Exoplanetary surface composition prediction using ML

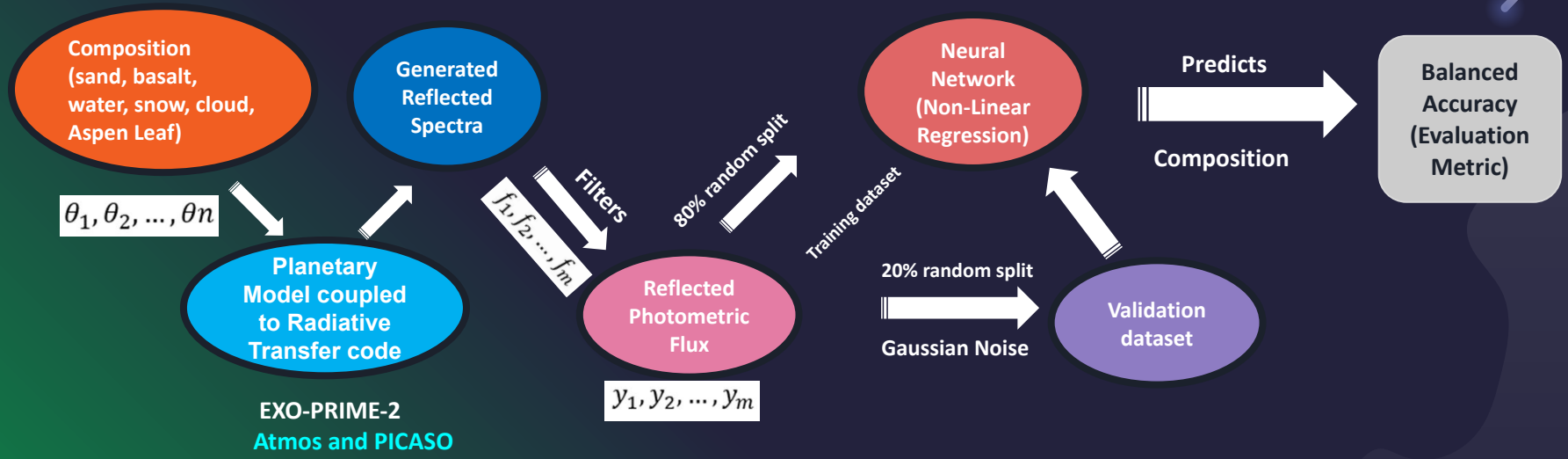
IDEA:

To identify the presence of minerals on the surface of exoplanets (mainly terrestrial) by implementing Machine Learning on the reflection photometric flux from spectra generated using planetary models (Atmos, PICASO) and spectral library (USGS, PSG and MODIS).

This can help characterize future telescopes for predicting composition using photometric flux and follow up in time-intensive spectroscopic data.

TABLE OF CONTENTS

Literature Review and Project Idea:



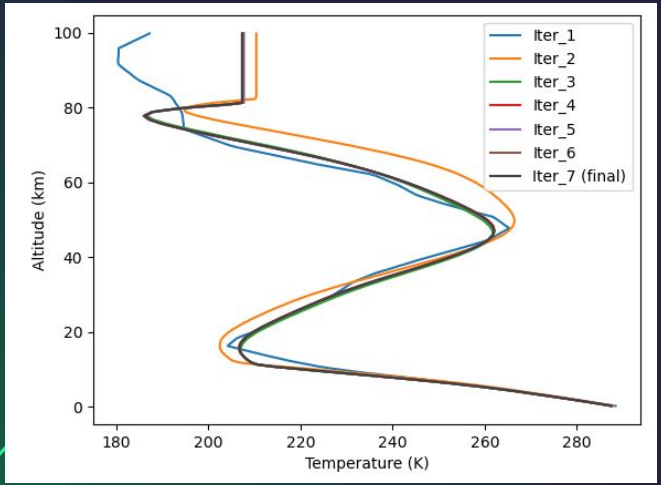
Atmos: A coupled climate-photochemical model

