lab2_student

January 28, 2024

Before you turn this problem in, make sure everything runs as expected. First, **restart the kernel** (in the menubar, select Kernel→Restart) and then **run all cells** (in the menubar, select Cell→Run All).

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE", as well as your name and collaborators below:

[1]:
```
NAME = "John Parrack"
COLLABORATORS = "Amy, Anna, Roshan"
```

1 Physics C170M/270M Lab 2: Naive Bayes

In this lab, you will - Incorporate the 8 steps of machine learning - Calculate a small Naive Bayes problem by hand - Code a Gaussian Naive Bayes model to analyze the iris dataset. - Use scikit-learn and Naive Bayes models to make predictions - Application of Naive Bayes in Astrophysics: classify stars at the center of our Galaxy

Reminder: save and checkpoint often!

Lab Created by: Tuan Do & Bernie Boscoe

Last updated by: Tuan Do

Fill in the missing values, by computing the priors. (Just use a piece of paper, you will need them for a later question)

1.1 Question 1

(2 pts.) So, in the figures above, we have calculated $P(x_i|y_j)$ for each x_i in X and y_j in \$y \$ manually in the tables 1-4. For example, probability of playing golf given that the temperature is cool, i.e P(play golf = Yes | temp = Cold) = 2/5.

Also, we need to find class probabilities ($P(y)$). For example, P(play golf = Yes) = 5/10.

today = (Sunny, Hot, Low)

so $P(Yes|today) = P(Sunny|Yes) * P(Hot|Yes) * P(Low|Yes) * P(Yes)$ (we can ignore denominator) = ?

Type in the four fractions below, and the result when you multiply them together.

1. 1. P(Yes|today) = 2/5 * 3/5 * 1/5 * 1/2

Numerator = 6

1.2 Question 2

(1 pt.) Now, what is P(No | Today)? Next, convert these two values into probabilities by making the sum equal to 1 (normalization): example: P(Yes|today)/ P(Yes |today) + P(No | today) Write the probability that golf will be played today. How likely is it golf will be played today?

2. P(No|today) = 3/5 * 1/5 * 4/5 * 1/2

Numerator = 12

P(Yes|today) = 6/18 = 1/3

2 Iris Classification with Naive Bayes

In this part of the assignment, you will again use the Iris dataset. Recall, it consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). Four features were measured from each sample: the length and the width of the sepals and petals, in centimeters. For a reference, see the following papers:

- R. A. Fisher. "The use of multiple measurements in taxonomic problems". Annals of Eugenics. 7 (2): 179–188, 1936.

Your goal is to construct a Naive Bayes classifier model that predicts the correct class from the sepal length and sepal width features. This lab will help learn about using probability distributions in the Naive Bayes classifier.

[2]:
```
# run this cell
#### PACKAGE IMPORTS ####
import time
import numpy as np
import matplotlib.pyplot as plt
import sklearn as sk
from sklearn.metrics import accuracy_score
from sklearn import datasets, model_selection
import pandas as pd
import seaborn as sns
%matplotlib inline
```

3 Step 1: Look at the big picture

What is the problem you want to solve? How do you plan to use and benefit from the the machine learning model? What is your metric for success?

In this exercise, we will give you the goal and the metric:

Machine Learning Goal: classify irises based on the measurement of their petals.

Metric: accuracy - defined as the number of correctly typed irises divided by the total number of irises

4 Step 2: Get the data

What data are available to you for training your model? How will you access and download all the data? The type and quantity of data available will often impact your choice in the subsequent stages.

Scikit Learn has a bunch of pre-built machine learning datasets that are helpful for learning new models and testing. #### Load and prepare the data

We will first read in the Iris dataset, and then split the dataset into training and test sets. The extra cell below the code solution is provided to check your work.

4.1 Question 3

[3]:
```
# (1 pt.) Load the iris dataset (hint: see how Lab 1 loaded the Iris dataset)
iris = datasets.load_iris()
iris_df = pd.DataFrame(data= np.c_[iris['data'], iris['target']], columns=␣
 ↪iris['feature_names'] + ['target'])
```

5 Step 3: Explore the data

Discover and visualize the data to gain insight. For example, are there more representatives of certain types of data than others?

5.1 Question 4

(6 pts)

For this lab, please answer the following questions about your data. Use as many cells and plots as you like.

1. What are the features? What is the label?
2. Compute some summary statistics about your dataset. For example, the number of samples, mean, median, dispersion of the features.
3. Plot some of the features to get a sense of the data.

