

Did Mars ever have environmental conditions that could have supported life?

For this competition, we used data collected with gas chromatography–mass spectrometry (GCMS) chemical analysis method, as those collected by Curiosity rover's SAM instrument suite. There are nine label classes, each indicating presence of material in the sample belonging to the respective rock, mineral, or organic compound families. As this is a multilabel task there can be present more than one classes in sample.

Use of External Data	No
Language	Python
Training platform	Google Colab Pro
Data preparation duration	less than 1.5 hours (can be optimized)
Train duration	Less than 2 hours
Inference duration	40 min data preparation, 15 min inference in CPU (less in GPU) (for both valid and test data)

Table 1. Training Info

Datasets Preparation

As are raw data comes in many files, initially 3 sets of train and test datasets are created for later usage in training. Output shape of each dataset is (number of samples, 600 mz, 500 timesteps). All float raw mz values are rounded to integers. As number of different mz per sample vary, all intensity gaps are zero filled. As number of timesteps per sample vary, if greater than 500 then use the max in between value else repeat following or previous value. Diagram below shows the differences between datasets. Whether a log transformation is applied after root transformation or not, if ion curve over time is smoothed or not and the upper clip value. Finally, datasets are scaled and saved as uint8 data type for compression and later use as image.

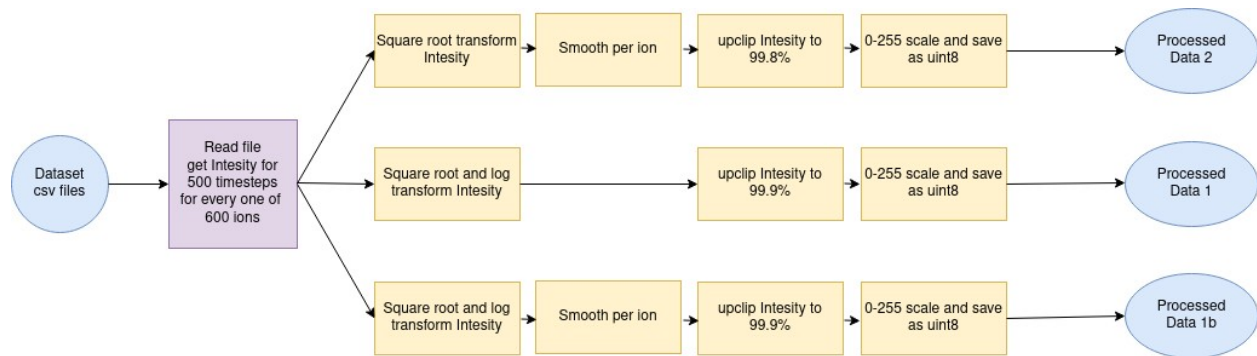


Figure 1. Flowchart of initial dataset creation.

Plots in figure 2 below are from the same sample, showing the difference between Dataset 1 and Dataset 2. Dataset 1 has more noise than Dataset 2 but can capture things that escape Dataset 2 focus. Variation is always an important feature in machine learning. Training models with diverse datasets can increase final ensemble performance.

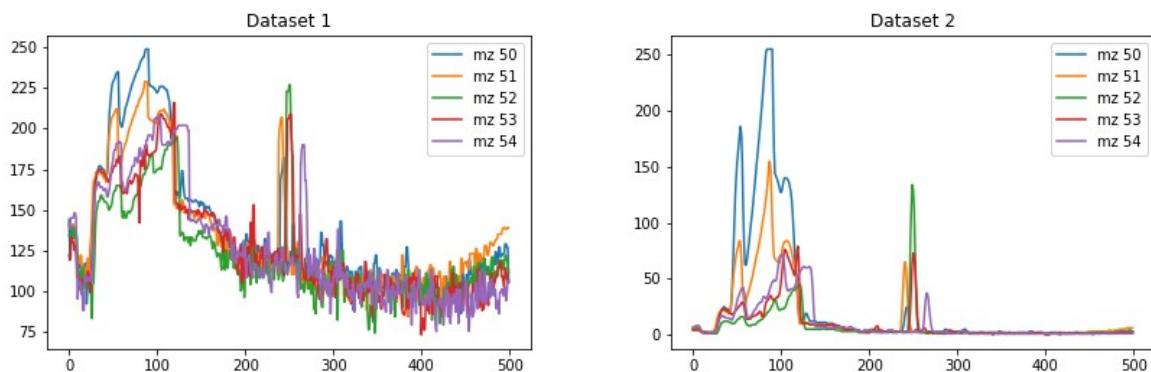


Figure 2. Plots for the same ions of the same sample in different datasets.

