: Ice Bed Topography Prediction/Research/Yi_Work/Documentation Velocity Magnitude

Background Research:
MyDrive/REU 2023 Team 1: Ice Bed Topography Prediction/Research/Yi_Work/Background/*

Data:
MyDrive/REU 2023 Team 1: Ice Bed Topography Prediction/Research/Yi_Work/Data_derivedVelocity/*

Scripts:
MyDrive/REU 2023 Team 1: Ice Bed Topography Prediction/Research/Yi_Work/Scripts_derivedVelocity/*

Documentation 1:
MyDrive/REU 2023 Team 1: Ice Bed Topography Prediction/Research/Yi_Work/Documentation

Derive Magnitude Velocity

File: derive_velocity.ipynb
Input:
mainPath = '/content/gdrive/MyDrive/REU 2023 Team 1: Ice Bed Topography Prediction/Research/Yi_Work/Data_derivedVelocity/'

data_full_ = mainPath + 'data_full.csv' #training and test combined
data_1201_ = mainPath + 'df_1201_validation_data.csv'

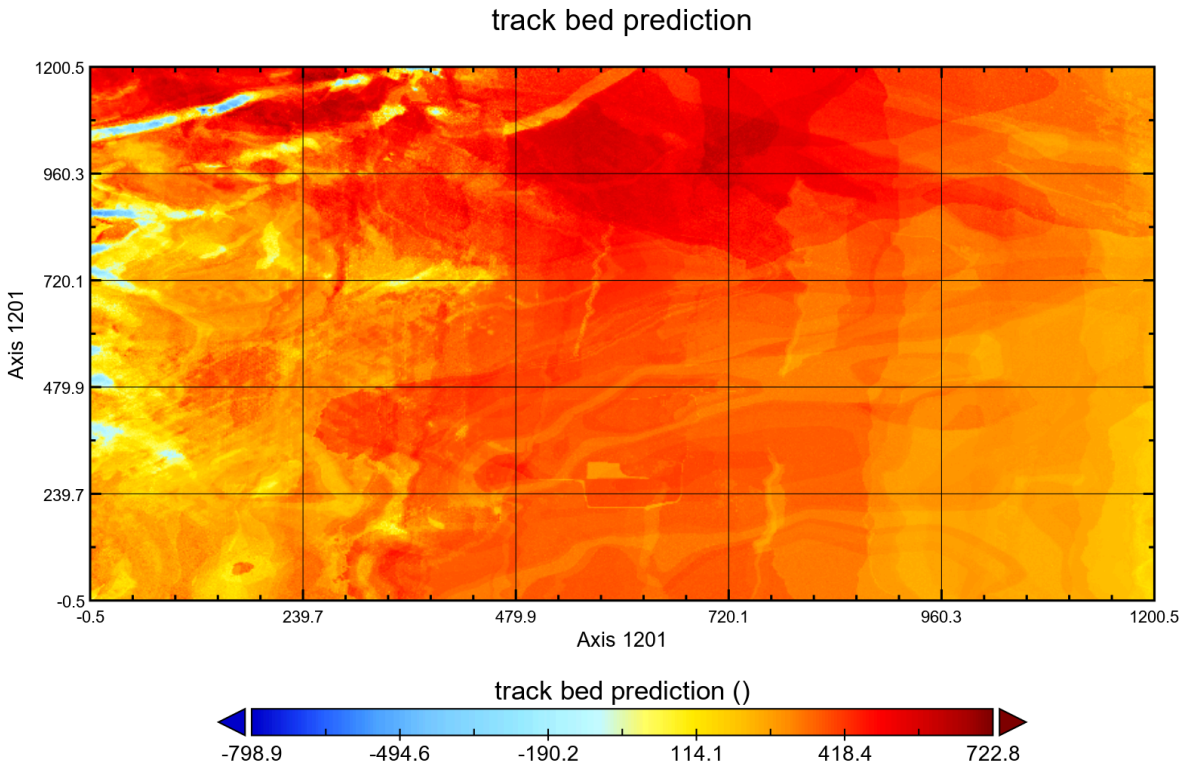
Output:
data_full_vMag.csv
d1201_vMag.csv

Equation:
Magnitude at x,y = sqrt(x^2 + y^2)

VAE-XGB with vMag

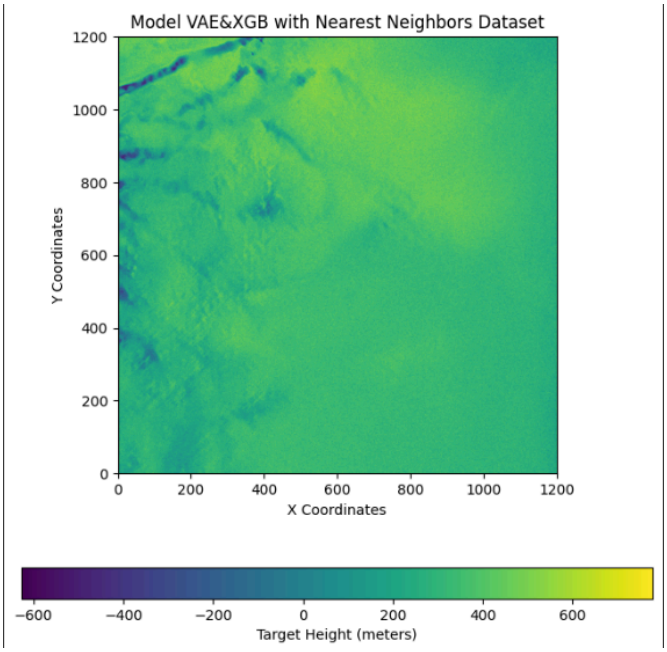
File: VAE-XGB_mag_ky_230706.ipynb

Test 1
With vMag (dropping surf centers)
Pull out true validation data



Thoughts… XGB performs better alone at the moment. XGB stand alone will be pursued at this time.

Test 2 w/ v_mag
Print testing stats statements.
RMSE: 129.75976429980784
RMSE Percentage: 3888.079988801366
Mean Absolute Error: 100.03457276030343
Mean Absolute Percentage Error: 1.5122093676304211
R^2 Score: 0.4807987751170596
CPU times: user 17.1 s, sys: 613 ms, total: 17.7 s
Wall time: 15.4 s

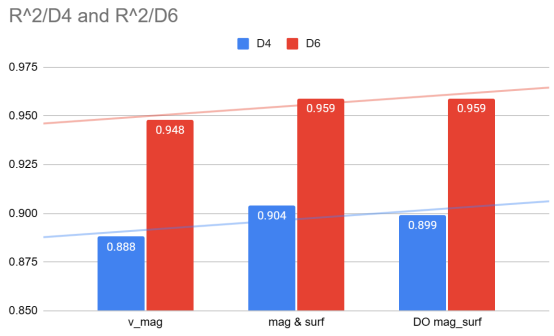

Model VAE&XGB with Nearest Neighbors Dataset
Y Coordinates
X Coordinates
0
200
400
600
800
1000
1200
−600
−400
−200
0
200
400
600
Target Height (meters)

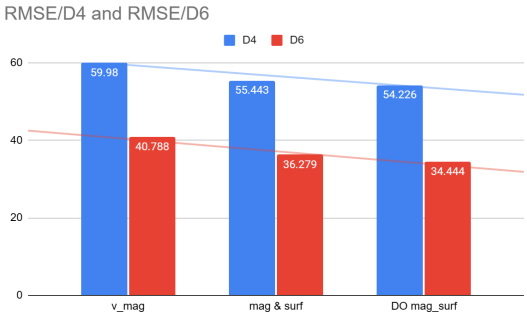
Determining Methodology for XGBT4 (Irrelevant due to Poor Map Quality Produced)

Trial4

powerful enough to deal with all sorts of irregularities of data

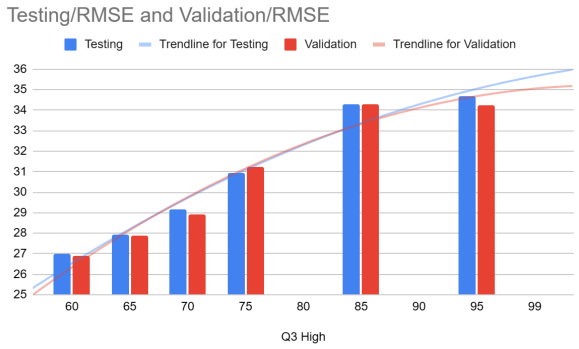
Determining the best features and drop outliers methodology for the XGBT4 Model						
Test and Params	v_mag		mag & surf		Drop outliers over .88%	
R^2	0.888	0.948	0.904	0.959	0.899	0.959
RMSE	59.98	40.788	55.443	36.279	54.226	34.444
Seed	168	168	168	168	168	168
Depth	4	6	4	6	4	6
	Conclusion: (1) dropping outliers reduces error (RMSE) (2) adding features (surf) improves the R^2 and reduces error					
	R^2		RMSE			
	D4	D6	D4	D6		
v_mag	0.888	0.948	59.98	40.788		
mag & surf	0.904	0.959	55.443	36.279		
DO mag_surf	0.899	0.959	54.226	34.444		
Range	0.016	0.011	5.754	6.344		

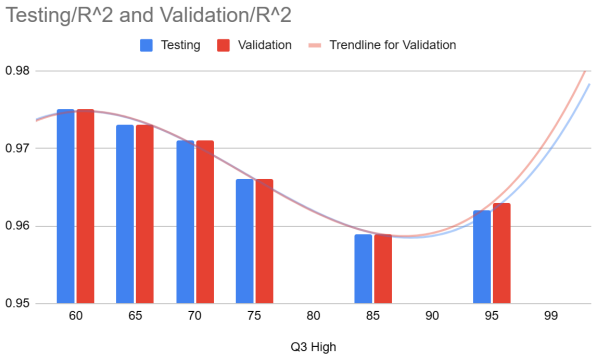




Next:
Determine the best outlier dropout

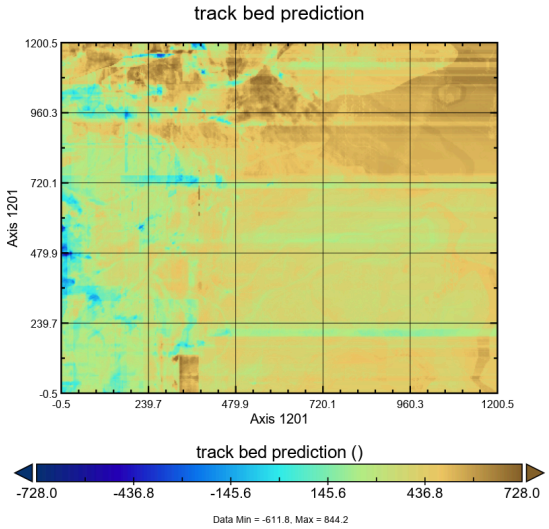
XGBoost_t4_dropoutliers_vMag_surf_ky_230707.ipynb, 168 seed. Q1 @.25, Q3 adjusted.					
		Testing		Validation	
Q3 High	**Num Dropped**	**RMSE**	**R^2**	**RMSE**	**R^2**
60	296297	26.995	0.975	26.892	0.975
65	239518	27.935	0.973	27.864	0.973
70	182007	29.170	0.971	28.898	0.971
75	103563	30.927	0.966	31.213	0.966
80					
85	20271	34.269	0.959	34.297	0.959
90					
95	2791	34.662	0.962	34.259	0.963
99					



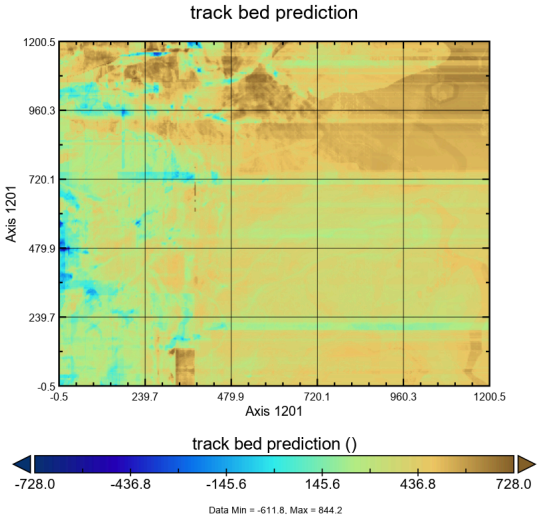


Graphing the results leaves “streaks” on the images. Revert to keeping all data (not removing outliers) to maintain a better map.

60% Q3 drop



90% Q3 Drop



XGB-T4 Mag & Surf

Already established that more features results in a lower RMSE score.
File: XGBoost_trial4_vMag_surf_ky_230706.ipynb

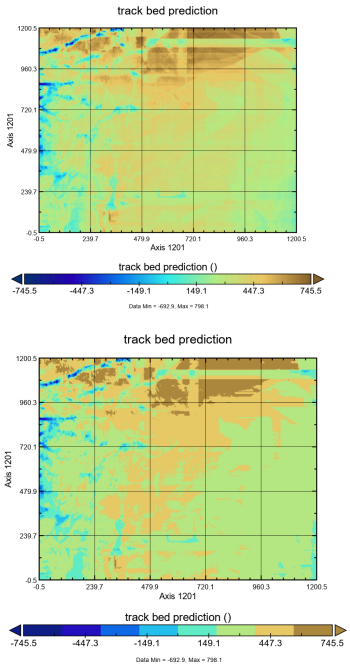
Identify the best #iterations & depth for trees

XGBoost_trial4_vMag_surf_ky_230706.ipynb				
50 Iterations				
	Testing		Validation	
Depth	RMSE	R^2	RMSE	R^2
4	76.946	0.817	77.047	0.815
6	50.306	0.922	50.064	0.922
7	42.222	0.945	42.161	0.944
8	35.941	0.960	35.932	0.960
10	27.901	0.976	27.848	0.976
100 Iterations				
	Testing		Validation	
Depth	RMSE	R^2	RMSE	R^2
6	40.78	0.949	40.62	0.949
150 Iterations				
	Testing		Validation	
Depth	RMSE	R^2	RMSE	R^2
6	35.9	0.96	35.621	0.961
8	28.169	0.976	28.224	0.975
10	24.349	0.982	24.425	0.981

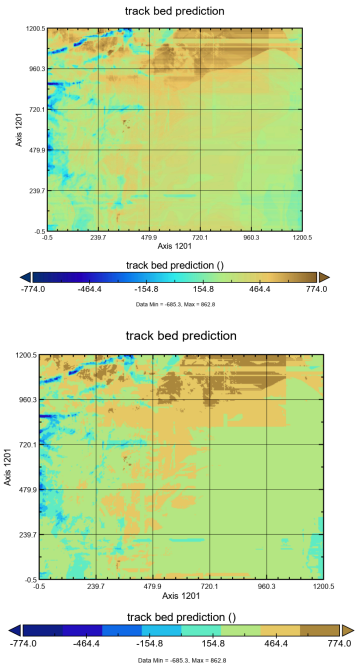
Fuzzy image - great mt gradients and predictions at the top. Colors are difficult to isolate. Move more towards a simpler model that over predicts the highs to get a better contrast in the imaging

Next: move towards a simpler model that highlights the contrast of the mountains and valleys.

