Modeling Stellar Spectra

Enmanuel Hernandez

December 20, 2022

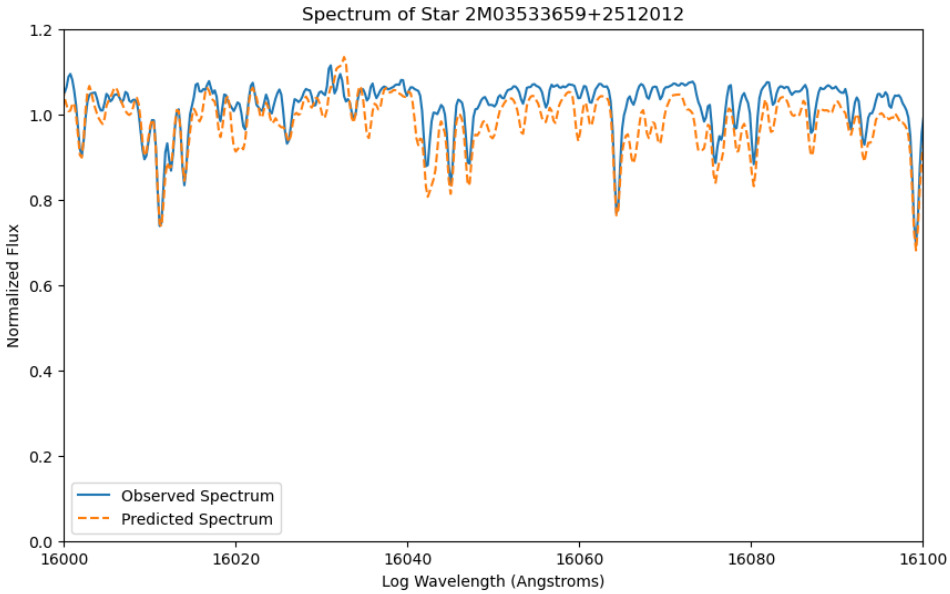

Figure 1: Sample of Stellar Spectrum Prediction

1 Introduction

Our goal, in this lab is to create a model that can anticipate how stellar spectra will look based on properties of stars, such as temperature, surface gravity and the amounts of elements present. These factors. Like temperature (Teff) surface gravity (logg) and the levels of iron (Fe) magnesium (Mg) and silicon (Si). Play a role in our understanding of star characteristics and evolution. We will specifically focus on studying stars using data gathered from the APOGEE survey within the Sloan Digital Sky Survey (SDSS).

The model building process relies heavily on data starting with a set of labeled spectra for training purposes to learn how each label impacts the spectrum. This methodology is described in detail by Ness et al. 2015 which acts as a reference for our work. Additionally we refer to sources such as Majewski et al. 2017 and Ahumada et al. 2020 to gain insights into the APOGEE survey along with Holtzman et al. 2015 for information on the ASPCAP pipeline used by APOGEE to determine properties, from spectra. The lab comprises elements, such, as standardizing data identifying outliers using linear and data derived models applying cross validation optimizing non linearly employing MCMC methods and comprehending how stars evolve. By utilizing these methods the objective is to create a model capable of precisely forecasting star properties based on their spectra.

2 Methods

2.1 Data Acquisition: Downloading APOGEE Spectra

To access APOGEE spectra files we start by specifying the fields we're interested, in and the main URL for the APOGEE spectra repository. Next we create a folder to save the files. For every field we generate the URL to reach the directory containing the files and utilize the 'requests library to send a GET request to this URL. The response is then parsed using 'BeautifulSoup' to extract the names of the files, which are recognized by their prefix 'apStar r'. We then create a download URL for each file verify if it already exists in our download location and if not proceed with downloading and saving it. This method allows us to efficiently acquire and store APOGEE files for our chosen fields making them ready, for examination.

Listing 1: Utilities for Apogee Spectra Extraction

```
# Fields and the base URL for the APOGEE spectra
fields = ["M15", "N6791", "K2_C4_168-21", "060+00"]
base_url = "https://data.sdss.org/sas/dr16/apogee/spectro/redux/r12/stars/apo25m/"

# Directory to store the downloaded files
download_dir = "apogee_spectra"
os.makedirs(download_dir, exist_ok=True)

# Iteration over the fields and download the spectra
for field in fields:
    field_url = base_url + field + "/"
    response = requests.get(field_url)

    if response.status_code == 200:
        soup = BeautifulSoup(response.text, 'html.parser')
        file_names = [a['href'] for a in soup.find_all('a') if a['href'].startswith('a

        # Downloading each file
        for file_name in file_names:
            file_name = file_name.replace('%2B', '+')
            file_url = field_url + file_name
            file_path = os.path.join(download_dir, file_name)

            # Check if the file already exists
            if not os.path.exists(file_path):
                r = requests.get(file_url)

                if r.status_code == 200:
                    with open(file_path, 'wb') as f:
                        f.write(r.content)

save_dir = "apogee_spectra"
```

To build the wavelength array, for all APOGEE spectra in our data files we begin by accessing the FITS file that holds the combined spectrum using the 'function from the 'astropy.io.fits' module. We retrieve the flux values, from the extension of this FITS file. The header of this extension provides details needed to create the wavelength array:

- 'CRVAL1': the starting wavelength in Angstroms, - 'CDELT1': the logarithmic wavelength step, and - 'NAXIS1': the number of wavelength pixels.

Constructing the wavelength array involves utilizing the provided header values. We begin at the specified 'start wavelength'. Then increment, by the designated 'log wavelength step', for a total of 'num pixels iterations. This relationship can be mathematically represented as follows:

$$\text{wavelength} = \text{start\_wavelength} \times 10^{\text{log\_wavelength\_step} \times \text{np.arange(num\_pixels)}}$$

This eq calculates the wavelength for each single pixel in the spectra, producing an array of wavelengths corresponding to the flux values in the coadded spectrum.

2.2 Data Preparation: Extracting Stellar Labels from the allStar Catalog

After importing the 'allStar' catalog, we applied several cuts to ensure data quality and focus on giant stars: We discarded spectra with SNR less than 50, surface gravity ('LOGG') greater than 4, effective temperature ('TEFF') greater than 5700 K, and metallicity ([Fe/H]) less than -1. Additionally, we ensured that the labels 'TEFF', 'LOGG', 'FE H', 'MG FE', and 'SI FE' are not NaN (Not a Number). These criteria results in the 'filtered allstar' catalog, containing a dataset of giant stars usable for building our training set.