[4]:
```
iris_df.head()
```

[4]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	
1	4.9	3.0	1.4	0.2	
2	4.7	3.2	1.3	0.2	
3	4.6	3.1	1.5	0.2	
4	5.0	3.6	1.4	0.2	

```
0     0.0
1     0.0
2     0.0
3     0.0
4     0.0
```

1. The features are: sepal length, sepal width, petal length, and petal width. The label is the target, it one of three values, 0.0, 1.0, and 2.0 representing the three varietes of iris in the dataset.

2.

[5]:
```
iris_df.describe()
```

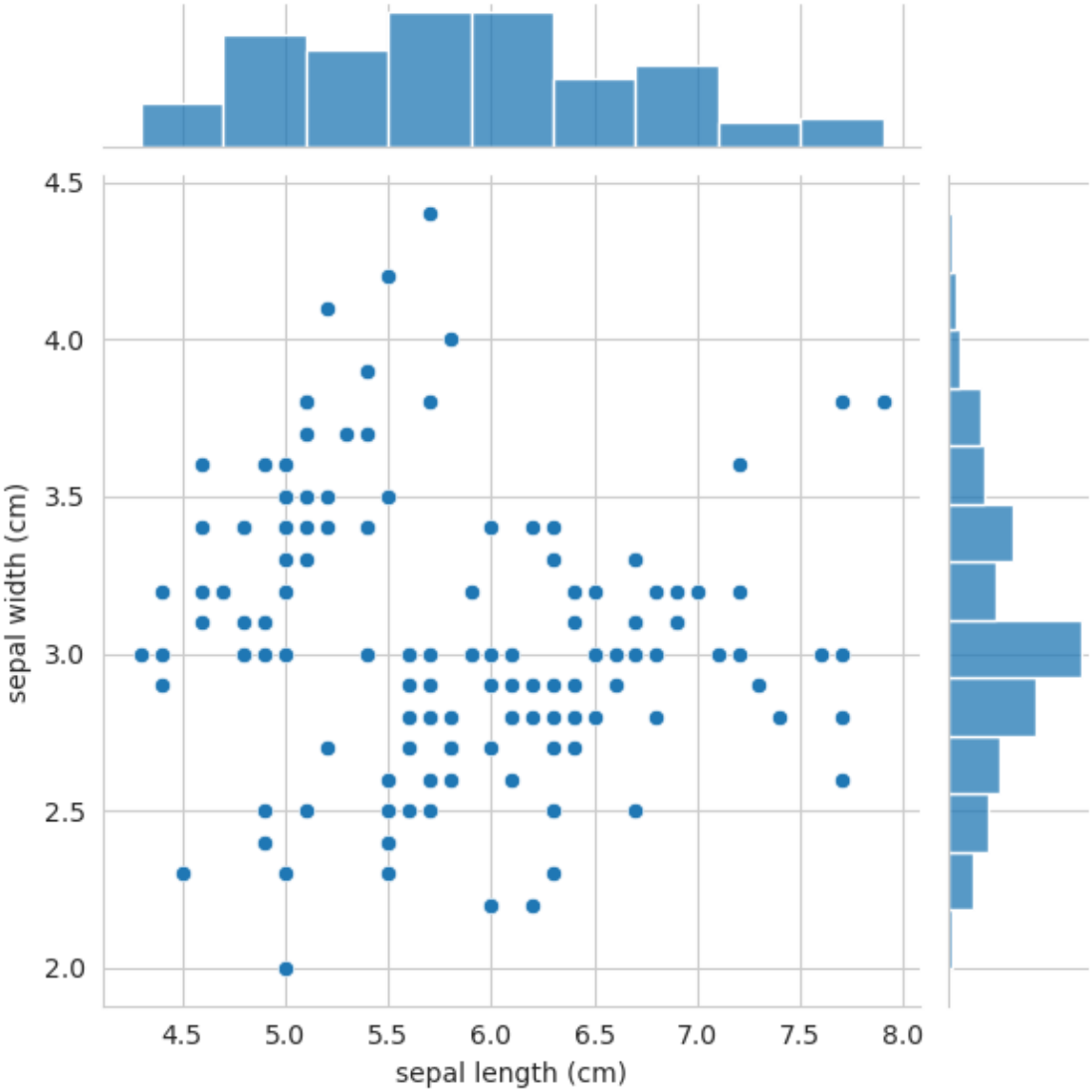
[5]:
```
       sepal length (cm)  sepal width (cm)  petal length (cm)  \
count         150.000000        150.000000         150.000000
mean            5.843333          3.057333           3.758000
std             0.828066          0.435866           1.765298
min             4.300000          2.000000           1.000000
25%             5.100000          2.800000           1.600000
50%             5.800000          3.000000           4.350000
75%             6.400000          3.300000           5.100000
max             7.900000          4.400000           6.900000

       petal width (cm)      target
count        150.000000  150.000000
mean           1.199333    1.000000
std            0.762238    0.819232
min            0.100000    0.000000
25%            0.300000    0.000000
50%            1.300000    1.000000
75%            1.800000    2.000000
max            2.500000    2.000000
```

3.