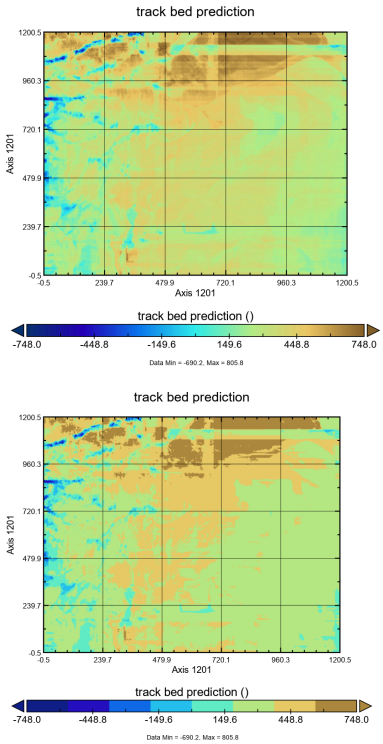
Depth 6, 100 iters; scale centered on 0



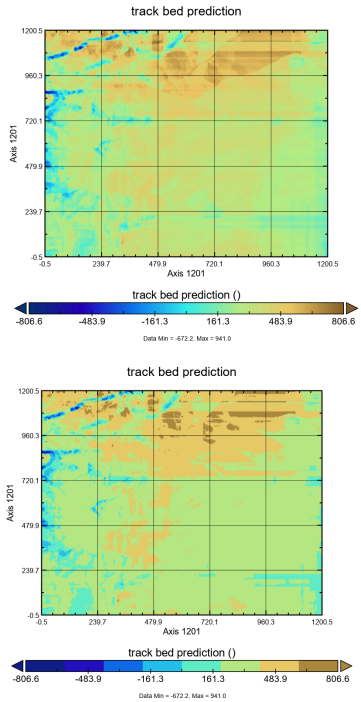
Depth 5, 100 iters; scale centered on 0



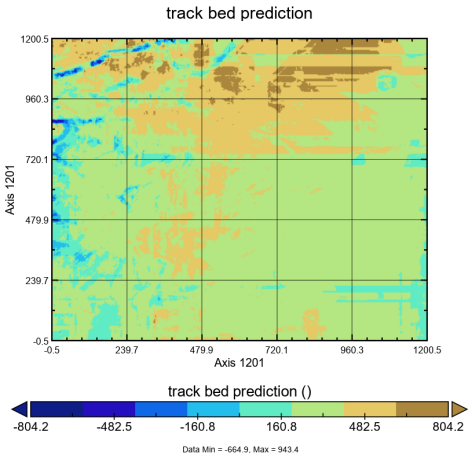
Depth 6, 150 iters; scale centered on 0



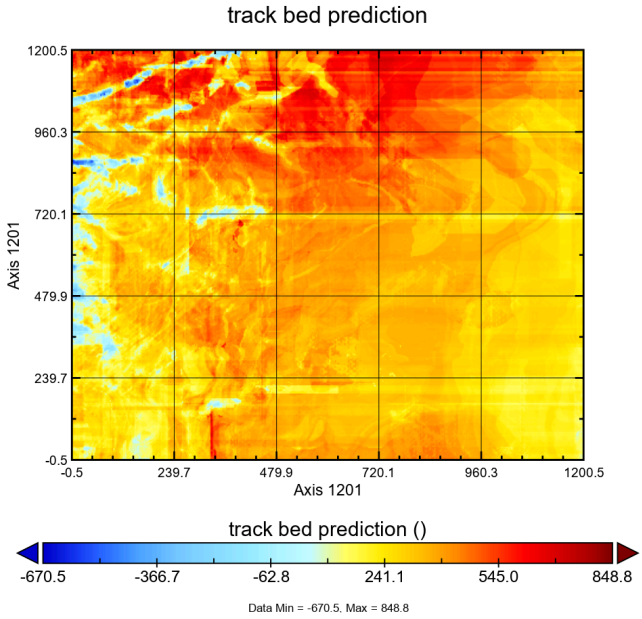
Depth 4, 150 iters; scale centered at 0



Depth 4, 200 iters; scaled centered on 0

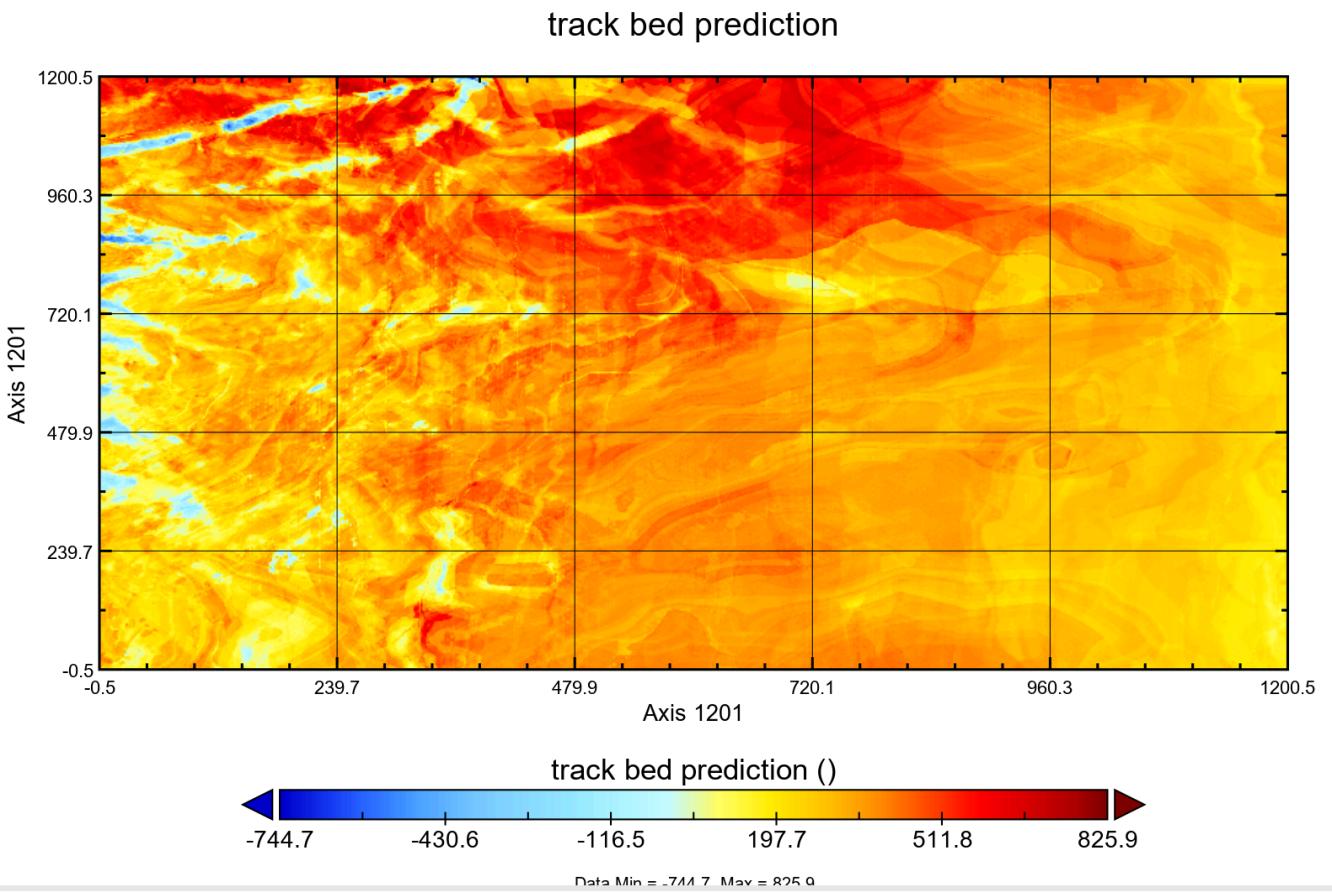


XBG-T4_Mag and surf causes streakiness in the map., mag and surf



XGB Results to match… does this include surf?!

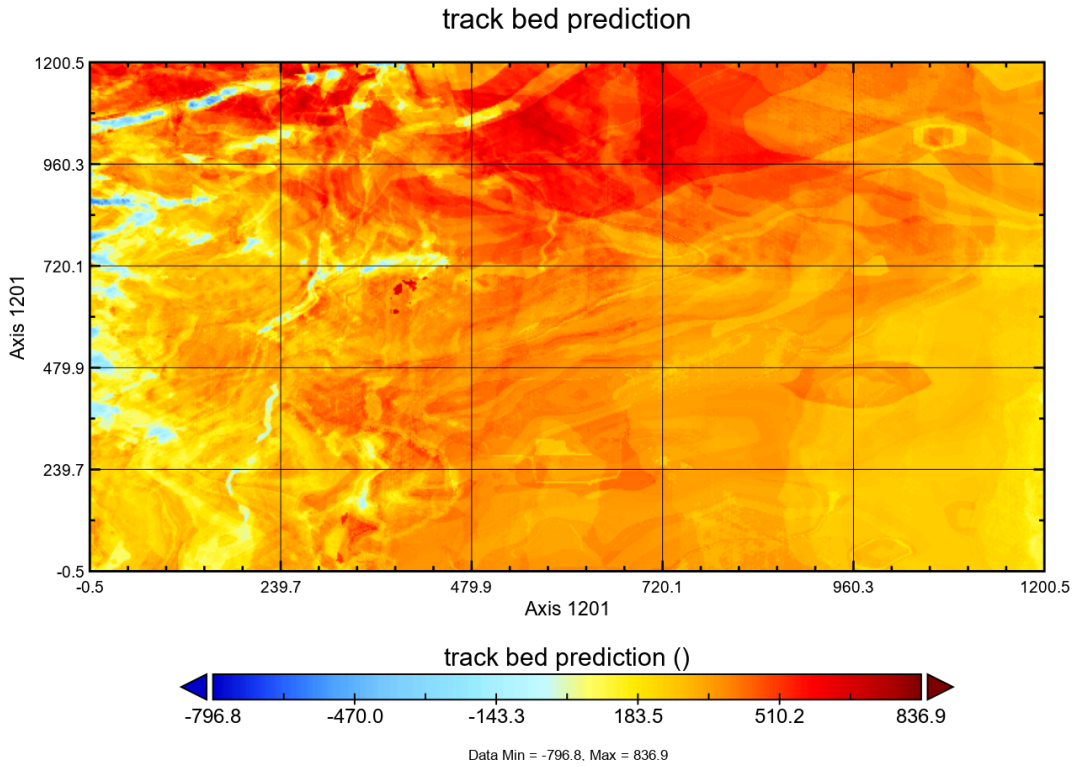
XGB-T4 Magnitude Only*



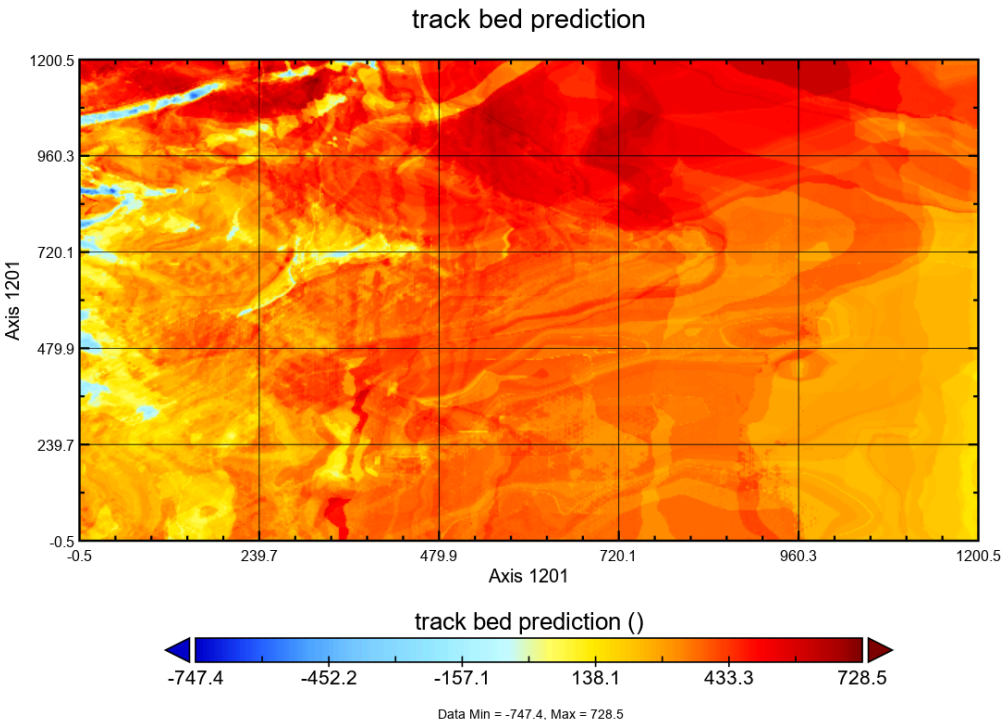
Temp write up

- Adding surf helps with the RMSE score and the R^2 score but adding surf causes strange streaks across the map
- Dropout does not help the RMSE or the R^2 enough to matter. More outlier dropout (better statistics) is a more generalized image that's flatter and missing definition of high and low points
- More depth in the trees leads to lower RMSE and closer predictions
    - Less depth in trees leads to more exaggerated predictions of high points (wanted) ~4-6
- Boosting/increasing eta reduces RMSE and increases R^2
- Revert back to magnitude only XGB file.
- Need RMSE into 20 - low 30's for a clearer image