Listing 2: Utilities for Apogee Spectra Extraction

```
filtered_allstar = matched_catalog[(matched_catalog['SNR'] >= 50) &
                                   (matched_catalog['LOGG'] < 4) &
                                   (matched_catalog['TEFF'] < 5700) &
                                   (matched_catalog['FE_H'] > -1) &
                                   (~np.isnan(matched_catalog['TEFF'])) &
                                   (~np.isnan(matched_catalog['LOGG'])) &
                                   (~np.isnan(matched_catalog['FE_H'])) &
                                   (~np.isnan(matched_catalog['MG_FE'])) &
                                   (~np.isnan(matched_catalog['SI_FE']))]
```

2.3 Data Cleaning: Identifying and Handling Bad Pixels

To spot faulty pixels, in each range we rely on a bitmask and a set of indicators linked to problems like cosmic ray impacts or sky subtraction glitches. These problematic pixels are then marked off. Their inaccuracies are adjusted to a value to prevent them from playing a major role, in the fitting process likelihood calculation.

Listing 3: Utilities for Bad Pixel indentifying in each spectrum

```
def create_bad_pixel_mask(bitmask, flags):
    #Creates a mask for bad pixels based on the given flags
    mask = np.zeros_like(bitmask, dtype=bool)
    for flag in flags:
        mask |= (bitmask & (1 << flag)) != 0
    return mask

def set_uncertainty_for_bad_pixels(flux, flux_err, bitmask, flags, large_value=1e5):
    # Set the uncertainty for bad pixels to a large value
    # Ensures flux is an array
    flux = np.atleast_1d(flux)

    # Converting scalar flux_err to a NumPy array
    if np.isscalar(flux_err):
        flux_err = np.full_like(flux, flux_err)

    bad_pixel_mask = create_bad_pixel_mask(bitmask, flags)
    flux_err[bad_pixel_mask] = large_value
    return flux, flux_err
```

2.4 Data Normalization: Implementing Pseudo-Continuum Normalization

Next I utilized the 'pseudo continuum normalize' function to standardize the flux and error values for every star before starting the modeling process. This function calculates the continuum by fitting a Chebyshev polynomial to wavelengths of the continuum then scales the spectrum and error

arrays by dividing them with the calculated continuum. It references the continuum wavelengths from 'continuum pixels apogee.npz' which are imported earlier in the process.

Listing 4: Utility for Normalizing Spectrum

```
def pseudo_continuum_normalize(spectrum, error, wavelength, continuum_wavelengths):
    # flux values at the continuum wavelengths
    continuum_flux = spectrum[np.isin(wavelength, continuum_wavelengths)]

    # Chebyshev polynomial to the continuum points
    coeffs = Chebyshev.fit(wavelength[np.isin(wavelength, continuum_wavelengths)], cor

    # Calculating the continuum for the full wavelength range
    continuum = coeffs(wavelength)

    # Normalizing the spectrum and error by the estimated continuum
    normalized_spectrum = spectrum / continuum
    normalized_error = error / continuum

    return normalized_spectrum, normalized_error, continuum
```

2.5 Model Training: Building and Validating the Spectral Model

To begin the modeling validation process we first mix up the APOGEE IDs and their corresponding spectra in the 'spectra tables dictionary. Afterward we divide these shuffled IDs into two groups; one, for training and the other for cross validation. By matching the split keys with the spectra in the 'spectra tables' we create datasets for training and cross validation.

Then we convert these keys into lists of APOGEE IDs for both sets. Subsequently we refine the 'filtered allstar' table, which holds star labels to generate tables for training and cross validation labels. These refined tables only contain labels linked to APOGEE IDs found in their datasets.

Lastly we narrow down the 'filtered allstar' table to include those APOGEE IDs, in the 'spectra tables dictionary. Using this subset helps us construct training and cross validation label tables that exclusively encompass stars represented in our data.

2.6 Model Optimization: Fitting the Training Set Spectra (B)

The function called 'create design matrix' builds the design matrix (X), for our model. In this matrix each row represents a star in the training set and the columns signify the terms in our model, which include the term, linear terms and quadratic terms with cross products.

On the hand the function 'fit pixel' aims to determine the model parameters (theta) and intrinsic scatter (s squared) for a specific wavelength pixel. It considers the flux and error values for that pixel across all stars in the training set along with their labels. By performing a grid search on values of s squared it seeks to maximize the log likelihood function that measures how well the model aligns with data for a given s value. The optimal s squared is identified as the one that maximizes this log likelihood.

After finding the s value this function calculates the optimal theta values—these are parameters of our model that minimize differences, between observed flux and predicted flux by considering intrinsic scatter. These optimized parameters are then utilized to predict flux at a given wavelength pixel based on any set of labels. This process iterates through all wavelength pixels to derive all parameters using these defined methodologies.

Listing 5: Design Matrix and Functions to Attain Optimal Parameters

```
def create_design_matrix(labels):
    # design matrix X for a 2nd-order polynomial model
    X = np.column_stack([
        np.ones(len(labels)),
        labels['TEFF'],
        labels['LOGG'],
        labels['FE_H'],
```

```
        labels['MG_FE'],
        labels['SI_FE'],
        labels['TEFF'] ** 2,
        labels['TEFF'] * labels['LOGG'],
        labels['TEFF'] * labels['FE_H'],
        labels['TEFF'] * labels['MG_FE'],
        labels['TEFF'] * labels['SI_FE'],
        labels['LOGG'] ** 2,
        labels['LOGG'] * labels['FE_H'],
        labels['LOGG'] * labels['MG_FE'],
        labels['LOGG'] * labels['SI_FE'],
        labels['FE_H'] ** 2,
        labels['FE_H'] * labels['MG_FE'],
        labels['FE_H'] * labels['SI_FE'],
        labels['MG_FE'] ** 2,
        labels['MG_FE'] * labels['SI_FE'],
        labels['SI_FE'] ** 2
    ])
    return X

def fit_pixel(flux, error, labels, s_squared_grid):
    # design matrix X
    X = create_design_matrix(labels)

    # Function to compute the negative log-likelihood for a given s_squared
    def neg_log_likelihood(s_squared):
        # model parameters theta that minimize the squared residuals
        theta = np.linalg.lstsq(X.T @ X + s_squared * np.eye(X.shape[1]), X.T @ flux,
        # residuals
        residuals = flux - X @ theta
        # log-likelihood
        log_likelihood = -0.5 * np.sum((residuals / np.sqrt(error ** 2 + s_squared))
        return -log_likelihood

    # value of s_squared that maximizes the log-likelihood
    optimal_s_squared = minimize(neg_log_likelihood, x0=[0.01], bounds=[(0, None)]).x

    # optimal theta using the optimal s_squared
    optimal_theta = np.linalg.lstsq(X.T @ X + optimal_s_squared * np.eye(X.shape[1]),

    return optimal_theta, optimal_s_squared
```

2.7 Model Optimization: Fitting the Training Set Spectra (D)

The 'predict spectrum' function is then created to utilize the optimized model parameters for forecasting the normalized brightness of a star, at every wavelength point, based on its characteristics. It forms a label vector for the model consisting of linear and quadratic terms with interactions. Subsequently the function computes the spectrum by multiplying the parameters for each wavelength point, with the label vector.