Photochemical Model

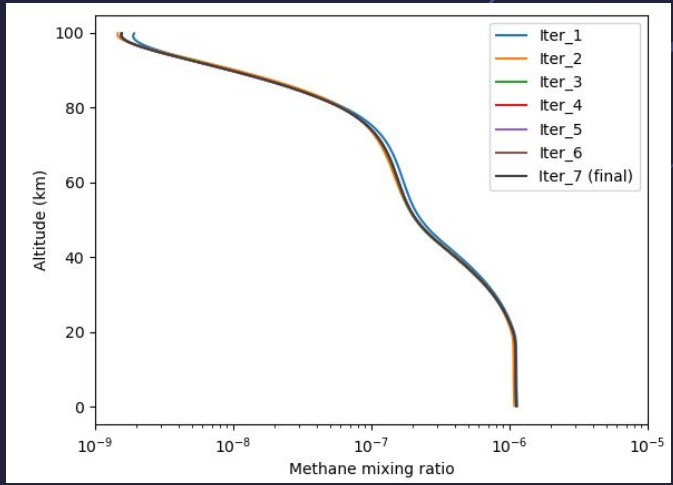
- Generates an initial atmospheric state
- User-specified boundary conditions - gas mixing ratios or fluxes and deposition velocities, the stellar spectrum, the total atmospheric pressure, the initial temperature-pressure profile
- 233 chemical reactions and includes 50 chemical species, 9 of which are short-lived
- Outputs: altitude-dependent abundances of H_2O photochemically produced in, or transported to, the stratosphere, CO_2, O_3, CH_4, O_2, N_2, and C_2H_6.

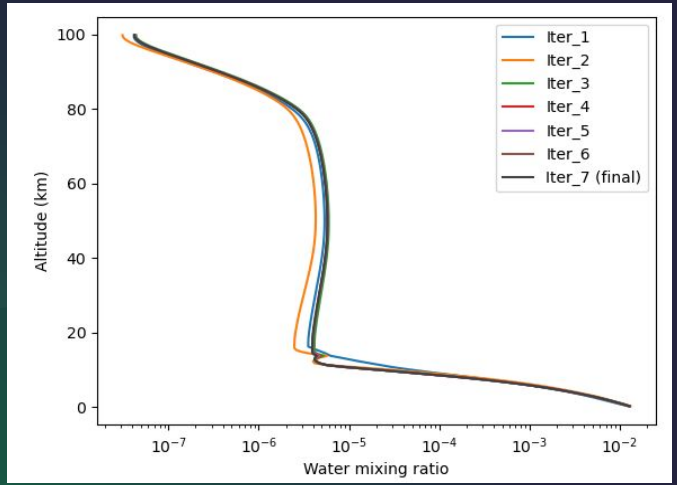
Climate Model

- The tropospheric temperature profile calculated by following a wet adiabatic lapse rate to the altitude at which the stratospheric temperature is reached
- Gases in the upper atmosphere can have a heating or cooling effect on the temperature profile depending on the relative abundance of gases in the upper atmosphere and the extent of shortwave heating.
- Input parameters - the number of steps to run the model, pressure at the surface, pressure at the top of the atmosphere, surface temperature, surface albedo, solar constant and surface gravity.
- Outputs : altitude, temperature, water mixing ratio



Plots showing Atmos finding Convergence for Modern Earth conditions





PICASO : A Radiative Transfer Model

- PICASO is an atmospheric radiative transfer model that we are using to produce the reflection spectra. The original documentation was used as reference for using PICASO in our codes.
- PICASO can be used for obtaining transmission, emission and reflection spectra.
- In this project we use the PICASO for obtaining the reflection spectra of exoplanet with a certain wavelength dependent albedo function for its surface.