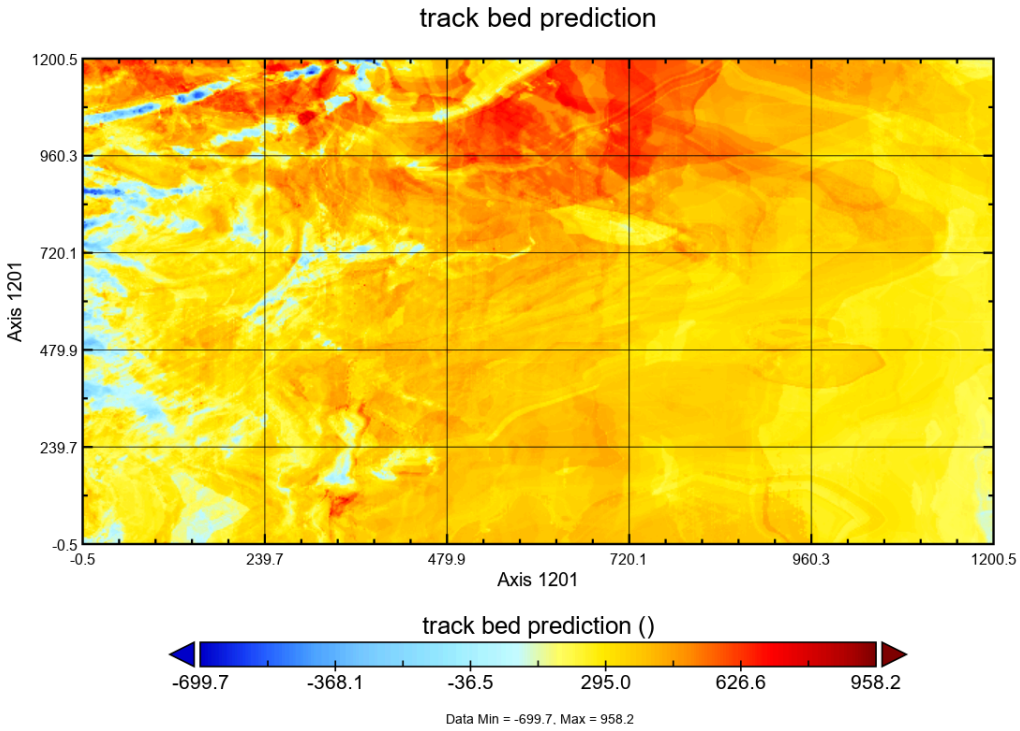
Depth 4, 150 iters, magnitude, not centered on the axis
CLOSEST TO ABOVE



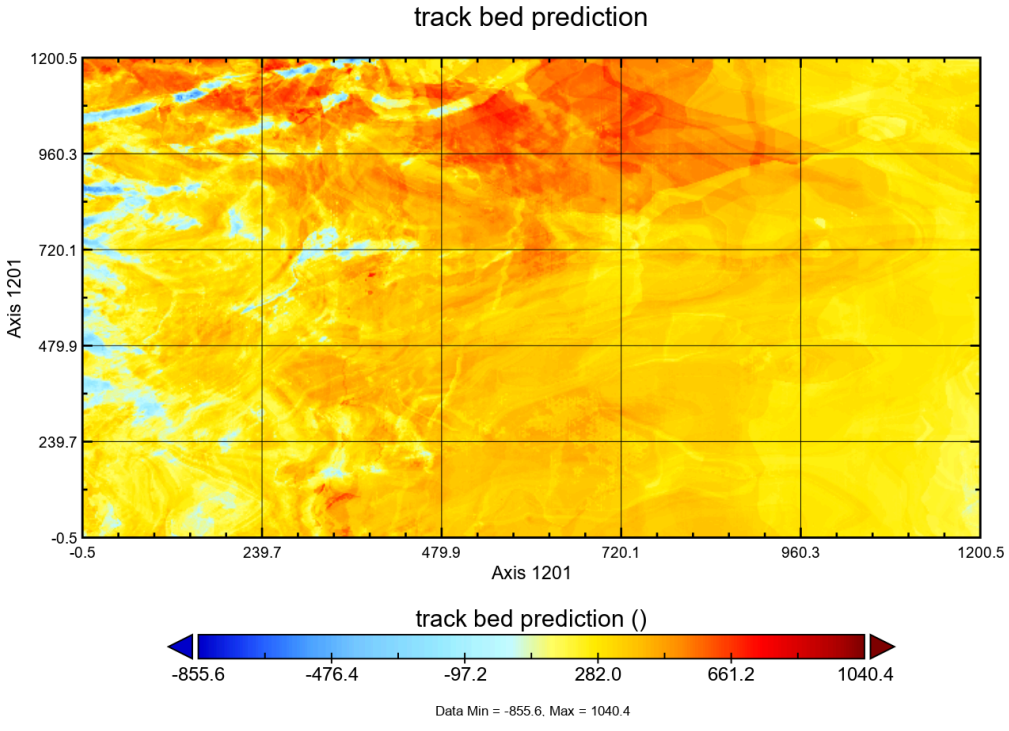
Depth 3, 150 iters, magnitude, not centered on the axis



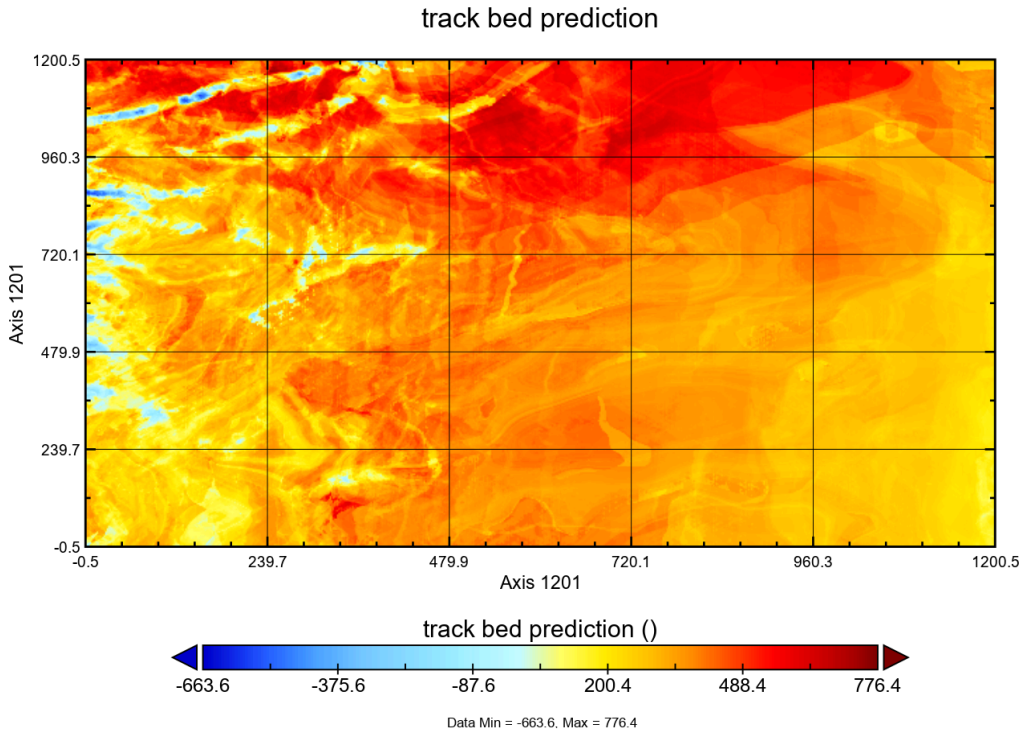
Depth 5, 150 iters, magnitude, not centered on the axis



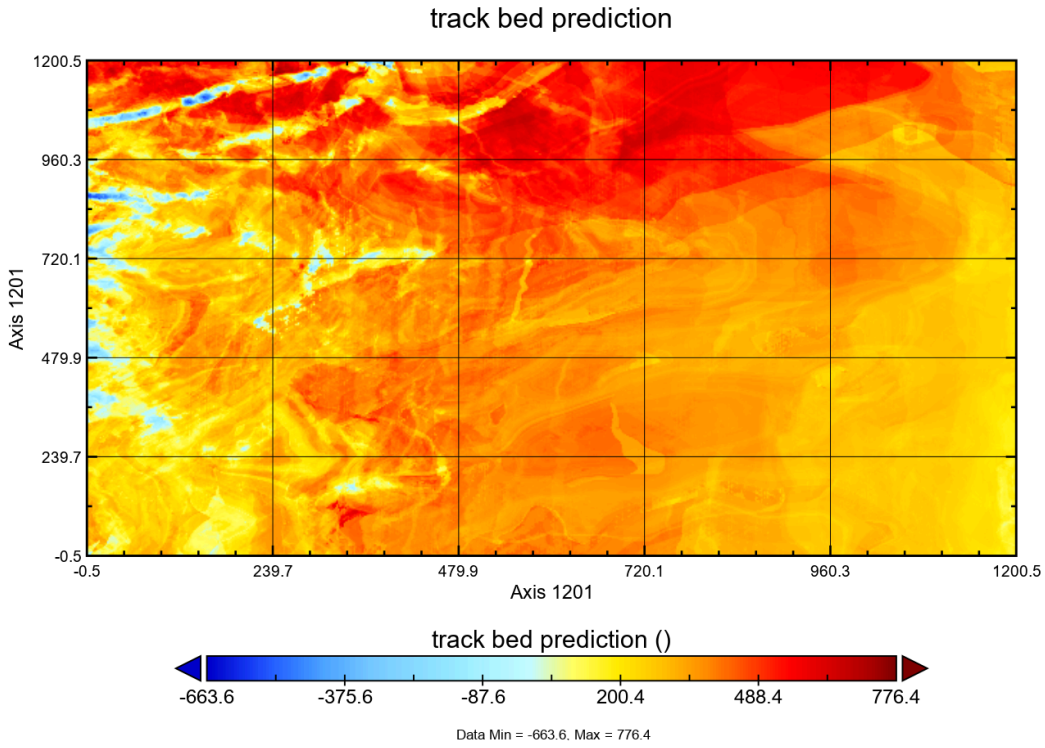
Depth 6, 150 iters, magnitude, not centered on the axis



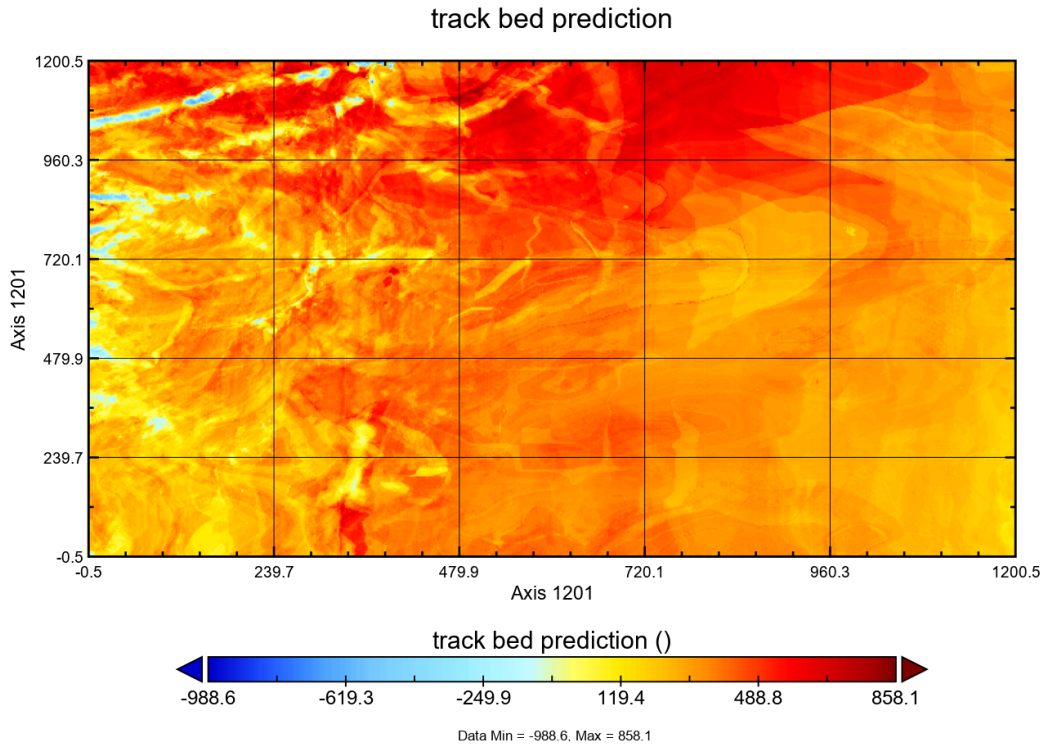
Depth 6, 150 iters, eta .3, early stopping = 20



Depth 6, 150 iters, eta .3 ***



Depth 6, 150 iters, eta .5



Verify 1201 Results

Statistics

Definitions

Euclidean

- Flatten 2D arrays into 1D arrays
- Computes normal (magnitude) of vector from difference of arr1 and arr2
    - 1. arr2 - arr1
    - 2. Sqrt of the sum of squared elements

Cosine Similarity

- Cosine similarity measures the **similarity between two vectors or sets of data based on the angle between them.**
- It ranges from -1 to 1, where 1 indicates identical vectors, 0 indicates unrelated vectors, and -1 indicates opposite vectors.
- Cosine similarity is used in various applications, such as text analysis, information retrieval, and recommendation systems.
- It focuses on the direction or pattern of the vectors, disregarding their magnitudes or scales.
- Cosine similarity is efficient for high-dimensional data, unaffected by vector length.

Pearson Correlation Coefficient

- Pearson correlation measures the **strength** and direction **of the linear relationship between two variables.**
- It quantifies how the values of the variables move together or in opposite directions.
- Pearson correlation ranges from -1 to 1, where 1 indicates a perfect positive linear relationship, 0 indicates no linear relationship, and -1 indicates a perfect negative linear relationship.
- It is widely used in statistics, data analysis, and machine learning to assess the association between variables.
- Pearson correlation is sensitive to outliers and assumes that the relationship between variables is linear and the data is normally distributed.