Listing 6: Model Prediction Function

```
def predict_spectrum(labels, optimal_parameters, wavelength_pixels):
    # labels to a NumPy array
    labels_array = np.array([
        labels['TEFF'], labels['LOGG'], labels['FE_H'], labels['MG_FE'], labels['SI_FE
    ])

    # label vector for the polynomial model
```

```
    label_vector = np.array([1] + list(labels_array) + [labels_array[i] * labels_array

    # normalized flux for each wavelength pixel
    predicted_spectrum = np.array([np.dot(optimal_parameters[i], label_vector) for i i

    return predicted_spectrum
```

2.8 Gradient Analysis: Assessing Sensitivity to Stellar Labels

The strong lines for Si

$$[15361.161, 15376.831, 15833.602, 15960.063, 16060.009, 16094.787, 16215.670, 16680.770, 16828.159]$$

and Mg

$$[15740.716, 15748.9, 15765.8, 15879.5, 15886.2, 15954.477]$$

were chosen based on established data the Apogee GitHub repository

$$(https://github.com/jobovy/apogee/blob/main/apogee/spec/plot.py)$$

, which provides a list of known absorption lines in the APOGEE spectra. These lines are recognized for their sensitivity to the respective elemental abundances.

2.9 Cross-Validation: Comparing Best-Fit Labels with ASPCAP Values

The prediction function based on labels computes the estimated flux using a degree model, with linear and interaction components included.

The fitting function acts as a cover for optimization employing curve fitting to determine the labels for each star by reducing the gap, between observed and estimated flux while taking into account uncertainties in the observed data. Setting boundaries for the labels helps maintain optimization within specified limits.

Listing 7: Functions to Attain Best Fit Labels

```
#function to make predictions from labels
def prediction_from_labels(labels):
    products = np.outer(labels, labels)
    triu_indices = np.triu_indices(5)
    unique_products = products[triu_indices]

    label_vector = np.hstack((np.array([1]), labels, unique_products))
    preds = np.array(optimal_parameters) @ label_vector
    return preds

# Fitting function
def fit_func(x, *labels):
    return prediction_from_labels(labels)
```

2.10 Error Analysis: Investigating Discrepancies in Cross-Validation Results

With this function our goal is to sift through the catalog to weed out data by excluding items, with STARFLAG and ASPCAPFLAG values that are not zero as these could signal data anomalies. It then creates samples using the uncertainties associated with the labels to evaluate the accuracy and variability in each label offering an assessment of how trustworthy and precise the data is. This step is essential, for gauging the credibility of the labels and guaranteeing that only dependable data is employed for analysis and interpretation purposes.

Listing 8: Function to Remove unreliable data from allstar catalog

```
def remove_unreliable_labels(filtered_allstar):
```

```
    # objects that do not have flags indicating unreliable data:
    reliable_objects = filtered_allstar[(filtered_allstar['STARFLAG'] == 0) & (filtere

    # relevant parameters and errors
    teff = reliable_objects['TEFF'].data
    logg = reliable_objects['LOGG'].data
    fe_h = reliable_objects['FE_H'].data
    teff_err = reliable_objects['TEFF_ERR'].data
    logg_err = reliable_objects['LOGG_ERR'].data
    fe_h_err = reliable_objects['FE_H_ERR'].data

    # Generating random samples considering the uncertainty in each parameter
    teff_random_samples = np.random.normal(teff, teff_err)
    logg_random_samples = np.random.normal(logg, logg_err)
    fe_h_random_samples = np.random.normal(fe_h, fe_h_err)

    # Calculating bias and scatter for each label
    teff_bias = np.mean(teff_random_samples - teff)
    teff_scatter = np.std(teff_random_samples - teff)

    logg_bias = np.mean(logg_random_samples - logg)
    logg_scatter = np.std(logg_random_samples - logg)

    fe_h_bias = np.mean(fe_h_random_samples - fe_h)
    fe_h_scatter = np.std(fe_h_random_samples - fe_h)

    # bias and scatter
    results = {
        'teff': {'bias': teff_bias, 'scatter': teff_scatter},
        'logg': {'bias': logg_bias, 'scatter': logg_scatter},
        'fe_h': {'bias': fe_h_bias, 'scatter': fe_h_scatter}
    }

    return results
```

2.11 Bayesian Inference: Fitting Mystery Spectrum with MCMC

The code below uses MCMC to analyze the spectrum with a model, in PyMC. The log likelihood function evaluates how well the data aligns with the model parameters (labels) which is then encapsulated in a custom PyMC Op class (LogLike) to integrate this external likelihood function into the PyMC framework. The model sets uniform priors for the parameters based on data and leverages the MCMC algorithm to draw samples from the posterior distribution offering insights into the parameters that best match the mystery spectrum. This methodology mirrors an illustration from the PyMC documentation (

($https://www.pymc.io/projects/examples/en/latest/howto/blackbox_external_likelihood_numpy.html$)

) showcasing how to include a black box likelihood function, within a PyMC model.

Listing 9: MCMC model for Predictions

```
def log_likelihood(labels, x, data, sigma):
    prediction = prediction_from_labels(labels)
    return -0.5 * np.sum((data - prediction) ** 2 / (sigma ** 2) + np.log(2 * np.pi *

class LogLike(pt.Op):
    itypes = [pt.dvector]
    otypes = [pt.dscalar]
```

```
    def __init__(self, loglike, data, x, sigma):
        self.likelihood = loglike
        self.data = data
        self.x = x
        self.sigma = sigma

    def perform(self, node, inputs, outputs):
        theta = inputs[0]
        logl = self.likelihood(theta, self.x, self.data, self.sigma)
        outputs[0][0] = np.array(logl)

# mystery_norm_flux and mystery_norm_uncertainty are values from mystery_spectrum
sample = np.linspace(0, 100, 100)
logl = LogLike(log_likelihood, normalized_mystery_spectrum, sample, spectrum_uncertain

with pm.Model() as model:
    teff = pm.Uniform('teff', lower=3000, upper=6000)
    logg = pm.Uniform('logg', lower=-1, upper=4.5)
    fe_h = pm.Uniform('fe_h', lower=-3, upper=1)
    mg_fe = pm.Uniform('mg_fe', lower=-1, upper=1)
    si_fe = pm.Uniform('si_fe', lower=-1, upper=1)

    theta = pt.as_tensor([teff, logg, fe_h, mg_fe, si_fe])
    pm.Potential('likelihood', logl(theta))

    trace = pm.sample(8000, tune=1000)
```

2.12 Red Giant Branch Evolution: Tracing Changes in the RGB Spectrum

The selected criteria, for tracking the development of stars on the Red Giant Branch include the surface gravity (logg) values ranging from 3.5 to 0.5 which're typical for stars progressing along the RGB. Additionally the effective temperature (Teff) values range from 5000 K to 4500 K mirroring the anticipated temperature decrease as a star expands and moves, along the RGB. These parameters were chosen based on isochrone paths observed in the Kiel diagram during the RGB phase.