Data preprocessing

As from figure 3 below, a sample can be viewed and used in training as an image. Dataset 1 and Dataset 1b are much alike but Dataset 2, which is used in neural network models, seems quite different. Horizontal axis represents the time and vertical the ion (m/z). Pixel value is the intensity of an ion at a given timestep.

Note that at the bottom there is always a dark stripe of zero values when there are no ions of such m/z present in the sample (first row always zero too).

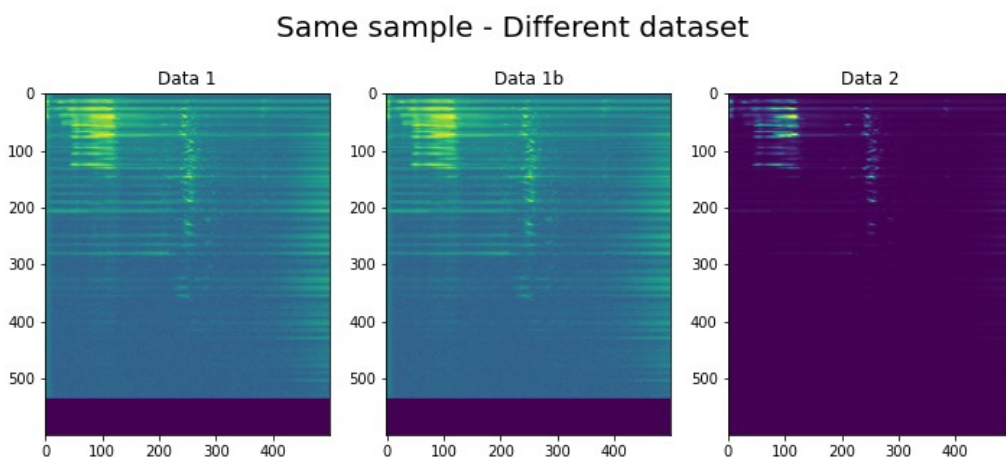


Figure 3. Spectrograms of the same sample for all 3 different initial datasets.

All other models, except CNN, are trained with statistical features of every sample, derived from processed datasets. A code example of this new derived dataset can be seen in figure 4 below.

```
# make a statistical features dataset for logistic regression model
data_maxes2=np.concatenate((
    np.mean(train_img_data1,-1).astype('float')/255-0.5,
    np.std(train_img_data1,-1).astype('float')/255-0.5,
    sp.skew(train_img_data1,-1).astype('float')/255-0.5,
    np.mean(train_img_data1,-2).astype('float')/255-0.5,
    np.expand_dims(der,-1)
),1)
```

Figure 4. Statistical Features B from processed data 1.

Statistical features A dataset comes from Dataset 2, by taking the mean, max and std (standard deviation) features timewise and mean and std ionwise for every sample all scaled to [0,1] interval. Finally, derivative feature (0 or 1) is added to dataset. This dataset is used in simple keras neural network models.

Statistical features B dataset comes from Dataset 1, by taking the mean, std and skew features timewise and mean ionwise for every sample all scaled to [-0.5,0.5] interval. Finally, derivative feature (0 or 1) is added to dataset. This dataset is used in logistic regression model.

Statistical features C dataset comes from Dataset 1, by taking the mean, max and std features timewise and mean ionwise for every sample all scaled to [0,1] interval. Finally, derivative feature (0 or 1) is added to dataset. This dataset is used in ridge classification 1 models.

Statistical features D dataset comes from Dataset 1b, by taking the mean, max and std features timewise and mean ionwise for every sample all scaled to [0,1] interval. Finally, derivative feature (0 or 1) is added to dataset. This dataset is used in ridge classification 2 models.

Modelling

Models used:

- **Simple keras** over tensorflow neural network
- **Pretrained** on imagenet **CNN (efficientnetB0/1/2 keras** over tensorflow)
- **Logistic Regression** (scikit-learn)
- **Ridge classification** (scikit-learn)

Random forest (scikit-learn) models are also used for feature selection only, for ridge classification models and therefore are not direct part in final ensemble.

All models except logistic regression was trained multiple times. This helped both individual model score stability and performance. Two Ridge classification models was trained, each with a different training dataset. Both models used random forest to reduce number of features, to reduce overfitting (the higher the number of features the higher the overfitting probability, especially when number of samples is small) and to increase performance. For simple keras models, the same model is used and averaged 3 times, with 3 different random seeds. Also for keras CNN models, 3 different backbones and random seeds are used.

