

## III. Model documentation and write-up

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

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

I am a senior data scientist [REDACTED] in Athens, Greece. I am a geologist and an oceanographer by education turned to data science through online courses and by taking part in numerous machine learning competitions.

2. What motivated you to compete in this challenge?

I also competed in Mars 1 competition in Spring 2022. Both competitions combine my education in geology, my machine learning profession/skills and my personal interest for space. The fact that this was a NASA competition and results could someday be used for Mars exploration or space in general, was an extra motivation to me.

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

To increase variation **three initial sets of data** was created. Some indicative plots/images are show in figures 1 and 2 below. Each model is trained using a different set of data.

My final solution is an **ensemble of four different type of models: logistic regression, ridge classification and simple neural networks**, all trained with **statistical features** and **pre-trained CNN** trained with **spectrogram** type of data. One of the main difficulties in this competition was the small size of the dataset, with test data distribution being a little different from train one. Not to overfit required more attention and effort than usual. Ridge models are in principal the least overfitting models. Pretrained models also help not to overfit due to their starting point. Logistic regression only need one parameter to tune which is set constant during cross validation for all 9 classes for the same reason. 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, CV indicated an increased weight for simple NN = 2.5)

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

The following plots/images are from our initial constructed datasets. Different processing of raw data give us the capability of capturing a different aspect of view. This increased variation is important in order to increase performance in final model ensemble.

At figure 1, Dataset 1 is constructed with double transformation on ions intensity, using both

square root and log transformation whereas for Dataset 2, square root transform is only used.

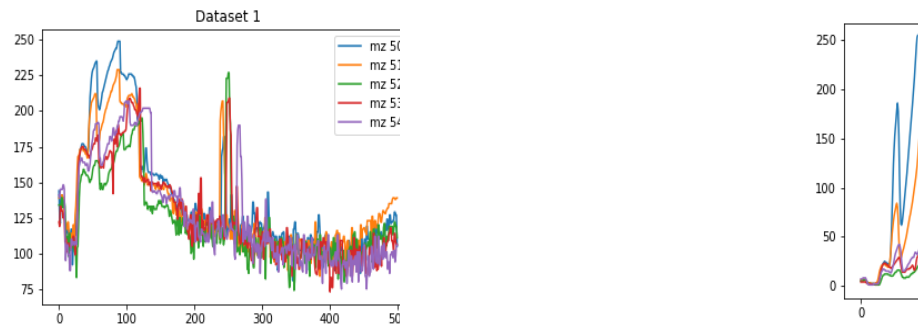


Figure 1. Plots for the same ions of the same sample of different datasets.

Spectrograms as in Data 2 of figure 2 are feed directly to CNN.

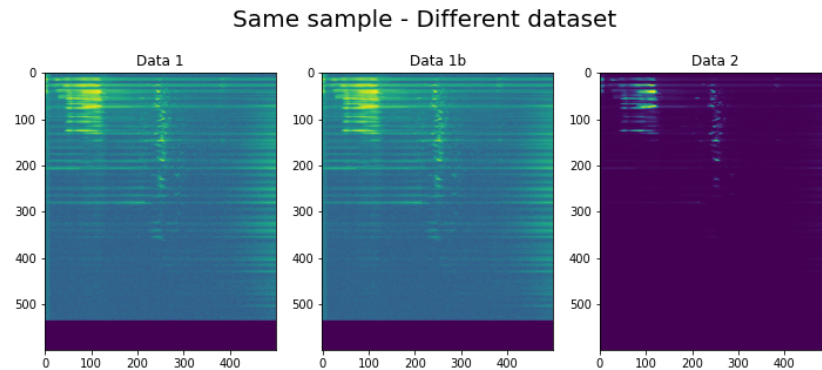


Figure 2. Spectrograms of the same sample for all 3 different initial datasets.