3 Results

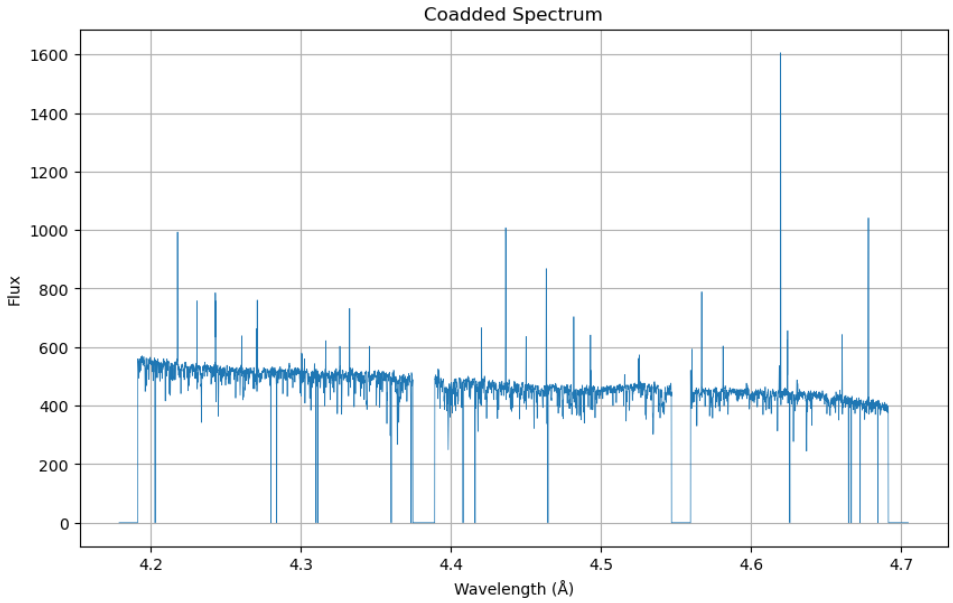

Figure 2: The chart above shows the combined spectrum of a star chosen at random with the vertical axis indicating the flux intensity, in units and the horizontal axis indicating the wavelength in Angstroms. The multiple peaks in the graph indicate absorption lines indicating wavelengths of light that are absorbed by elements, in the stars atmosphere.

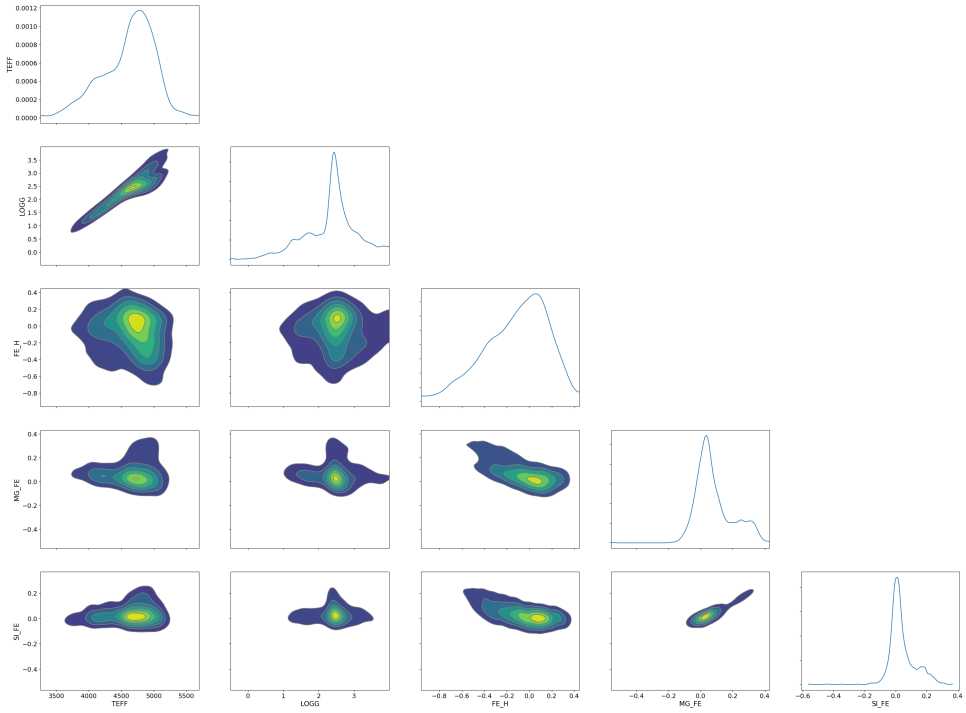

Figure 3: this visualizes the distribution of stellar properties for 1855 stars in our training set. The plot provides insights into the multidimensional relationships between these labels

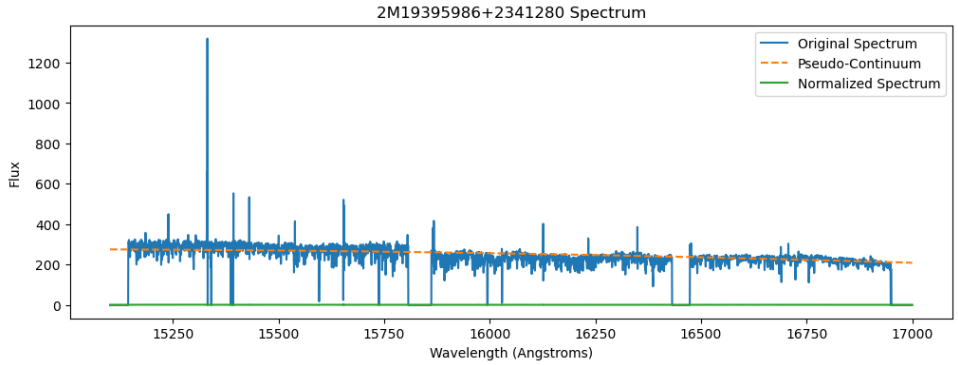

Figure 4: This displays the original spectrum, the estimated pseudo-continuum , and the resulting normalized spectrum for the star 2M19395986+2341280. It demonstrates the effectiveness of the pseudo-continuum normalization process over the entire wavelength range.