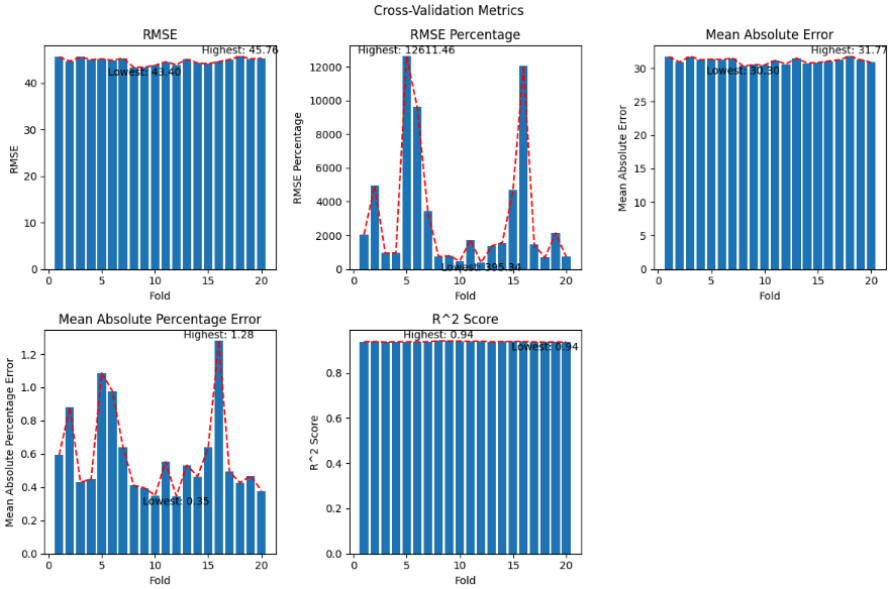
XGBT4, 230710, B6I150E.3S247

```
Euclidean Distance: 95292.953125
Cosine Similarity: 97.745%
Pearson Correlation Coefficient: 80.425%
```

XGBT4, 230710, B6I150E.3S239

```
Euclidean Distance: 92950.453125
Cosine Similarity: 97.853%
Pearson Correlation Coefficient: 81.448%
```

K-Fold Statistics



Jumps in MAPE & RMSE are due to the large range we are working with - this is common.
Constant RMSE ~45, MAE ~32, R^2 ~.90 for 20 folds.

- Seed: 735
- 60-40-20

XGB Final Processing Steps! 🙂

Raw Starter Files; Note: Change all file paths as needed; ensure packages are installed

- train.csv
- test.csv
- y_test.csv

Merging
File: merging_ky_230615.ipynb
Input: raw starter files
Output:

- Test_full.csv
    - This is the merged test only, this file is not actually used in any scripts
- Data_full.csv
    - This is the merged train, test, y_test files.
    - This is the most important file in the scripts.

Derive Velocity Magnitude Feature
File: derive_velocity.ipynb
Purpose: calculate the velocity magnitude of iceflow at (x,y)
Input:

- data_full.csv
- df_1201_validation_data.csv

Output:

- data_full_vMag.csv
- d1201_vMag.csv

Modeling
File: Plotting_Validation_XGB-T4_M_ky_230710.ipynb
Input:

- data_full_vMag.csv
- d1201_vMag.csv
- bed_BedMachine.h5 #1201 physics model

Output: #change as needed, currently output as

- Visualizations

XGB Final Model

File: Plotting_Validation_XGB-T4_M_ky_230710.ipynb

Parameters:

- Generated seed = 168
- 60-40-20 split
- depth_ = 7 #higher --> Lower RMSE; lower --> more extreme highs and lows
- iters_ = 350 #higher is higher R^2 and more reflection on the detail
- eta_ = .25 #higher is more overfitting
- max_depth=depth_,
- n_estimators=iters_,
- min_child_weight=0.25,
- subsample=0.8,
- eta=eta_,
- seed=generated)

Statistics:
Training time:
CPU times: user 3min 36s, sys: 323 ms, total: 3min 36s
Wall time: 2min 8s

Model Prediction Beginning
Model predicted.
Transform data back to original scale.
CPU times: user 3.84 s, sys: 6.45 ms, total: 3.84 s
Wall time: 2.73 s

Print testing stats statements.
RMSE: 32.67978646085573
RMSE Percentage: 1453.64394168839
RMSE Percentage-1: 214.14640577887783
Mean Absolute Error: 22.2728335872512
Mean Absolute Percentage Error: 0.34093320863626686
R^2 Score: 0.9670683089432893

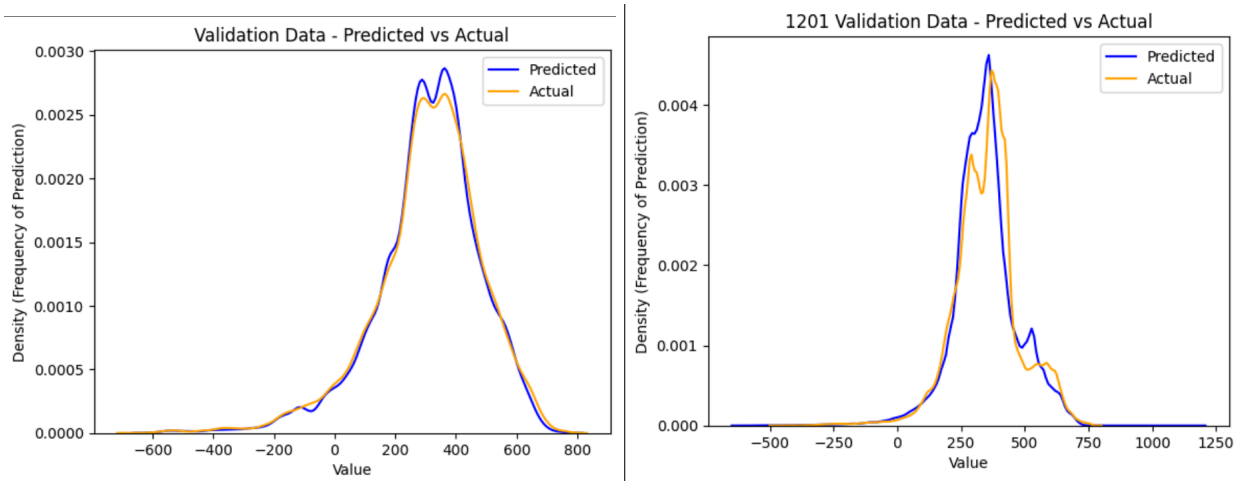
Print validation stats statements.
RMSE: 32.284464365975296
RMSE Percentage: 12894.338512283366
Mean Absolute Error: 22.13960268173217
Mean Absolute Percentage Error: 1.0216433060319823
R^2 Score: 0.9673897122924211
CPU times: user 1.21 s, sys: 0 ns, total: 1.21 s
Wall time: 659 ms

Predicting 1201
Predicting 1201 Complete.
Time taken: 12.964ms

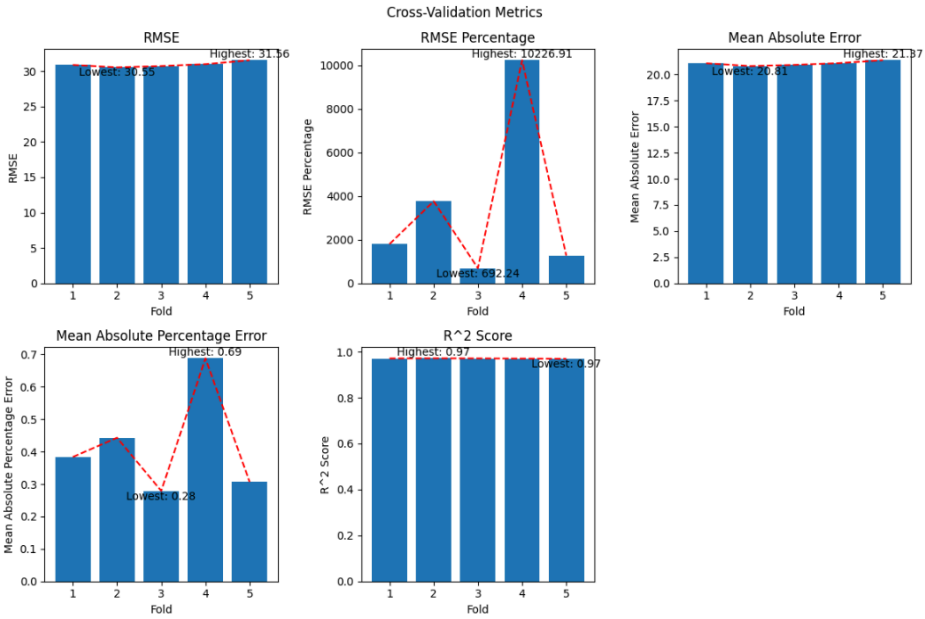
Predicted 1201 compared to the Physics Model
Euclidean Distance: 96433.46875
Cosine Similarity: 97.688%
Pearson Correlation Coefficient: 80.611%

KFold Cross Validation; k = 5
CPU times: user 24min 51s, sys: 1.91 s, total: 24min 52s
Wall time: 15min 11s

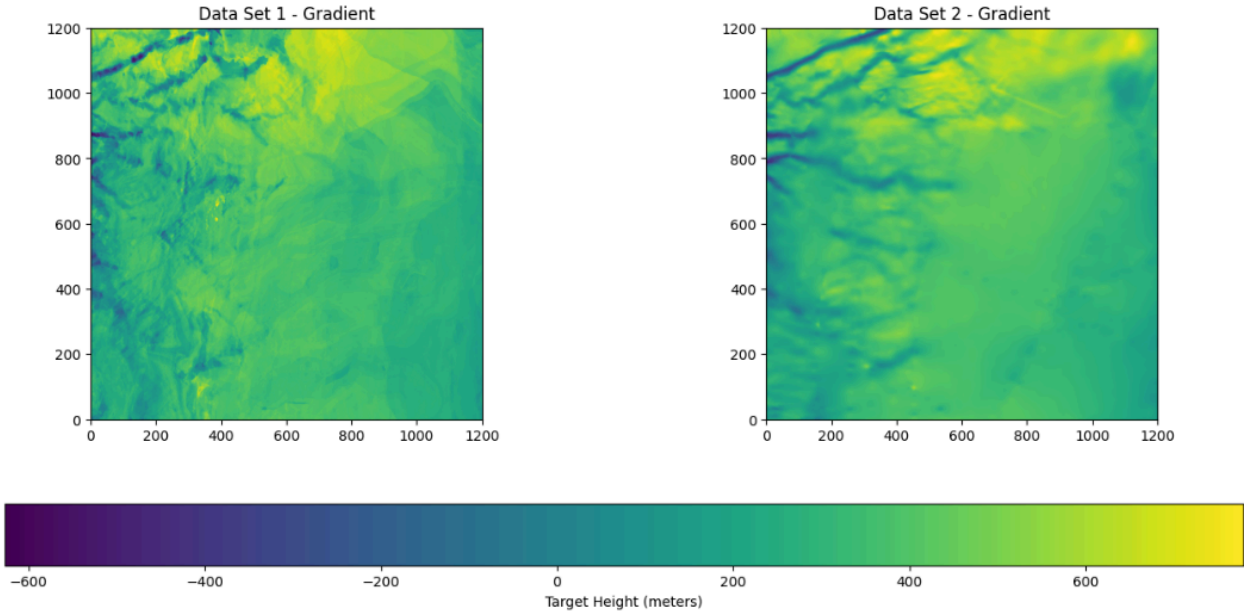
Viz:

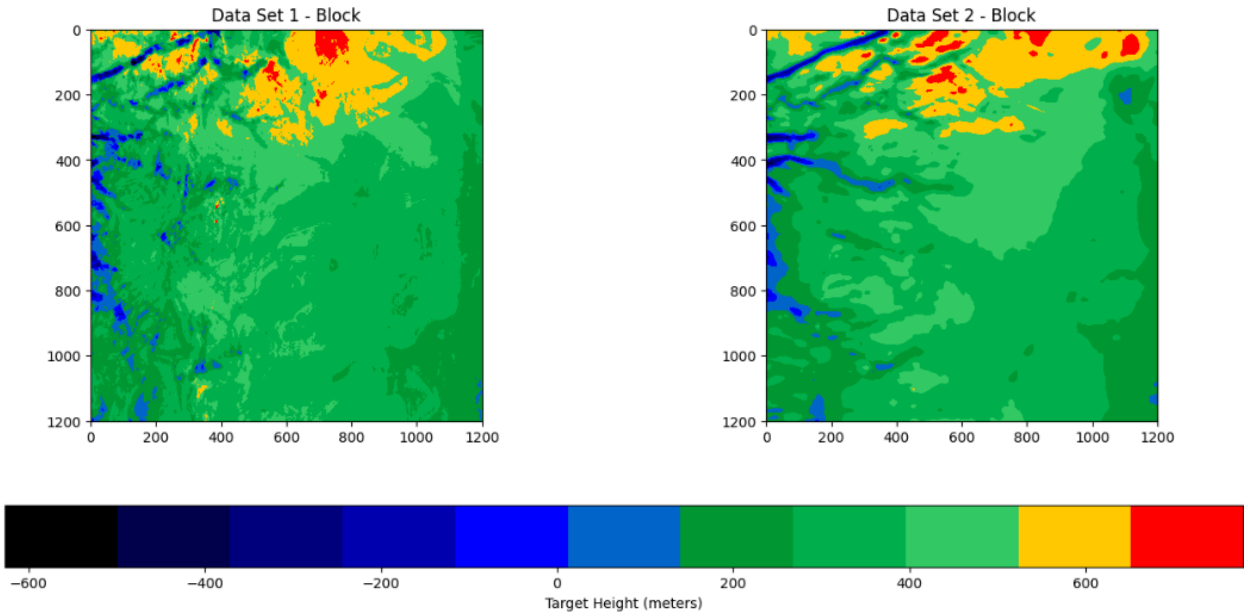


RMSE: [30.55, 31.56]
MAE: [20.81, 21.37]
R^2: .97
Jumps in MAPE & RMSE are due to the large range we are working with - this is common.