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

One of the most impactful parts is the array2lengthM function which **samples 500 representative timesteps** from every one dimension array of any length. This helped create a dataset of shape (number of samples, 600 ions, 500 timesteps)

```
def array2lengthM(array1D, length=500):
    """
    Turns an array of any length to length equal to length variable.
    If original length > desired length, then select the higher value.
    Args:
        array1D: 1 dimension numpy array
        length: desired output length
    Returns:
        1 dimension numpy array
    """
    t=len(array1D)/length
    ps=[int(x*t) for x in range(length)]
    ps2=ps[1:]+[ps[-1]]

    return np.max(np.stack((array1D[ps],array1D[ps2])),0)
```

From our  
datasets,

constructed  
a new type of

dataset is created with **statistical features** of every sample. For instance the code for logistic regression training data that follows consists of 600 features of mean value, 600 features for standard deviation and 600 for skew for every one of 600 different ions, 500 for mean value for every one of 500 timesteps and 1 feature if sample is derivatized or not (0 or 1), a total of 2301 features for every sample. All but last scaled to [-0.5, 0.5].

```
# 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)
```

Out of many architectures tried for **simple keras NN**, the following performed best. It is a pyramid like with batch normalization layers and swish activation for the dense layers network. Experiments also conducted using this architecture with one output at the time but didn't make it to final ensemble.

```
def get_model_simple(num_outputs=9, dim=(2501,)):
    """
    simple keras model architecture for training on statistical features
    Args:
        num_outputs: number of outputs - classes
        dim: number of training features
    Returns:
        a keras model
    """

    tf.keras.backend.clear_session()
    gc.collect()

    input_layer = Input(shape=(dim), name='input_layer')

    x = Dense(400, activation='swish')(input_layer)
    x = BatchNormalization()(x)

    x = Dense(300, activation='swish')(x)
    x = BatchNormalization()(x)

    x = Dense(200, activation='swish')(x)
    x = BatchNormalization()(x)

    output = Dense(num_outputs, activation='sigmoid')(x)
    model = Model(input_layer, output)
    return model
```

6. Please provide the machine specs and time you used to run your model.
  - CPU (model): logistic regression, ridge classification, simple keras NN
  - GPU (model or N/A): keras CNN (efficientnetB0, efficientnetB1, efficientnetB2)
  - Memory (GB): 30GB
  - OS: Linux (Google Colab Pro)
  - Data preparation duration: for all 3 train-test sets of data, less than 1.5 hours.
  - Train duration: less than 2 hours
  - Inference duration: 40 min preprocess data, 15 min inference in CPU (less in GPU) (for all test data).

7. Anything we should watch out for or be aware of in using your model (e.g. code quirks, memory requirements, numerical stability issues, etc.)?
  - Training CNN model may not be fully reproducible due to augmentation layers.
  - Even for inference, CNN models need to first download pretrained on imagenet weights.
8. Did you use any tools for data preparation or exploratory data analysis that aren't listed in your code submission?

No additional tools for data preparation or exploratory data analysis are used.
9. How did you evaluate performance of the model other than the provided metric, if at all?

Evaluation conducted using competition's metric only. Both training and inference pipelines are a 5-fold cross validation with stratified split.
10. What are some other things you tried that didn't necessarily make it into the final workflow (quick overview)?

Things that tried but didn't make it to final ensemble:

  - other models such as RNNs, transformers, tree models, tabnet and SVC,
  - feature engineering,
  - pseudolabelling
  - sample mixing augmentation
  - train NN with one label output at the time
11. If you were to continue working on this problem for the next year, what methods or techniques might you try in order to build on your work so far? Are there other fields or features you felt would have been very helpful to have?

The difference between train and test distribution is not too large but is capable and easy to overfit on train data. More data for training would be important for this task. Having more data could help make models that failed work and boost our results. New data are not always easy to be collected. Having to keep using the same datasets, I would stop my experiments with pseudolabels and I would work more with all other things that tried and didn't work. Also, if training and inference speed is not an issue I would add **more pretrained CNNs** and probably **add pytorch pretrained CNNs** both for diversity and easier to access to more architectures.