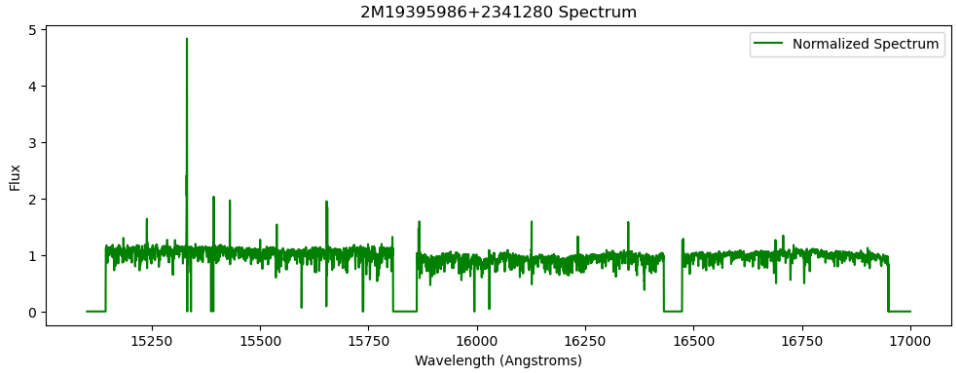

Figure 5: A closer look on the normalized flux

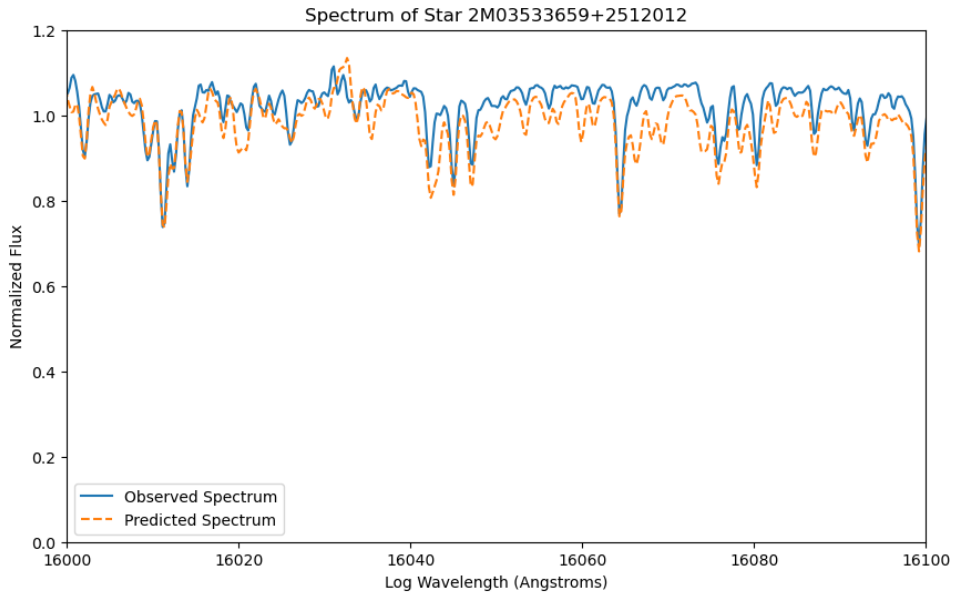

Figure 6: The comparison above shows how closely the observed spectrum matches the predicted spectrum, for Star 2M03533659+2512012. Both spectra have been. Plotted on a log wavelength scale in Angstroms. The strong alignment between the two lines, over the range of wavelengths displayed confirms the reliability of the model employed for prediction.

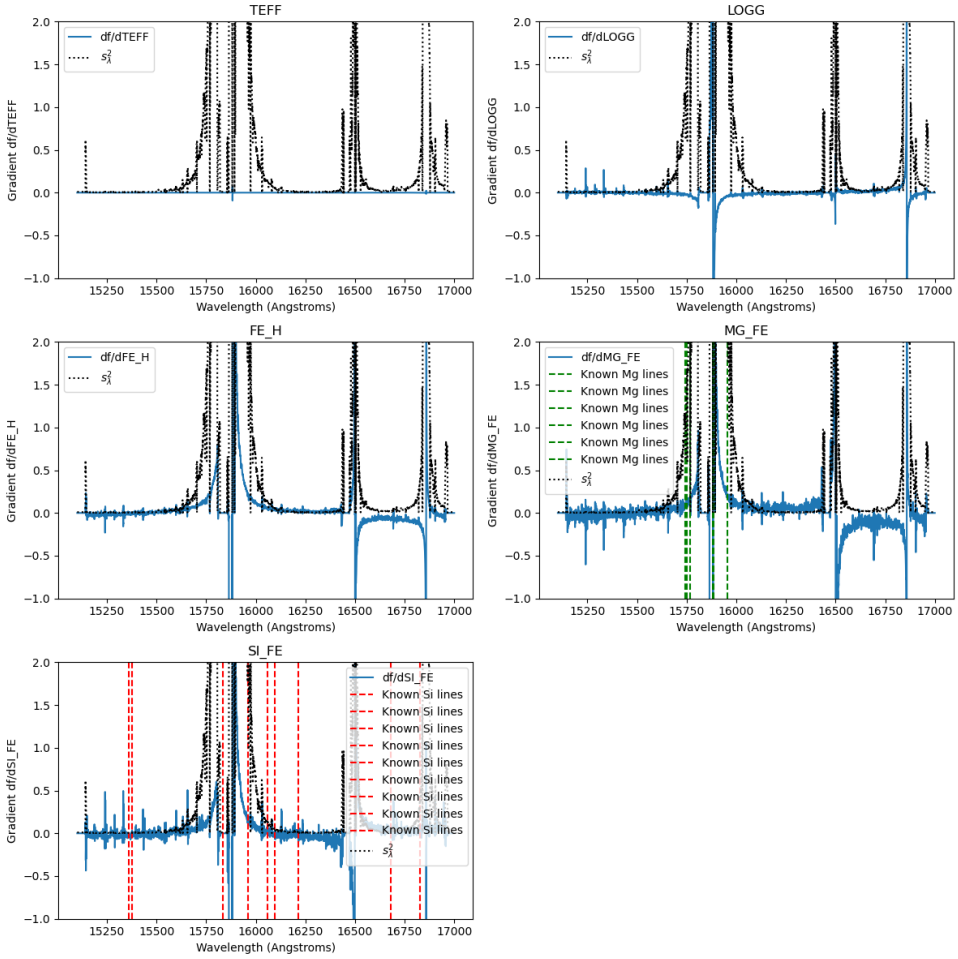

Figure 7: This illustrates the gradient spectra (df/d*lambda) for each of the five labels (Teff, logg, [Fe/H], [Mg/Fe], [Si/Fe]), which helps identify the wavelengths most sensitive to changes in each label. It also includes the intrinsic scatter term (s*lambda^2) and known strong absorption lines for Si

$$[15361.161, 15376.831, 15833.602, 15960.063, 16060.009, 16094.787, 16215.670, 16680.770, 16828.159]$$