Maps:


Data Set 1 - Gradient
Data Set 2 - Gradient
0
200
400
600
800
1000
1200
−600
−400
−200
0
200
400
600
Target Height (meters)

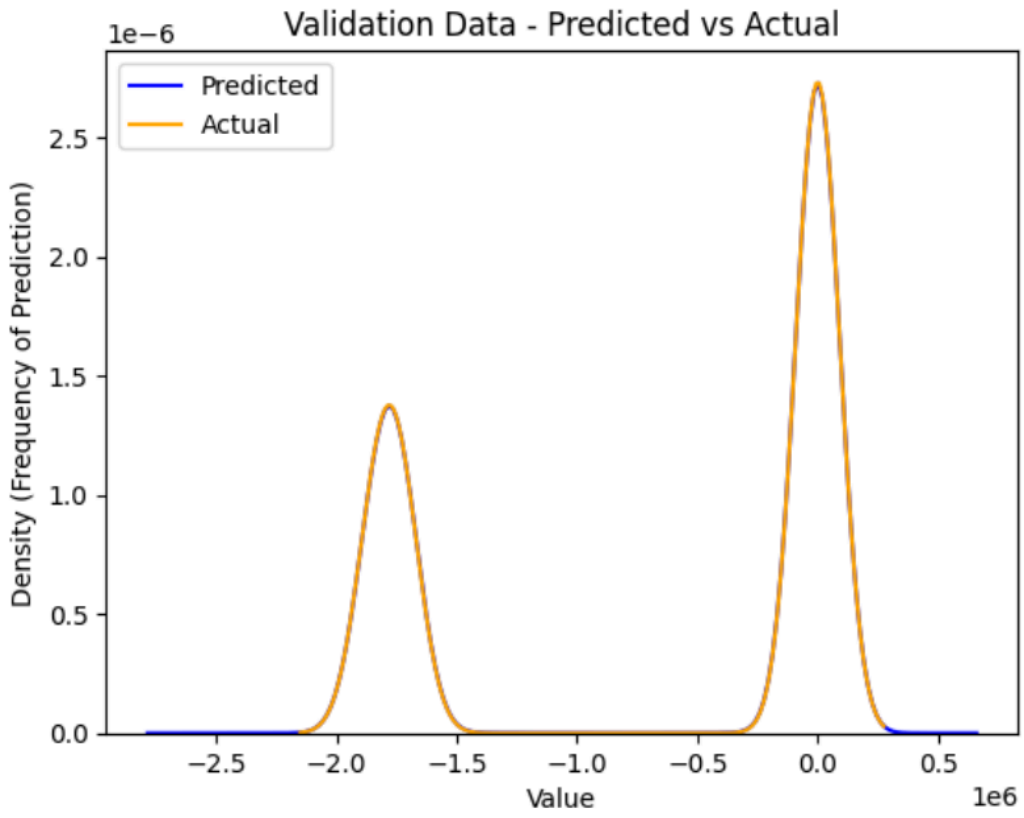

Data Set 1 - Block
Data Set 2 - Block
0
200
400
600
800
1000
1200
−600
−400
−200
0
200
400
600
Target Height (meters)

Bilinear Interpolation with XGBT4_M Final

File:
/REU 2023 Team 1: Ice Bed Topography Prediction/Research/Yi_Work/Bilinear/XGB-T4_M_bilinear_ky_230714.ipynb

Input:
mainPath = '/content/gdrive/MyDrive/REU 2023 Team 1: Ice Bed Topography Prediction/Research/Yi_Work'
data_full_ = mainPath + '/Bilinear/data_full_bilinear_vMag.csv' #training and test combined

```
Print bilinear testing stats statements.
RMSE: 28136.567841359218
RMSE Percentage: 106557.47749702883
RMSE Percentage-1: 132476.835811231
Mean Absolute Error: 2088.19176716106
Mean Absolute Percentage Error: 9.047468951925854
R^2 Score: 0.9989359027012221
```



XGB Tuning Model per Feedback

Note: this adjusts and moves into confirmation bias wants.

Trial 1

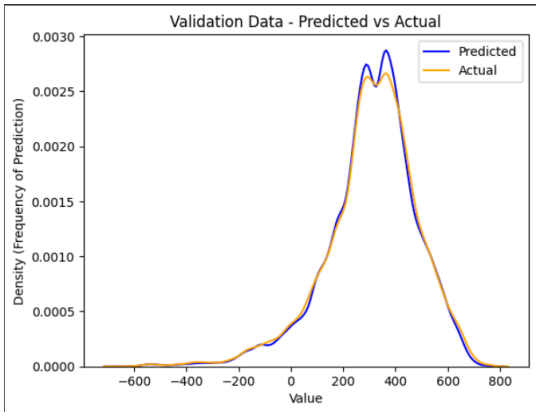
File: Final Process Adjusted > Copy of 2_XGBT4M_full_ky_230711.ipynb
Parameters

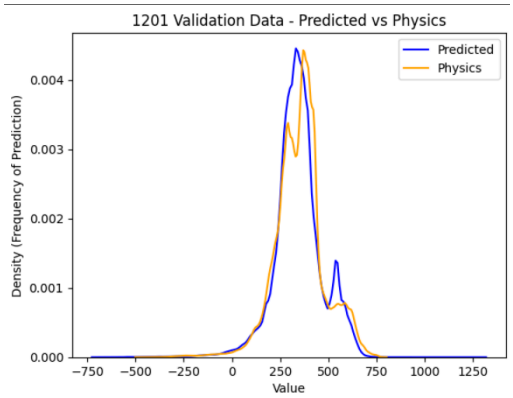
- depth_ = 9 #higher --> Lower RMSE; lower --> more extreme highs and lows
- iters_ = 2000 #higher is higher R^2 and more reflection on the detail
- eta_ = .1 #higher is more overfitting
- model = XGBRegressor(
- max_depth=depth_,
- n_estimators=iters_,
- min_child_weight=0.7, #.4
- subsample=0.5,
- eta=eta_,
- seed=generated)

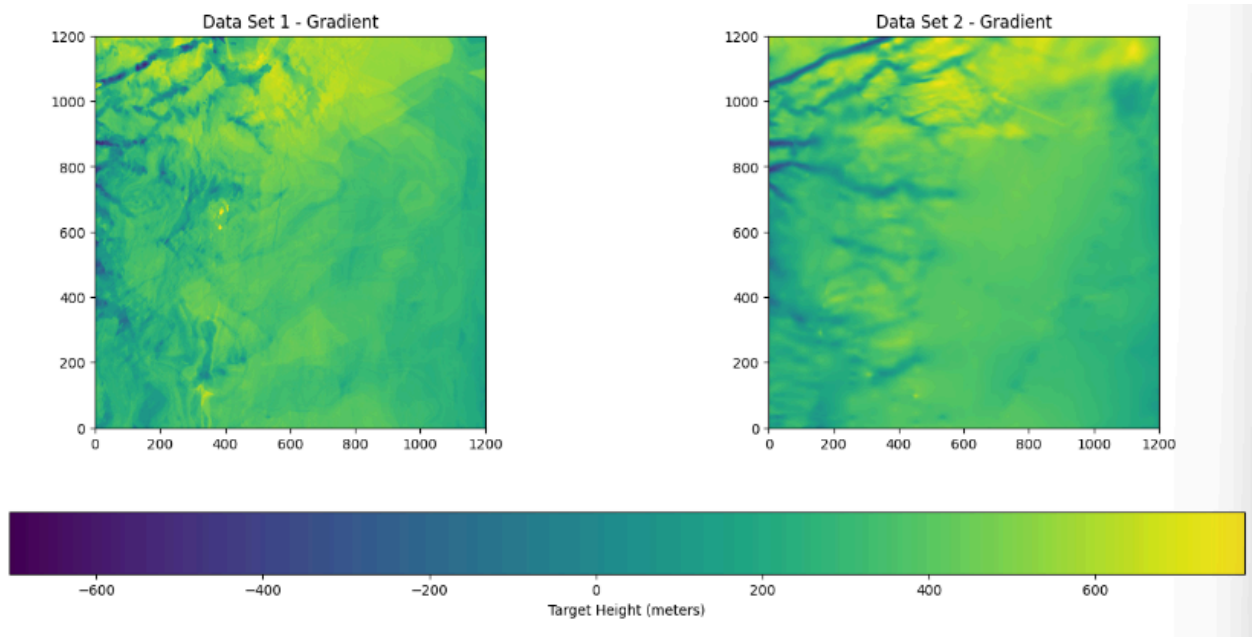
Training time:
CPU times: user 28min 28s, sys: 4.36 s, total: 28min 32s
Wall time: 17min 17s

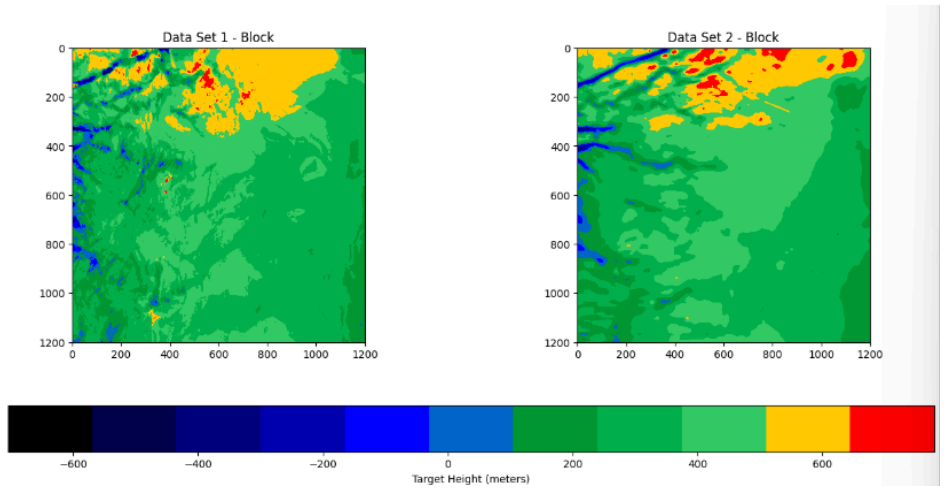
```
Print testing stats statements.
RMSE: 28.394277897758194
RMSE Percentage: 1081.2353070179101
RMSE Percentage-1: 187.3926754210525
Mean Absolute Error: 18.943541900972257
Mean Absolute Percentage Error: 0.28575903419226084
R^2 Score: 0.9751390758024285
```

```
Print validation stats statements.
RMSE: 28.07472578710599
RMSE Percentage: 11925.589900488085
Mean Absolute Error: 18.805549146123155
Mean Absolute Percentage Error: 0.8492859327722422
R^2 Score: 0.9753396914701693
CPU times: user 12.3 s, sys: 29.7 ms, total: 12.4 s
Wall time: 8.38 s
```


Validation Data - Predicted vs Actual
Predicted
Actual
Density (Frequency of Prediction)
Value


1201 Validation Data - Predicted vs Physics
Predicted
Physics
Density (Frequency of Prediction)
Value


Data Set 1 - Gradient
Data Set 2 - Gradient
Target Height (meters)


Data Set 1 - Block
Data Set 2 - Block
Target Height (meters)

Trial 2

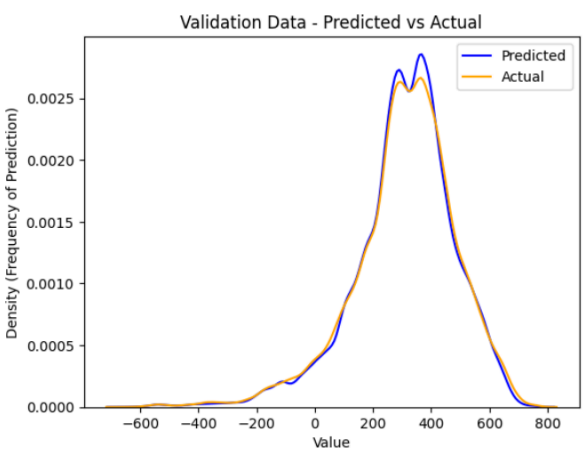
File: Final Process Adjusted > trial2_XGBT4M_full_ky_230711.ipynb
Parameters:

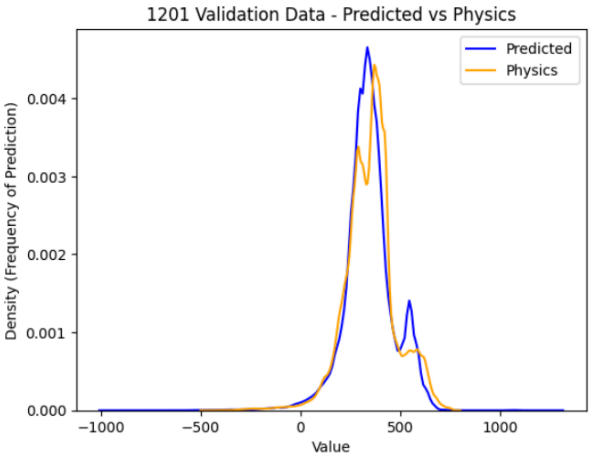
- depth_ = 9 #higher --> Lower RMSE; lower --> more extreme highs and lows
- iters_ = 5000 #higher is higher R^2 and more reflection on the detail
- eta_ = .05 #higher is more overfitting
- #define
- model = XGBRegressor(
- max_depth=depth_,
- n_estimators=iters_,
- min_child_weight=0.8, #.4
- subsample=0.4,
- eta=eta_,
- seed=generated)

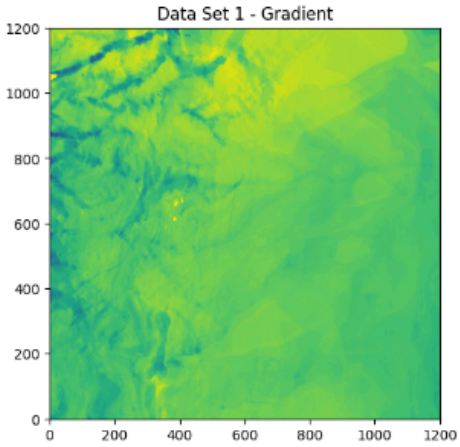
Training time:
CPU times: user 1h 3min 32s, sys: 5.37 s, total: 1h 3min 37s
Wall time: 37min 39s

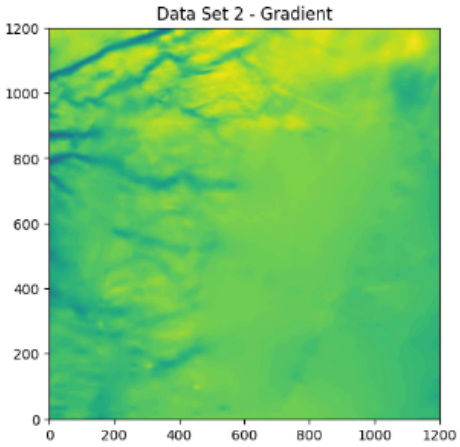
```
Print testing stats statements.
RMSE: 28.40860334998326
RMSE Percentage: 918.2757664168845
RMSE Percentage-1: 181.65369065633166
Mean Absolute Error: 19.041300020971395
Mean Absolute Percentage Error: 0.27701739825409943
R^2 Score: 0.9751139838579286
```