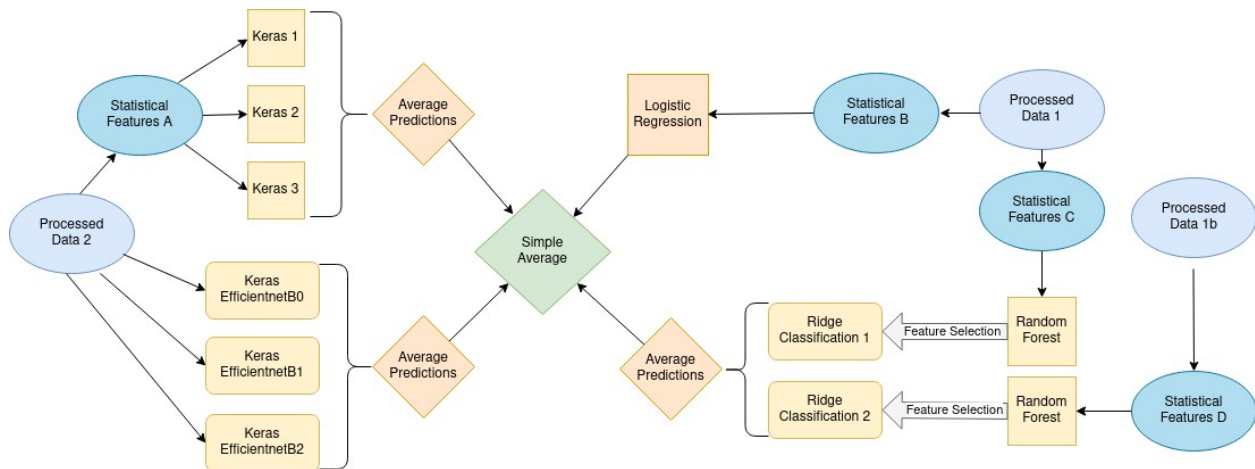


Figure 5. Flowchart of training and inference pipeline.

Model training was done using a 5 fold cross validation stratified split. Logistic regression and ridge classification models are trained multiple times, one for every of 9 classes (1 output at the time) whereas neural network models trained with all 9 classes. For neural networks training the cosine annealing schedule is used with the best weight of every fold saved for inference. Finally, for CNN training, light in-layer augmentation is used consisting of one layer for time shift and one layer for random contrast.

Generalization and Efficiency

In order to see if there is a difference between train and test data distributions, a model was trained to accurately separate train from test data. Results showed that train and test could be separated in some degree, indicating a difference between train and test distributions. This difference was also hinted by the score difference between cross validation and public leaderboard (PubL), which is not all explained by harder to predict classes in PubL data.

Modelling approach was strongly influenced from this difference between train and test dataset distributions, towards a **robust generalized approach**. Ridge classification models are considered in principal as models that tend to **avoid overfitting** as they use L2 regularization. Logistic regression only needed 1 parameter to be tuned (C), which kept the same over all folds and all classes to avoid overfitting. Furthermore the use of pretrained on imagenet models are enhancing our blend's robustness as their starting point was the finishing point of imagenet (had already seen many data). Simple NN may overfit a little more than the other models and that's why it is underweighted in final ensemble (all models are equal weighted despite the fact that CV indicated an increased weight for simple NN equal to 2.5).

Except for generalization, **inference speed** without loosing accuracy in a speed-accuracy trade-off, was always a concern. At inference, 3 out of 4 models in blend are very fast in CPU and even CNN models can be used with CPU quite fast (GPU or other hardware, is not necessary).

Also tried

Thing that tried but didn't make it to final ensemble.

- pseudolabelling,
- sample mixing augmentation,
- feature engineering,
- train NN with one label output at the time,
- other models tried
 - lightgbm,
 - xgboost,
 - tabnet,
 - svm (good score alone but worsens blend),
 - 1 dimensional convolutional neural networks (Conv1D)
 - recurrent neural networks and
 - transformers

Continue working

This is for sure an interesting original dataset. It would be of great benefit of the results if more data was available. Models that didn't work before may work with additional data and/or models that was finally used may not be required any more. Continue working with this data I would:

- Try more pretrain CNN architectures
- Try pretrain CNN with pytorch
- Rethinking all things that didn't work.