and Mg

$$[15740.716, 15748.9, 15765.8, 15879.5, 15886.2, 15954.477]$$

based from of the Apogee GitHub repository

$$(https://github.com/jobovy/apogee/blob/main/apogee/spec/plot.py)$$

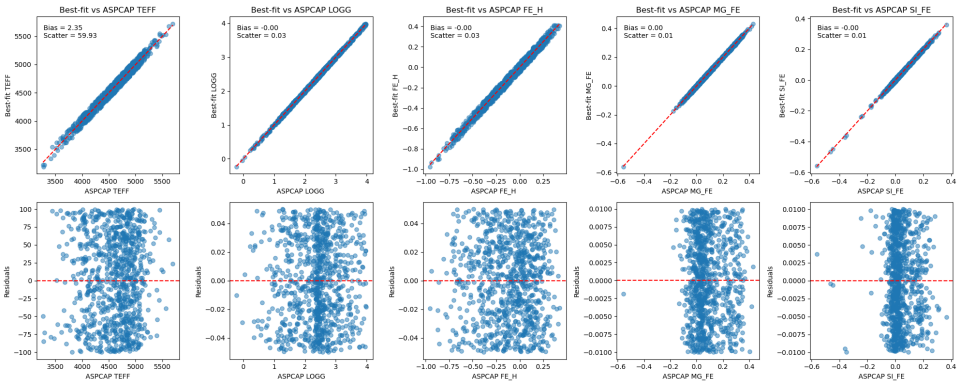

Figure 8: The plot above compares best-fit stellar labels derived from spectral fitting against labels provided by ASPCAP. The tight clustering around the one-to-one line, along with low bias and scatter after cuts for outliers, suggests a agreement between the modeled and observed value

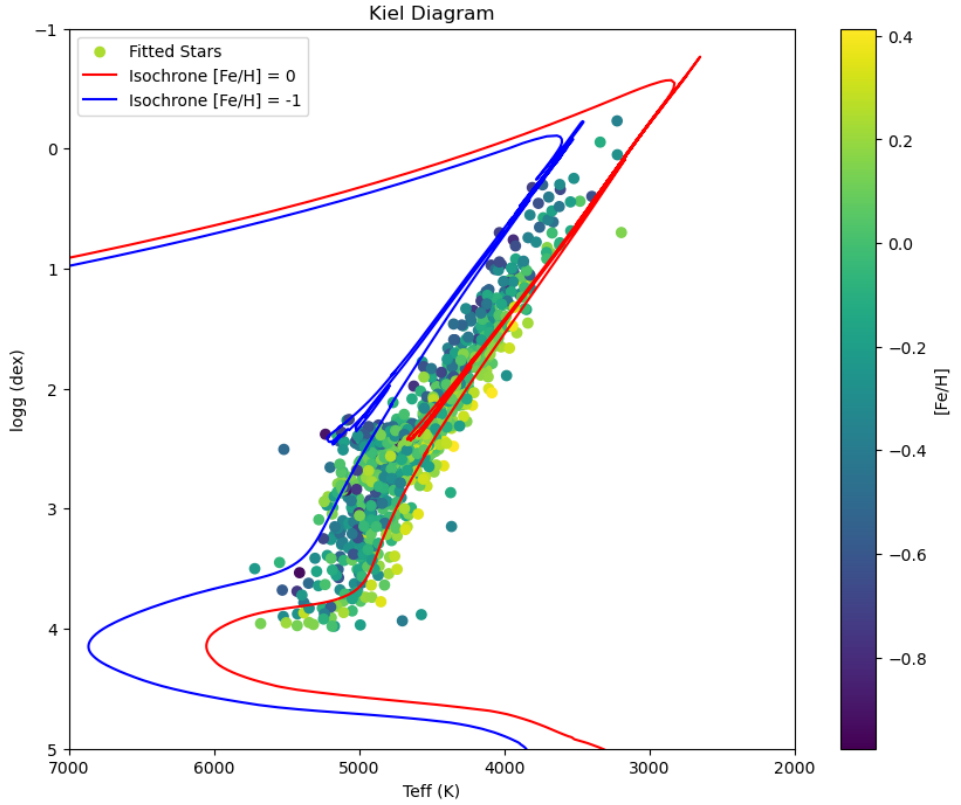

Figure 9: The diagram depicts the distribution of fitted stars in terms of surface gravity versus effective temperature, color-coded by metallicity. The overlaid isochrones for [Fe/H] = 0 and [Fe/H] = -1 illustrate the theoretical evolutionary tracks of stars at different metallicities.

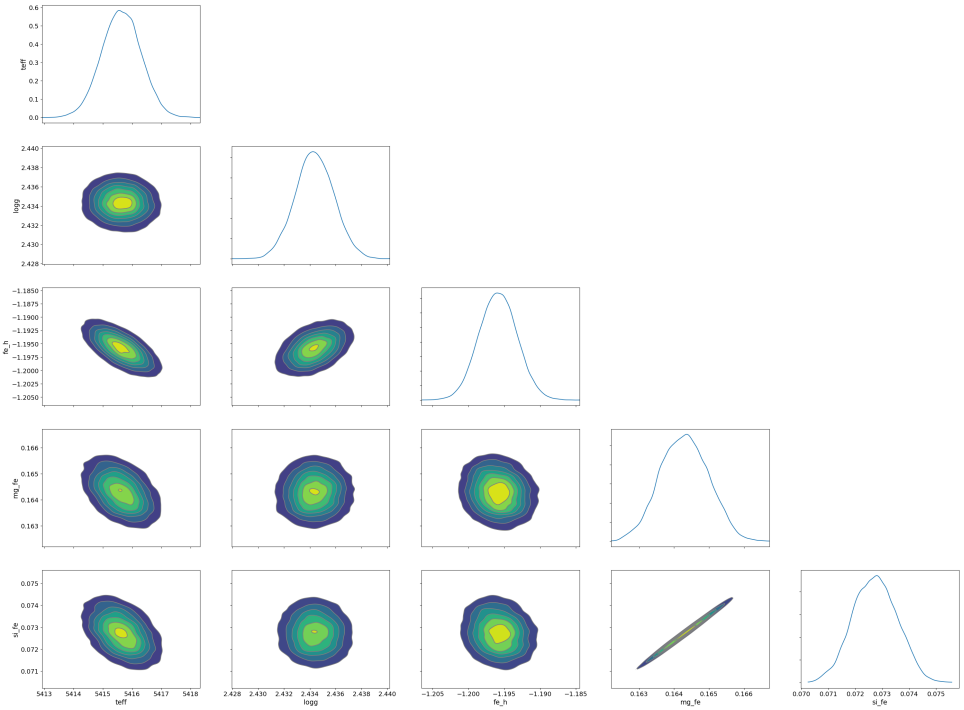

Figure 10: The corner plot above presents the posterior distributions and covariances for the five stellar labels obtained from the MCMC fitting of a mystery spectrum using a spectral model integrated with pymc modeling. The plot indicates the degree of uncertainty and the correlation between parameters obtained from my prediction labels function.