PICASO takes the following inputs

- Basic planetary properties(like planet mass, radius, stellar spectra)
- PT profile and altitude abundances(This is obtained from ATMOS)
- Cloud profile (angle-scattering albedo, asymmetry and total extinction)(not used in our case instead albedo of the cloud is obtained from a model and is used along with other albedo functions for finding effective albedo)
- Surface albedo: can be average surface albedo or wavelength dependent surface albedo which is used in this case.
- The main equation used in PICASO is the radiative transfer equation given below,

$$I\left(\tau_i,\mu\right)=I\left(\tau_{i+1},\mu\right)e^{\delta\tau_i/\mu}-\int_0^{\delta\tau_i}S\left(\tau',\mu\right)e^{-\tau'/\mu}d\tau'/\mu$$

Data- Set Description

- The data set provided in the paper which we will be trying to reproduce with PICASO and ATMOS consists of reflection spectra for various surface combinations.
- Filters are applied on this spectra to obtain the photometric flux. In our case we took nine photometric flux values which was found to provide us with a good
- Six surfaces were considered which are Snow, sand, basalt, cloud, vegetation and sea water
- Permutations with 5 percent steps for these six surfaces lead to a total of 53,130 different surfaces.
- The data is then divided into training and validation data with an 80-20 ratio.

Surface combinations

	cloud	snow	sand	seawater	basalt	veg
0	0.00	0.00	0.00	0.00	0.00	1.00
1	0.00	0.00	0.00	0.00	0.05	0.95
2	0.00	0.00	0.00	0.00	0.10	0.90
3	0.00	0.00	0.00	0.00	0.15	0.85
4	0.00	0.00	0.00	0.00	0.20	0.80
...	...	...	...	...	...	...
53125	0.95	0.00	0.00	0.00	0.05	0.00
53126	0.95	0.00	0.00	0.05	0.00	0.00
53127	0.95	0.00	0.05	0.00	0.00	0.00
53128	0.95	0.05	0.00	0.00	0.00	0.00
53129	1.00	0.00	0.00	0.00	0.00	0.00

Combinations of surfaces

	f1	f2	f3	f4	f5	f6	f7	f8	f9
0	66.450295	70.559679	60.551194	34.801215	13.230524	8.823439	6.337946	0.350842	1.599316
1	67.564830	69.209650	58.643643	33.787692	12.898762	8.762820	6.250520	0.381013	1.635478
2	68.679366	67.859620	56.736091	32.774169	12.567000	8.702201	6.163094	0.411184	1.671641
3	69.793902	66.509591	54.828539	31.760645	12.235237	8.641582	6.075669	0.441354	1.707803
4	70.908438	65.159561	52.920988	30.747122	11.903475	8.580963	5.988243	0.471525	1.743966
...	...	...	...	...	...	...	...	...	...
53125	234.117912	142.502061	90.005522	58.435347	33.283890	21.958330	16.991558	4.328158	5.985801
53126	232.629441	141.027892	89.131071	57.823231	33.000671	21.622128	16.787464	4.287007	5.882511
53127	238.863237	146.362171	92.777791	60.235069	34.058652	22.909336	17.573305	4.480611	6.353865
53128	245.008517	149.283036	93.169635	59.558788	33.361182	21.608670	16.806243	4.282940	5.877620
53129	241.769328	147.709586	93.563699	60.746116	34.688553	22.713449	17.644302	4.505732	6.178602

Implementing ML Algorithms

Creating random forest model and saving it¶

In [ ]:

```
# Load the training data from pickle file
with open("train_data.pkl", "rb") as f:
    train_data = pickle.load(f)

# Load the test data from pickle file
with open("test_data.pkl", "rb") as f:
    test_data = pickle.load(f)

# Separate the features and labels in the training data
train_X = [sample[0] for sample in train_data]
train_y = [sample[1] for sample in train_data]

# Separate the features and labels in the test data
test_X = [sample[0] for sample in test_data]
test_y = [sample[1] for sample in test_data]

# Train the random forest regression model
rfr = RandomForestRegressor(n_estimators=10)
rfr.fit(train_X, train_y)

# Save the trained model to a file
dump(rfr, "rfr.pkl")
```