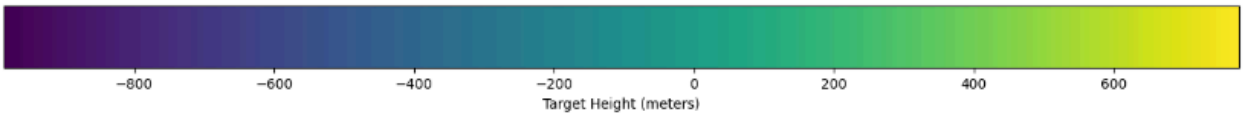
```
Print validation stats statements.
RMSE: 28.077162846582556
RMSE Percentage: 11741.47809707884
Mean Absolute Error: 18.889966278310055
Mean Absolute Percentage Error: 0.8373679878817291
R^2 Score: 0.9753354099503767
CPU times: user 27 s, sys: 20.9 ms, total: 27 s
Wall time: 15.8 s
```

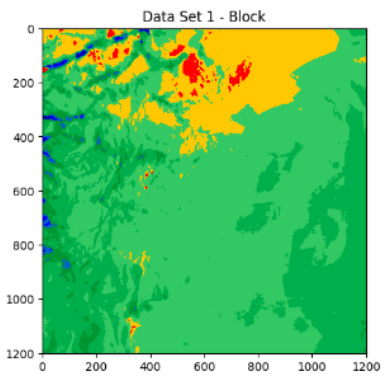

Validation Data - Predicted vs Actual
Predicted
Actual
Density (Frequency of Prediction)
Value

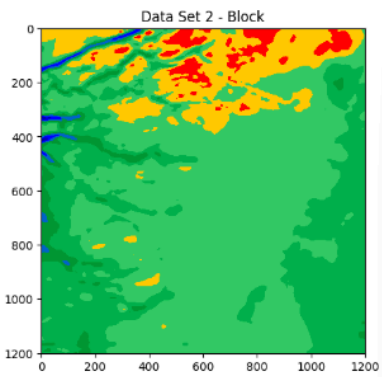

1201 Validation Data - Predicted vs Physics
Predicted
Physics
Density (Frequency of Prediction)
Value

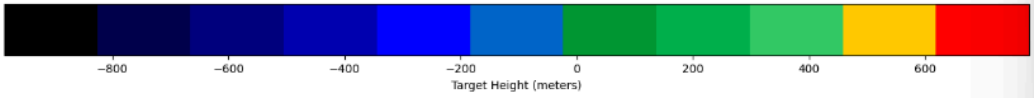

Data Set 1 - Gradient


Data Set 2 - Gradient


Target Height (meters)


Data Set 1 - Block


Data Set 2 - Block


Target Height (meters)

Trial 3

File: Final Process Adjusted > trial3_XGBT4M_full_ky_230711.ipynb
Parameters:

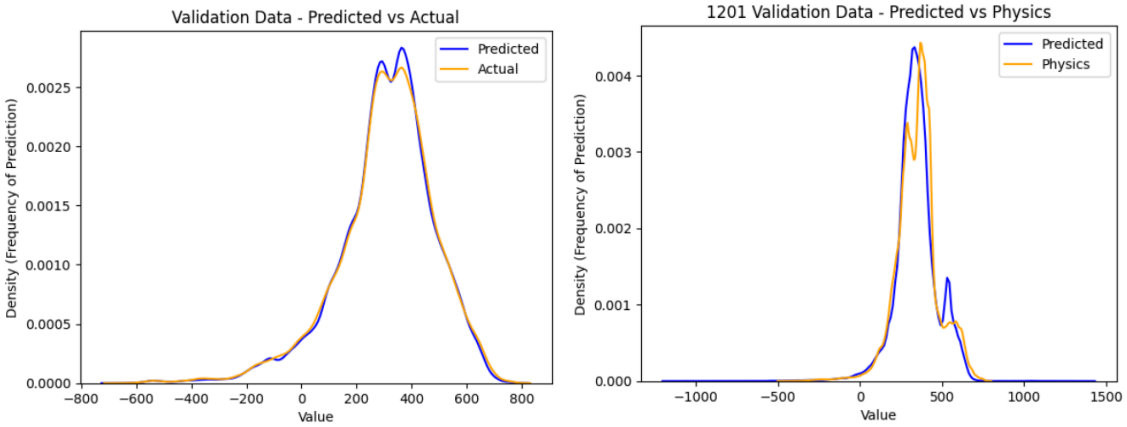
- depth_ = 9 #higher --> Lower RMSE; lower --> more extreme highs and lows
- iters_ = 5000 #higher is higher R^2 and more reflection on the detail
- eta_ = .1 #higher is more overfitting
- model = XGBRegressor(
- max_depth=depth_,
- n_estimators=iters_,
- min_child_weight=0.7, #.4
- subsample=0.5,
- eta=eta_,
- seed=generated)

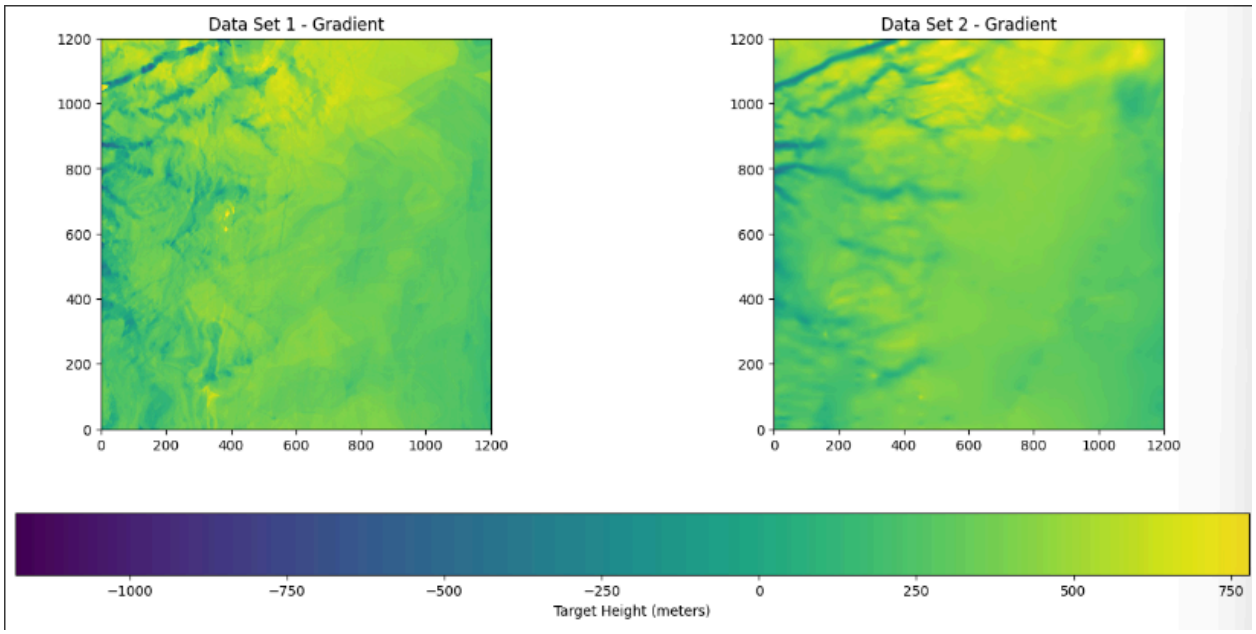
Training time:

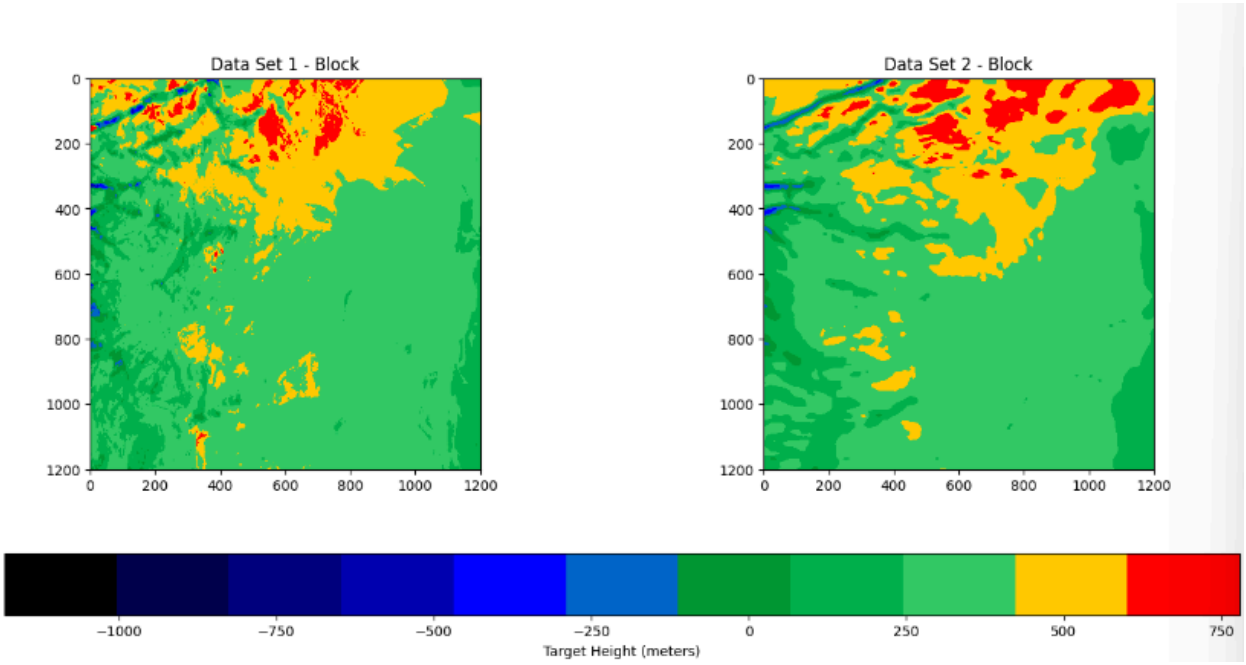
- CPU times: user 1h 5min 40s, sys: 4.95 s, total: 1h 5min 45s
- Wall time: 38min 32s

```
Print validation stats statements.
RMSE: 26.258241809407846
RMSE Percentage: 11488.49279827322
Mean Absolute Error: 17.159561292909387
Mean Absolute Percentage Error: 0.7822882318388792
R^2 Score: 0.9784275859950909
CPU times: user 26.6 s, sys: 26.3 ms, total: 26.6 s
Wall time: 15.5 s
```

```
Print testing stats statements.
RMSE: 26.538682873701006
RMSE Percentage: 783.5621106923726
RMSE Percentage-1: 169.76961487625638
Mean Absolute Error: 17.298451250326163
Mean Absolute Percentage Error: 0.25115266424444294
R^2 Score: 0.9782822743768396
```


Validation Data - Predicted vs Actual
Predicted
Actual
Density (Frequency of Prediction)
Value
1201 Validation Data - Predicted vs Physics
Predicted
Physics
Density (Frequency of Prediction)
Value


Data Set 1 - Gradient
Data Set 2 - Gradient
Target Height (meters)


Data Set 1 - Block
Data Set 2 - Block
0
200
400
600
800
1000
1200
−1000
−750
−500
−250
0
250
500
750
Target Height (meters)

Additional Metrics

Per the request of Dr. Wong, we are interested in the use of additional metrics. The metrics explored are in the following sections.

Terrain Ruggedness Index (TRI)

Background

Link: Terrain Ruggedness Index (TRI) and Vector Ruggedne... - Esri Community
Why? Quantifying characteristics of terrain
What? Expresses the amount of elevation difference between adjacent cells of DEM. Measures the difference between the center cell and eight cells directly surrounding it. These differences are squared and averaged then the sq rt of this result is taken for the TRI of the center cell. This is calculated for every cell.

Common reference:
Riley, S. J., S. D. DeGloria and R. Elliot (1999). A terrain ruggedness index that quantifies topographic heterogeneity, Intermountain Journal of Sciences, vol. 5, No. 1-4, pp.23-27.

Other sources:

- Higher TRI values suggest greater terrain ruggedness or complexity compared to lower TRI values.

What would this tell us?

This would tell us how much our predictions are capturing the terrain.
Try: calculating the TRI for the known targets then run the TRI on the predicted validation data

How to compare TRIs

1. 5 summary stats
2. Scatters known on x, predicted y. Want a diagonal line.
3. Difference plot
4. MAE, RMSE, R^2

Implementation

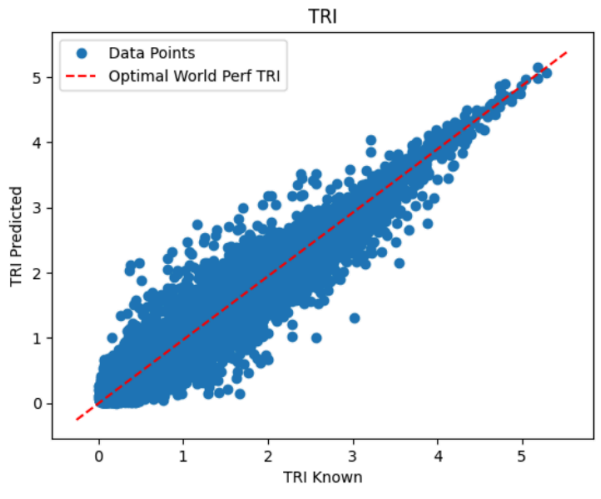
File: Final Process Adjusted > TRI_XGBT4M_full_ky_230718.ipynb

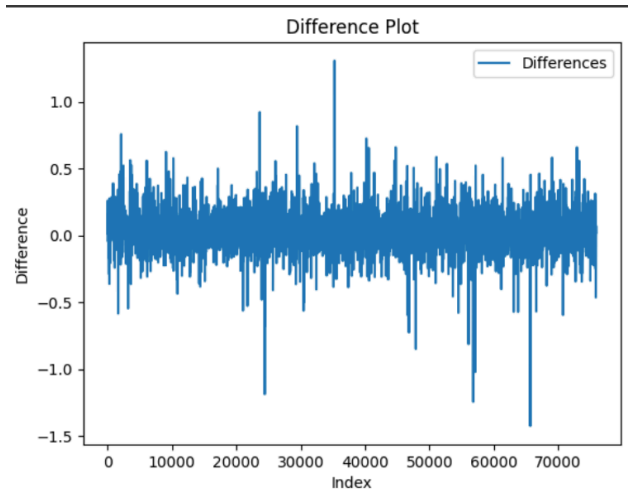
Results

Still Scaled

Interpret: very rough terrain. Model captures it with 93% R^2.