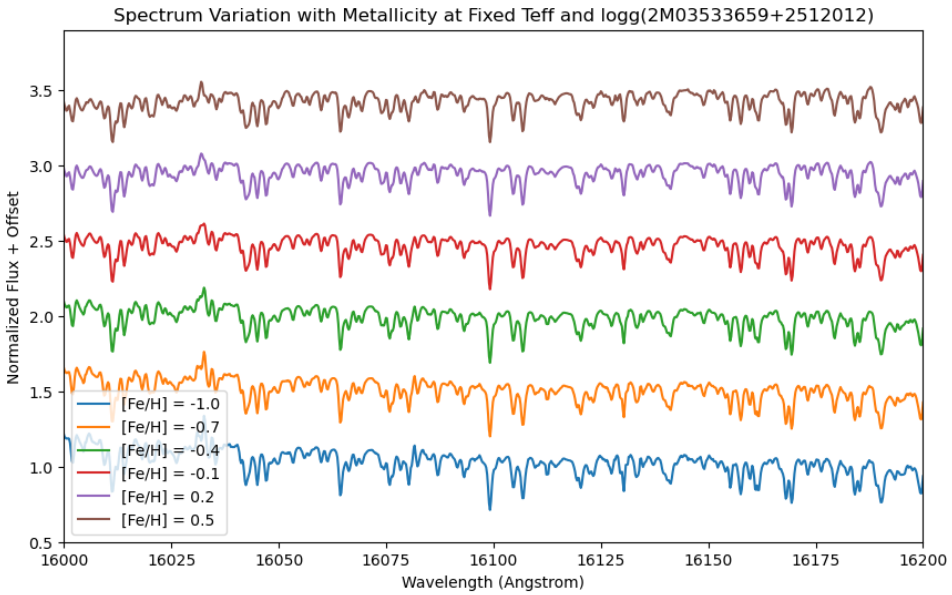

Figure 11: plot above visualizes the variation in normalized flux from sampled star (2M03533659+2512012) across a segment of the spectrum from 16000 to 16200 Angstroms as metallicity changes at fixed effective temperature and surface gravity. Each curve is color-coded to represent different [Fe/H] values, ranging from -1 to 0.5, illustrating the spectral influence of metal abundance.

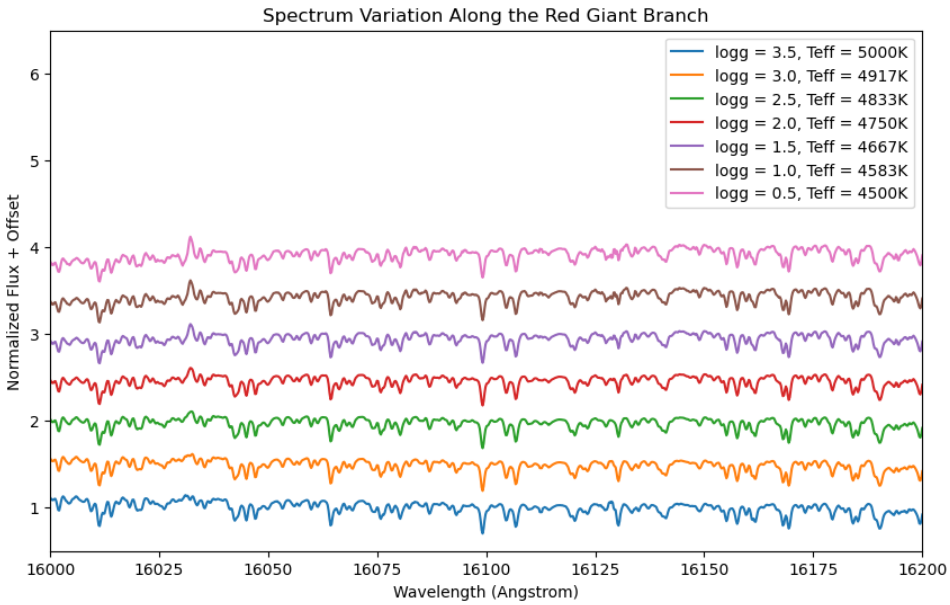

Figure 12: Here is how a stars spectrum flux changes as it moves along the branch. Each line represents a mix of surface gravity and effective temperature while keeping the metallicity constant at 0. The shifted spectra show how the stars spectral characteristics transform as it expands and cools following a path based on an isochrone (referring to the one, in the Kiel diagram)..

4 Discussion

4.1 Data Acquisition: Downloading APOGEE Spectra

ApStar files consist of combined spectra from visits, to a star adjusted for Doppler shifts caused by Earths orbit. This adjustment fluctuates with the changing position of Earth. The flux of the spectrum measured in erg/s/cm^2/Å to represent the energy received across time, area and wavelength range offers a profile of the energy distribution in the stars light.

4.2 Data Preparation: Extracting Stellar Labels from the allStar Catalog

The classification between dwarf and giant stars is done by evaluating the surface gravity,log g. A higher log g indicates a star is relatively compact (for example here log g < 4), characteristic of dwarfs, while a lower value signifies an expanded star, as seen in giants.

$$G = 6.67430 \times 10^{-8}\ \text{cm}^3\ \text{g}^{-1}\ \text{s}^{-2} \quad //Gravitationalconstantincgsunits \tag{1}$$

$$M_{\text{sun}} = 1.989 \times 10^{33}\ \text{g} \quad //Solarmassingrams \tag{2}$$

$$R_{\text{sun}} = 6.957 \times 10^{10}\ \text{cm} \quad //Solarradiusincm \tag{3}$$

$$g = \frac{G \cdot M}{R^2} \quad //gincgsunits \tag{4}$$

$$\log_{10} g \quad //Logarithmofg \tag{5}$$

$$M_{\text{star}} = 1 M_{\text{sun}} \tag{6}$$

$$R_{\text{main sequence}} = 1 R_{\text{sun}} \tag{7}$$

$$R_{\text{pre-he flash}} = 100 R_{\text{sun}} \tag{8}$$

$$R_{\text{core he burning}} = 15 R_{\text{sun}} \tag{9}$$

$$\log g_{\text{main sequence}} = \log_{10}\left(\frac{G \cdot M_{\text{star}}}{R^2_{\text{main sequence}}}\right)$$
$$\log g_{\text{pre-he flash}} = \log_{10}\left(\frac{G \cdot M_{\text{star}}}{R^2_{\text{pre-he flash}}}\right) \tag{10}$$
$$\log g_{\text{core he burning}} = \log_{10}\left(\frac{G \cdot M_{\text{star}}}{R^2_{\text{core he burning}}}\right)$$

In the case of a star with the mass of the Sun, its log g on the main sequence is around 4.44. As the star approaches the helium flash, 100 times the Sun's radius, its log g reduces to 0.44. During the core helium-burning phase, a radius 15 times that of the Sun, the log g value is around 2.09.