In [ ]:

```
# Load the saved model from file
load_model = load("rfr.pkl")

# Use the trained model to predict the labels for the test data
pred_y = load_model.predict(test_X)

# Calculate the mean squared error between the predicted and actual labels
mse = mean_squared_error(test_y, pred_y)

print("Mean squared error:", mse)
print(pred_y[2])
print(test_y[2])
```

Random Forest

Mean square error with predicted output.

```
print("Mean squared error:", mse)
print(pred_y[1])
print(test_y[1])
```

```
Mean squared error: 0.002353999929418408
[0.041  0.307  0.3235 0.1375 0.1475 0.0435]
[0.05 0.3  0.35 0.25 0.   0.05]
```

Support Vector Regression

Creating the svr model and training it¶

In [ ]:

```
# Load the training data from pickle file
with open("train_data.pkl", "rb") as f:
    train_data = pickle.load(f)

# Load the test data from pickle file
with open("test_data.pkl", "rb") as f:
    test_data = pickle.load(f)

# Separate the features and labels in the training data
train_X = [sample[0] for sample in train_data]
train_y = [sample[1] for sample in train_data]

# Separate the features and labels in the test data
test_X = [sample[0] for sample in test_data]
test_y = [sample[1] for sample in test_data]

# Create a support vector regression model
svr = SVR(kernel='linear')

# Create a multi-output regression model with the SVM model as the base estimator
model = MultiOutputRegressor(svr)

# Fit the model to the training data
model.fit(train_X, train_y)

# Save the trained model to a file
dump(model, "svr.pkl")
```

Out[ ]:

```
['svr.pkl']
```

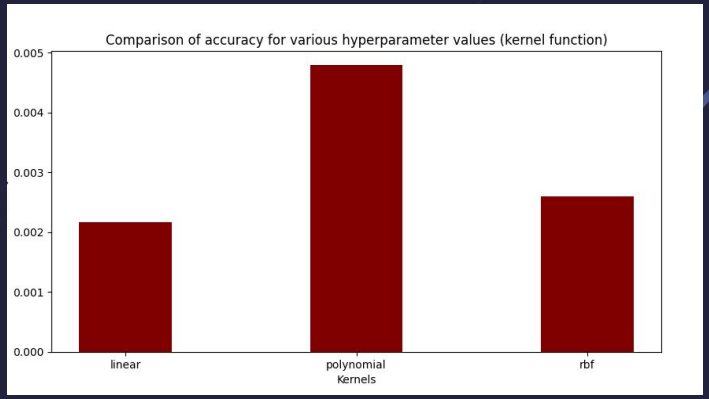
In [ ]:

```
# Load the test data from pickle file
with open("test_data.pkl", "rb") as f:
    test_data = pickle.load(f)

# Separate the features and labels in the test data
test_X = [sample[0] for sample in test_data]
test_y = [sample[1] for sample in test_data]

# Load the saved model from file
load_model = load("svr.pkl")

# Use the trained model to predict the labels for the test data
pred_y = load_model.predict(test_X)

# Calculate the mean squared error between the predicted and actual labels
mse = mean_squared_error(test_y, pred_y)

print("Mean squared error:", mse)
print(pred_y[2])
print(test_y[2])
```

```
Mean squared error: 0.002024739050471741
[0.03137104 0.42006451 0.23960476 0.11034186 0.09022923 0.12851113]
[0.   0.45 0.2  0.1  0.1  0.15]
```

Statistics

- The Hyperparameter for Support Vector Regressor (SVR) is the kernel function.
- Linear kernel function performs best for this problem
- Random Forest Regressor (RFR) has the number of trees as hyperparameter.
- Here we have used 100 estimators (trees).



FUTURE PLANS

Obtain Albedo for Cloud and

Comparing spectra from paper with one produced by our method

Noise Augmentation, Training and Validation using SVR, RF and MLP

Surface Composition Prediction