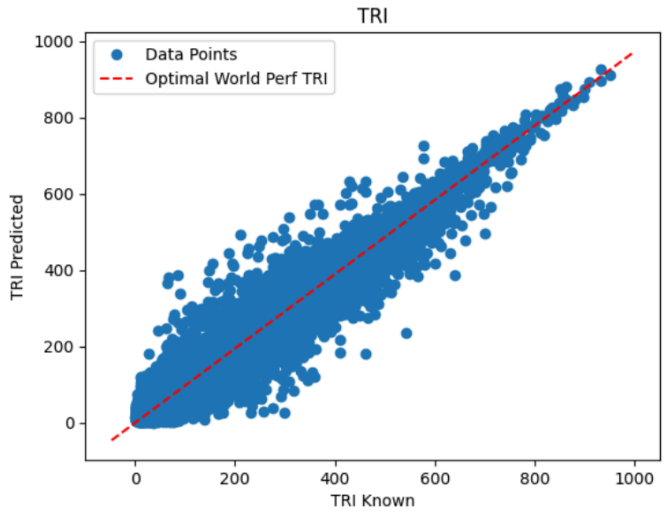
```
Print Terrain Roughness Index Stats statements.
Known TRI:
Mean: 0.964
Min, Max: 0.006, 5.287
Predicted TRI:
Mean: 0.9229999780654907
Min, Max: 0.0, 5.146999835968018
*****
RMSE: 0.1620612013936246
Mean Absolute Error: 0.11596191864905259
R^2 Score: 0.9315865644673327
*****
CPU times: user 9.8 ms, sys: 0 ns, total: 9.8 ms
Wall time: 9.95 ms
```

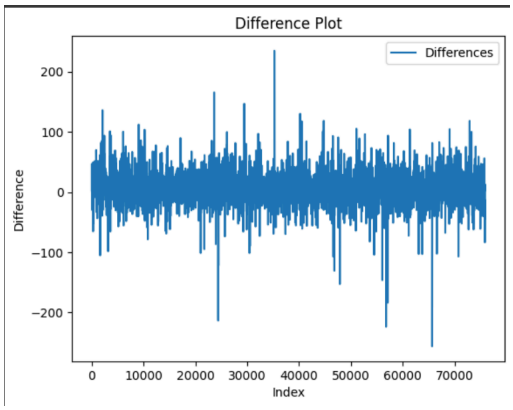


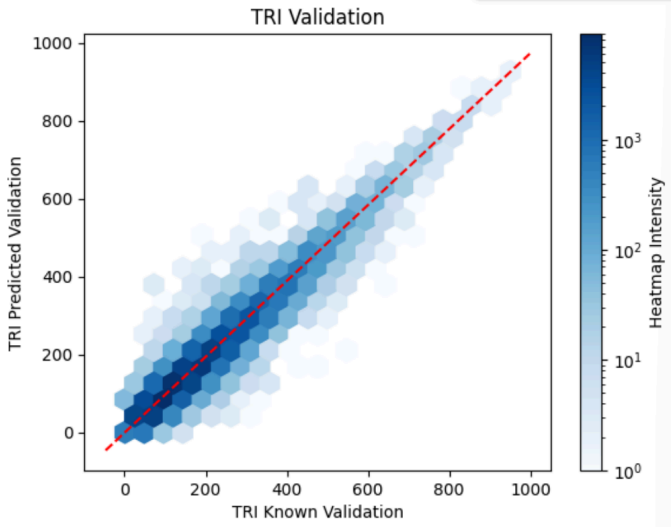


Normal Scale

```
Print Terrain Roughness Index Stats statements.
USING  WORLD SCALE - NOT SCALED DOWN
Known TRI:
Mean: 173.404
Min, Max: 0.994, 951.089
Predicted TRI:
Mean: 166.09300231933594
Min, Max: 0.019999999552965164, 925.8200073242188
*****
RMSE: 29.150830024080047
Mean Absolute Error: 20.858701196205455
R^2 Score: 0.9315865642185107
*****
CPU times: user 10 ms, sys: 0 ns, total: 10 ms
Wall time: 13.8 ms
```





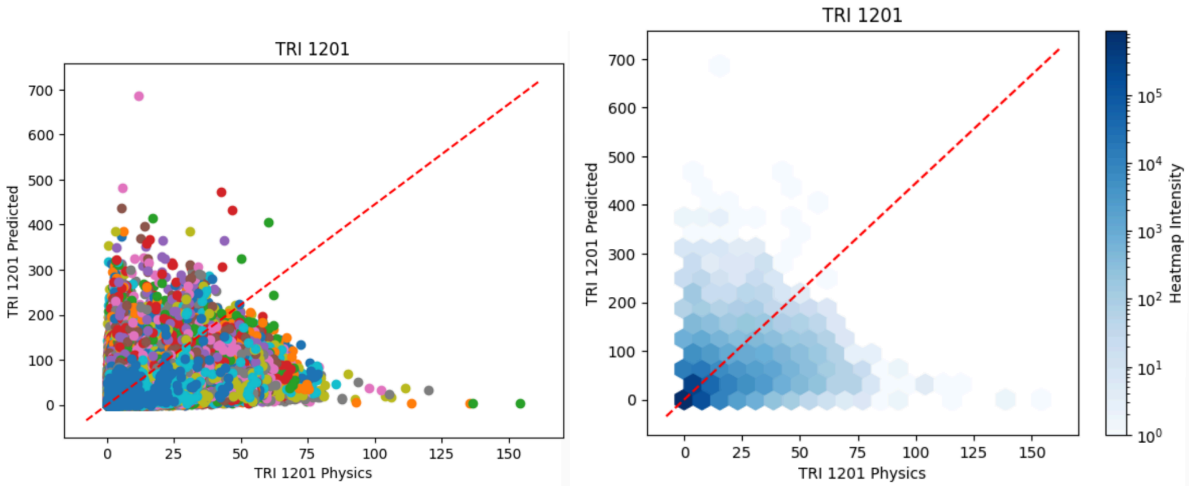


1201 Physics vs. XGB 1201 Predicted

Note: These statistics make sense as the physics model predicts a smooth topography and the XGB predicts a detailed topography.

Additionally, the 1201 predicted should be compared to a true value graph which we do not have, not another graph that shows it has large error in validation prediction scores.
Implement TRI as an early stopping parameter

```
Print Terrain Roughness Index Stats statements.
1201 Physics & 1201 Predicted
Physics TRI:
Mean: 3.302000045776367
Min, Max: 0.008999999612569809, 154.1719970703125
Predicted TRI:
Mean: 14.178000450134277
Min, Max: 0.0, 685.6220092773438
*****
RMSE: 18.867346
Mean Absolute Error: 11.401025
R^2 Score: -59.377096863223905
*****
CPU times: user 33 ms, sys: 3 ms, total: 36 ms
Wall time: 35.2 ms
```



Feedback: expected physics model to be smoother than reality.

Peak Signal-to-Noise Ratio (PSNR)

Background

PSNR (Peak Signal-to-Noise Ratio) is a metric used to quantify the quality of a reconstructed or compressed signal by comparing it to the original signal in terms of the ratio between the maximum possible signal power and the distortion introduced by noise or compression artifacts.

Python | Peak Signal-to-Noise Ratio (PSNR) - GeeksforGeeks

Implementation & Results

File: Final Process Adjusted> PSNR_XGBT4M_full_ky_230718.ipynb

Session Crashes due to RAM limitations.
POV: Not worth running.

Kriging in XGB

mainPath = '/content/gdrive/MyDrive/REU 2023 Team 1: Ice Bed Topography Prediction/Research/Yi_Work/Kriging/'
Derive V_Mag:
File: Kriging_1_derive_velocity_ky_230719.ipynb
Input: data_full_ = mainPath + 'angelina-kriging-interp-merged.csv' #training and test combined

Modeling

File: Kirging_2_XGBT4M_full_ky_230719.ipynb
Input: data_full_ = mainPath + '0_Data/data_full_vMag.csv' #training and test combined

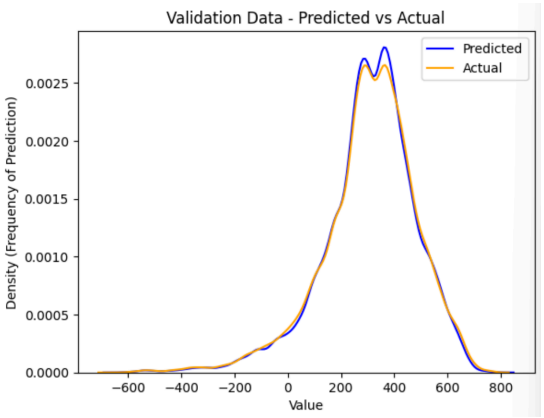
Results

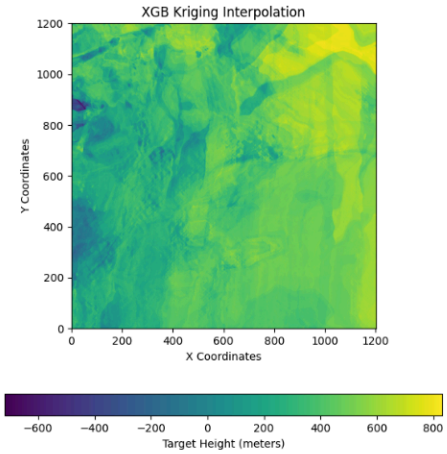
Data train-split complete with: 60.0% training, 40.0% testing, 20.0% validation

Train time:
CPU times: user 6min 13s, sys: 461 ms, total: 6min 14s
Wall time: 3min 43s

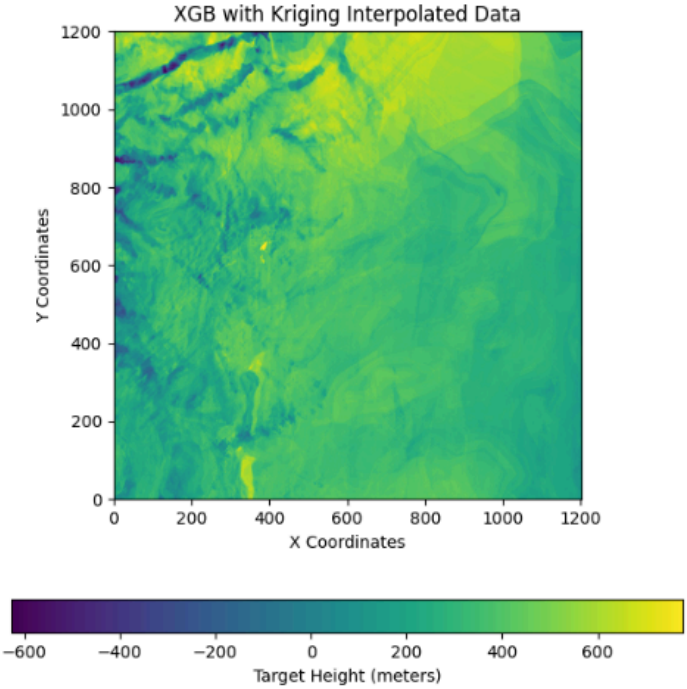
Print testing stats statements.
RMSE: 27.099901048231793
RMSE Percentage: 5579.869358578423
RMSE Percentage-1: 256.011761268177
Mean Absolute Error: 17.947478474119105
Mean Absolute Percentage Error: 0.4549929827981801
R^2 Score: 0.9773330670387878

Print validation stats statements.
RMSE: 26.94614238876861
RMSE Percentage: 1807.1591700932408
Mean Absolute Error: 17.97173574116191
Mean Absolute Percentage Error: 0.3371417739256386
R^2 Score: 0.977406577024783
CPU times: user 1.19 s, sys: 4.99 ms, total: 1.2 s
Wall time: 628 ms





Kriging interpolated predictions on the same scale as XGB with NNeighbors scale



Print Terrain Roughness Index on Kriging Stats statements.
Known TRI:
Mean: 174.113
Min, Max: 0.889, 908.215
Predicted TRI on validation data only:
Mean: 169.4980010986328
Min, Max: 0.7540000081062317, 904.8159790039062
*****

RMSE: 24.635423457002357
Mean Absolute Error: 17.10091135333973
R^2 Score: 0.9512904651141113

*****
CPU times: user 8.41 ms, sys: 0 ns, total: 8.41 ms
Wall time: 9.41 ms

Predicted TRI 1201:
Mean: 15.925999641418457

Kriging First Pass Prediction

File: 2_KrigingFirstGuess_XGBT4_full_ky_230719.ipynb
Goal:

1. Predict the residuals using the 5 features and kriging first pass guess
2. Add predicted residuals to the kriging first pass
3. Run the statistics on the new column and target column.