4.3 Data Normalization: Implementing Pseudo-Continuum Normalization

Pseudo continuum normalization, as discussed by Ness et al. In 2015 entails adjusting the spectrum to resemble a line overall making it easier to compare characteristics across spectra by reducing the impact of broad spectral features and instrument responses. Unlike normalization, which aligns the spectrum with the stars actual physical continuum level pseudo continuum normalization aligns it with a simplified version, in the form of a low order polynomial that provides a smoothed representation of the spectrum rather, than its true underlying continuum. This method is valuable as it highlights lines and other small scale features for determining stellar properties in our lab setting.

4.4 Model Optimization: Fitting the Training Set Spectra (A)

The spectral model :

$$X\theta_\lambda = f_\lambda$$

where:

- X is the design matrix, each row corresponds to the label vector for a star in the training set. For a 2nd order polynomial model with 5 labels, each row of X would be:

$$\left[1, \text{Teff}, \text{logg}, [\text{Fe/H}], [\text{Mg/Fe}], [\text{Si/Fe}], \text{Teff}^2, \text{Teff} \times \text{logg}, \ldots, [\text{Si/Fe}]^2\right]$$

- θ_λ is the vector of model parameters for the pixel at wavelength λ. For a 2nd order polynomial model with 5 labels, there are:
  - 1 parameter for the constant term
  - 5 parameters for the linear terms
  - 15 parameters for the quadratic terms (including cross-terms)

  In total, there are 21 free parameters in θ_λ.

4.5 Model Optimization: Fitting the Training Set Spectra (C)

Total number of free parameters in the spectral model: 180075, This number was acheived through multiplying the number wavelength pixels and the number parameters per pixel

4.6 Gradient Analysis: Assessing Sensitivity to Stellar Labels

In the gradient spectra plots for the labels [Mg/Fe], and [Si/Fe], peaks in sensitivity (i.e. regions of the spectrum where the gradient is large) align with the wavelengths of established Mg, and Si known absorption lines based off the ones shown in https://github.com/jobovy/apogee/blob/main/apogee/spec/plot.py Those correlations confirm the model's accuracy in identifying the spectral features influenced by elemental abundances such as these presented. Furthermore, the intrinsic scatter term s*lambda**2 exhibits peaks at the same wavelengths as the known absorption lines for Mg and Si aswell, suggesting that the model captures variances beyond elemental abundance effects.

4.7 Error Analysis: Investigating Discrepancies in Cross-Validation Results

In regards to seeing if there are flags in the catalog indicating the allStar labels might not be reliable for the training model to predict labels, I simulated a set of random samples that resemble the label data from the ASPCAP catalog, specifically for three stellar parameters: effective temperature (Teff), surface gravity (Logg), and metallicity (Fe/H). The aim was to create samples that are close to the original ASPCAP values, incorporating a small margin of variation to mimic real observational data or measurement errors.

For Teff, a slight bias of -1.01 units and a scatter of 88.38 units were produced around the ASPCAP values. This small bias indicates that the generated temperatures are on average just one unit below the ASPCAP temperatures, and the scatter represents the standard deviation of the generated temperatures around this biased mean, suggesting that most of the generated Teff values lie within ±88.38 units of the ASPCAP values. Logg and Fe/H had even smaller biases (close to 0) and scatters (0.05 and 0.01, respectively).

These results suggest that the generated random samples for Teff have a small bias and a reasonable scatter around the original ASPCAP values, which indicates that the generated data points are close to the original values but with some expected random variation which by removing the few outliers it would likely be able to improve my model by even more which was done in talk of my science. The Logg and Fe/H labels, on the other hand, show almost no bias and very small scatter which shouldn't have much of an effect on my model but should still be cautious regardless.

4.8 Kiel Diagram: Visualizing Stellar Properties

According to the diagram, from Kiel the labels assigned to the stars in the validation set follow a pattern that matches well known characteristics of stellar development. The colors of the stars, which indicate their metal content display trends indicating that stars with metal content are generally cooler and have less surface gravity. The consistency between the labeled stars and the 6 Gyr MIST isochrone for metal content suggests that both the fitting process and models utilized are trustworthy. Additionally the positioning of stars along the isochrone with [Fe/H] = 1 confirms that the observed trend in metallicities from aligns, with the expected isochrone pattern.

4.9 Bayesian Inference: Fitting Mystery Spectrum with MCMC

The layout of the graph shows that there are some uncertainties, in the parameters as seen by the bell shaped distributions at the edges and clear outlines indicating defined fits. When compared to the errors derived from ASPCAP labels our uncertainties appear smaller suggesting that our model may be too confident or not fully addressing errors. The same is true for those inferred from validation scatter. Differences in uncertainty could arise from factors such, as choices, model complexity, data quality and the fitting methods used in the code all potentially impacting the estimated error margins shown here.

4.10 Red Giant Branch Evolution: Tracing Changes in the RGB Spectrum

The plot depicts the changes in appearance of a star moving along the branch with a consistent metallicity level ([Fe/H]=0) but varying gravitational forces (logg ranging from 3.5, to 0.5) adjusting its temperature (Teff from 4500K to 5000K) to match an isochrone specifically those based on the ones referenced in question 11. Each spectrum displays differences with stars of gravity exhibiting broader features and a reddish shift indicating cooler temperatures. As stars progress along the RGB these distinctions become more obvious. To distinguish between a star with logg and a warmer star with higher logg but richer, in metals one should observe the intensity and breadth of absorption lines; metal rich stars exhibit deeper more distinct lines compared to the widened features seen in cool low gravity giants – this observation is further supported by our kiel diagram.

5 Conclusion

In summary we successfully met our objective of developing a data driven model to forecast spectra based on stellar features showcasing the potential of such models, in analyzing astronomical data. By collecting, preparing and standardizing data well as conducting thorough model training and validation we have established an approach to comprehend the spectral attributes of RGB stars. Applying this model to the APOGEE survey data revealed the link between a stars characteristics and its spectrum providing valuable insights into stellar development especially during the red giant branch phase. The use of inference, with MCMC to fit a spectrum further underscored the effectiveness of our model.

References

APOGEE Survey; https://doi.org/10.48550/arXiv.1509.05420

Ness et al. 2015; https://doi.org/10.48550/arXiv.1501.07604

Majewski et al. 2017; https://doi.org/10.48550/arXiv.1509.05420

Ahumada et al. 2020; https://doi.org/10.48550/arXiv.1912.02905

Holtzman et al. 2015; https://doi.org/10.48550/arXiv.1501.04110

Apogee GitHub repository; https://github.com/jobovy/apogee/blob/main/apogee/spec/plot.py

PyMC docs;

$https://www.pymc.io/projects/examples/en/latest/howto/blackbox_external_likelihood_numpy.html$