Predicting Residuals

Test 1

Params:

- depth_ = 15 #higher --> Lower RMSE; lower --> more extreme highs and lows
- iters_ = 200 #higher is higher R^2 and more reflection on the detail
- eta_ = .4 #higher is more overfitting
- #define
- model = XGBRegressor(
- max_depth=depth_,
- n_estimators=iters_,
- # min_child_weight=0.25,
- subsample=0.8,
- eta=eta_,
- seed=generated)

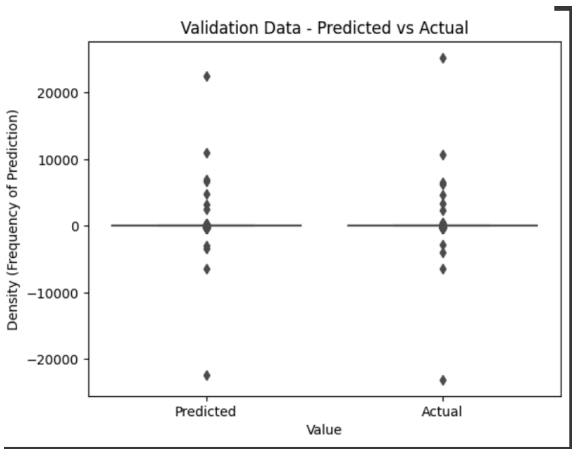
Statistics:
Train time:

```
CPU times: user 6min 11s, sys: 826 ms, total: 6min 12s
Wall time: 3min 46s
```

```
*****
Print testing stats statements.
RMSE: 26.48587056274223
Mean Absolute Error: 7.186677489783802
R^2 Score: 0.9753453837317544
CPU times: user 5.72 s, sys: 17 ms, total: 5.73 s
Wall time: 2.97 s
```

```
Print validation stats statements.
RMSE: 20.157991406889384
Mean Absolute Error: 7.109959529546217
R^2 Score: 0.9793067768211063
CPU times: user 1.71 s, sys: 3.97 ms, total: 1.72 s
Wall time: 900 ms
```

```
predicted mean: 0.2666322588920593
kriging mean: 0.2408553252020066
predicted max: 22533.109375
kriging max: 25240.1234241128
predicted min: -22425.71875
kriging min: -23084.962469115857
```



Test 2

Params:

- depth_ = 20 #higher --> Lower RMSE; lower --> more extreme highs and lows
- iters_ = 50 #higher is higher R^2 and more reflection on the detail
- eta_ = .4 #higher is more overfitting
- #define
- model = XGBRegressor(
- max_depth=depth_,
- n_estimators=iters_,
- # min_child_weight=0.25,
- subsample=0.8,
- eta=eta_,
- seed=generated)

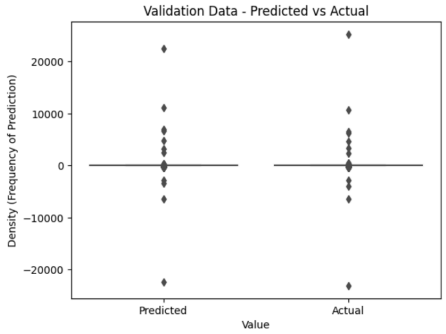
Statistics:

Train time: `CPU times: user 1min 58s, sys: 293 ms, total: 1min 59s Wall time: 1min 11s`

```
*****
Print testing stats statements.
RMSE: 26.219632438511486
Mean Absolute Error: 6.802811734641588
R^2 Score: 0.9758385529367214
CPU times: user 1.86 s, sys: 9.83 ms, total: 1.87 s
Wall time: 1.23 s
```

```
Print validation stats statements.
RMSE: 19.669794958463523
Mean Absolute Error: 6.717045105768216
R^2 Score: 0.9802969574247333
CPU times: user 562 ms, sys: 983 µs, total: 563 ms
Wall time: 554 ms
```

```
predicted mean: 0.2654353678226471
kriging mean: 0.2408553252020066
predicted max: 22516.615234375
kriging max: 25240.1234241128
predicted min: -22441.177734375
kriging min: -23084.962469115857
```



Test 3

Params:

- depth_ = 20 #higher --> Lower RMSE; lower --> more extreme highs and lows
- iters_ = 50 #higher is higher R^2 and more reflection on the detail
- eta_ = .25 #higher is more overfitting
- #define
- model = XGBRegressor(
- max_depth=depth_,
- n_estimators=iters_,
- # min_child_weight=0.25,
- subsample=0.8,
- eta=eta_,
- seed=generated)

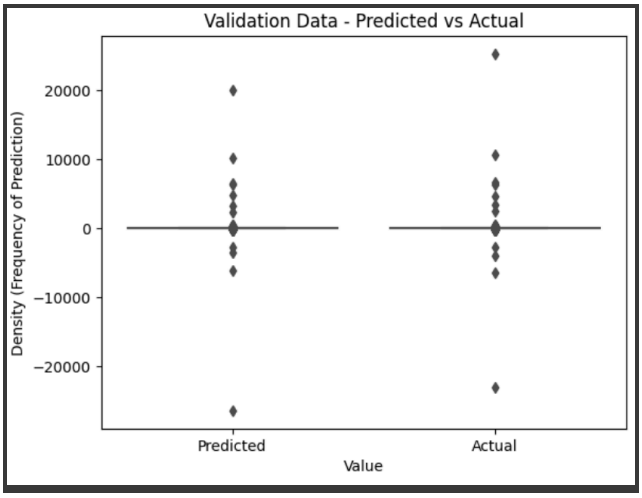
Statistics:

Train time: `CPU times: user 1min 55s, sys: 231 ms, total: 1min 55s Wall time: 1min 9s`

```
Print testing stats statements.
RMSE: 30.25234532111452
Mean Absolute Error: 6.544770933612918
R^2 Score: 0.9678346826211868
CPU times: user 1.72 s, sys: 1.98 ms, total: 1.73 s
Wall time: 908 ms
```

```
Print validation stats statements.
RMSE: 27.94027007948376
Mean Absolute Error: 6.509896577128279
R^2 Score: 0.9602447187821451
CPU times: user 522 ms, sys: 1.95 ms, total: 524 ms
Wall time: 275 ms
```

```
predicted mean: 0.1289461702108383
kriging mean: 0.2408553252020066
predicted max: 20000.93359375
kriging max: 25240.1234241128
predicted min: -26480.80078125
kriging min: -23084.962469115857
```



Full Pipeline Kriging First Guess

File: 2_KrigingFirstGuess_XGBT4_full_ky_230719.ipynb

1. I train the XGB model to predict residuals
    a. Parameters to match test 2.
2. Once trained, I run the XGB Predict on the original dataset
3. Then I take the newly predicted residuals and add them to the predicted kriging guesses
4. Then I compare that to the target_track_bed var

```
Kriging-Predicted Residuals vs. Target Track Bed for All Known Data
RMSE: 454.1993690591726
Mean Absolute Error: 392.6122457020212
R^2 Score: -0.4648162681465928
```

Physics Model Only Statistics

This is the Model To Beat and is the model used in practice.

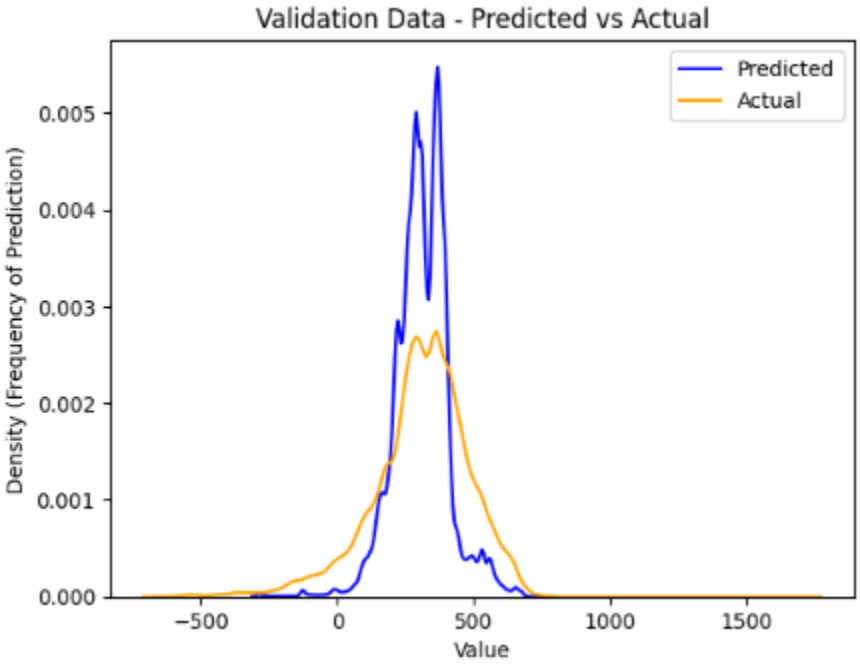
Homayra used bilinear interpolation (backwards?) on the bedMachine dataset to create this dataset.

mainPath = '/content/gdrive/MyDrive/REU 2023 Team 1: Ice Bed Topography Prediction/Research/Yi_Work/Bilinear/bedMachine_bilinear/'

File: physics_model_statistics_ky_230724.ipynb

Input:
data_full_ = mainPath + 'Copy of homayra-physics_train_test_combined.csv'

Results:
Print bilinear physics model stats statements.
RMSE: 213.152279402706
Mean Absolute Error: 162.17780626506877
R^2 Score: -0.40422345235340607

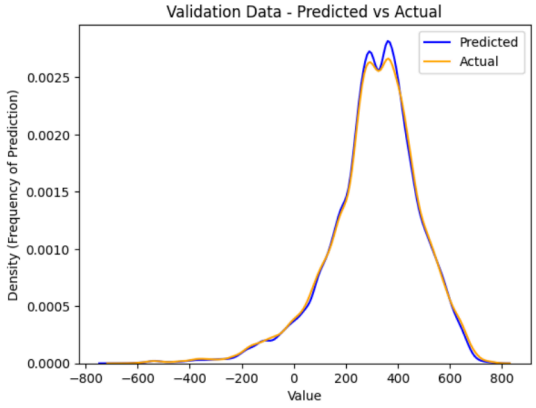


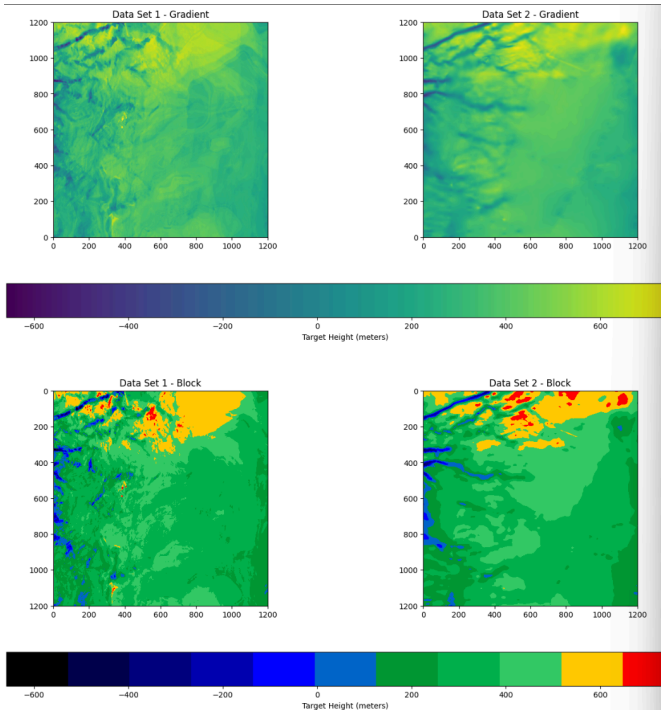
XGB w/ Bilinear Interp

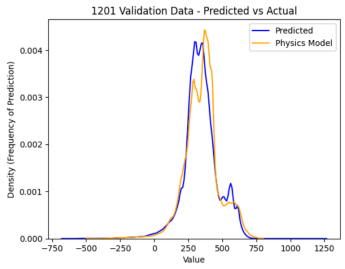
File: Bilinear/XGB-T4_M_bilinear_ky_230714.ipynb
Inputs:
#paths
mainPath = '/content/gdrive/MyDrive/REU 2023 Team 1: Ice Bed Topography Prediction/Research/Yi_Work'
data_full_ = mainPath + '/Bilinear/data_full_bilinear_vMag.csv' #training and test combined
data_1201_ = mainPath + '/Data_derivedVelocity/d1201_vMag.csv'
validation1201_ = mainPath + '/Bilinear/Copy of bed_BedMachine.h5' #physics model provided

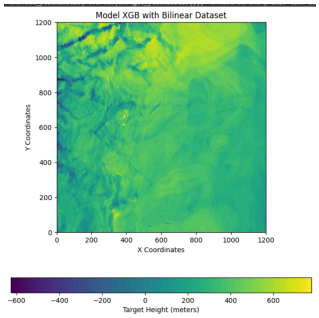
Print bilinear testing stats statements.
RMSE: 27.734627348311506
Mean Absolute Error: 18.35248703990835
R^2 Score: 0.9762807866111508

Print bilinear validation stats statements.
RMSE: 27.30672448164108
RMSE Percentage: 10744.410624895967
Mean Absolute Error: 18.22007376592227
Mean Absolute Percentage Error: 0.7663631795314992
R^2 Score: 0.9766704330972472
CPU times: user 1.25 s, sys: 2.93 ms, total: 1.25 s
Wall time: 863 ms









Print Terrain Roughness Index on Kriging Stats statements.
USING WORLD SCALE - NOT SCALED DOWN
Known TRI:
Mean: 173.404
Min, Max: 0.994, 951.089
Predicted TRI:
Mean: 168.38400268554688
Min, Max: 0.6470000147819519, 945.176025390625
*****

RMSE: 24.84701344105336

Mean Absolute Error: 17.32603411719647
R^2 Score: 0.950296389052055
*****
CPU times: user 11.9 ms, sys: 0 ns, total: 11.9 ms
Wall time: 14.3 ms

Predicted TRI:
Mean: 14.682000160217285

Combined Results All

			Evaluation Metrics for Test Data						
Who	Method	Dataset	RMSE	MAE	R^2	TRI R^2	Interpolation time (estimated)	Train Time	Total time
Katie	NA, Baseline	bedMachine_bilinear interp						NA	
Katie	XGB	KNN_VMag	32.680	22.273	0.967	0.932	50 min	3 min 36 seconds	55 min
Katie	XGB	Kriging_interp.	27.099	17.947	0.977	0.951	5.5 hours	6 min 13 sec	5 hour 40 min
Ray	Dense+LSTM	KNN_VMag	83.937	60.328	0.783				
Ray	LSTM	KNN_VMag	101.630	74.522	0.682				
Jason	Dense	KNN_VMag	104.475	74.381	0.663				
Katie	VAE	KNN	106.884	83.798	0.648				
Katie	VAE into XGB	KNN	129.760	100.035	0.481				
Katie	GPR	KNN	150.086	97.855	0.293				

Katie	STGP	KNN	225.772	126.819	-0.580				
Angelina	Kriging (alone)		190.452	7.841	-0.112				
Katie	XGB for residuals	Kriging residual & krig guess	462.677	410.309	-3.594				

Return to Contents

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (predicted_i - actual_i)^2}{N}}$$

$$MAE = \frac{\sum_{i=1}^{N} |actual_i - predicted_i|}{N}$$

$$R^2 = \frac{SSR}{SST} = \frac{\sum (predicted_i - actual)^2}{\sum (actual_i - \bar{y})^2}$$