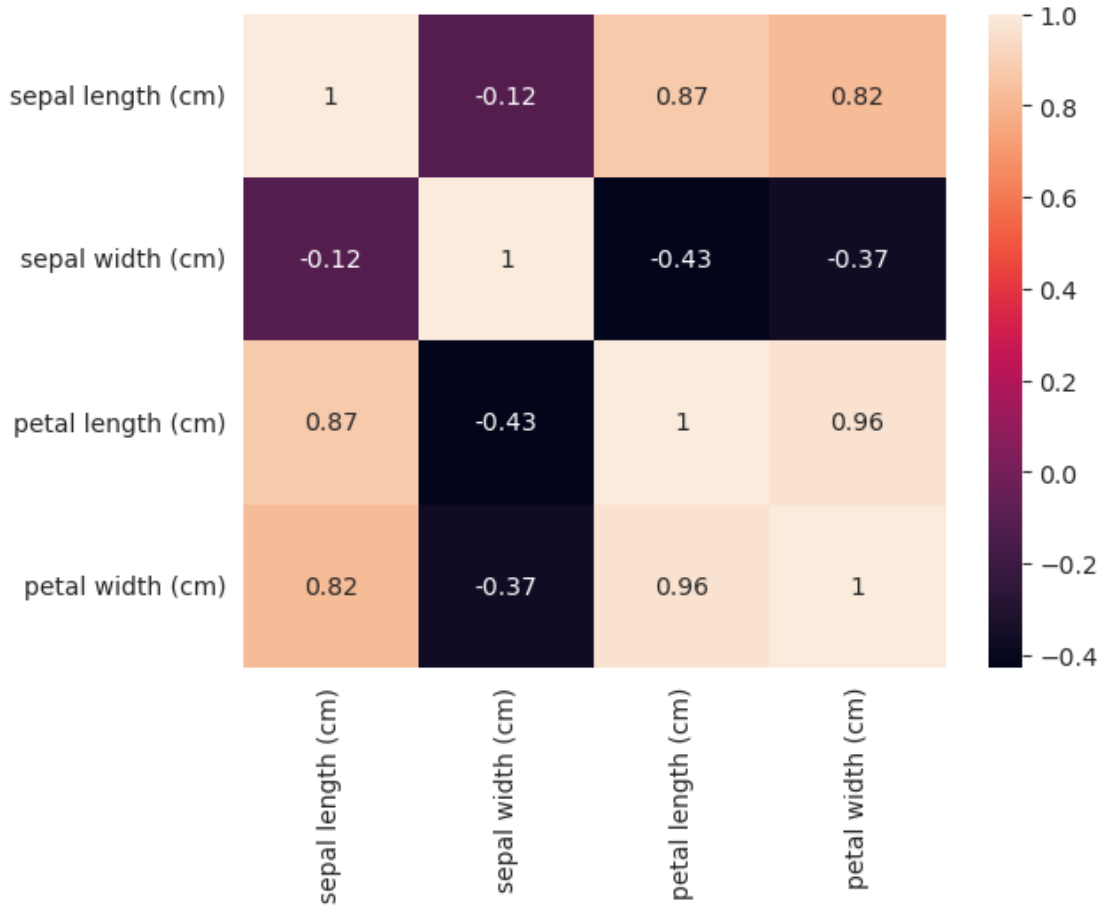
[6]:
```
# Here let's make some plots like in lab 1,
```

[7]:
```
sns.set_style('whitegrid')
sns.jointplot(x= 'sepal length (cm)', y = 'sepal width (cm)', data=iris_df)
```

[7]:
```
<seaborn.axisgrid.JointGrid at 0x78003e709cd0>
```



[8]:
```
data = iris_df.drop('target', axis=1)
sns.heatmap(data.corr(), annot = True);
```



6 Step 4: Prepare the data for ML algorithms

Machine learning algorithms often require data to be pre-processed in a certain way, such as scaling numerical values or map categories to other representations. You will also need to decide what data to use as training and as testing. You may also need to develop a strategy to deal with missing data.

Use only the first two features: **sepal length** and **width** for this assignment

6.1 Question 5

[9]:
```
# As we saw in lab 1, the flowers can be easily classified by petal length/
  ↪width. Sepal length/width should be more difficult
# (1 pt.) Create an array called data with the sepal length and width features␣
  ↪and
# Create an array called targets with the target values

data = iris_df[['sepal length (cm)', 'sepal width (cm)']].values
```

```
targets = iris_df[['target']].values
```

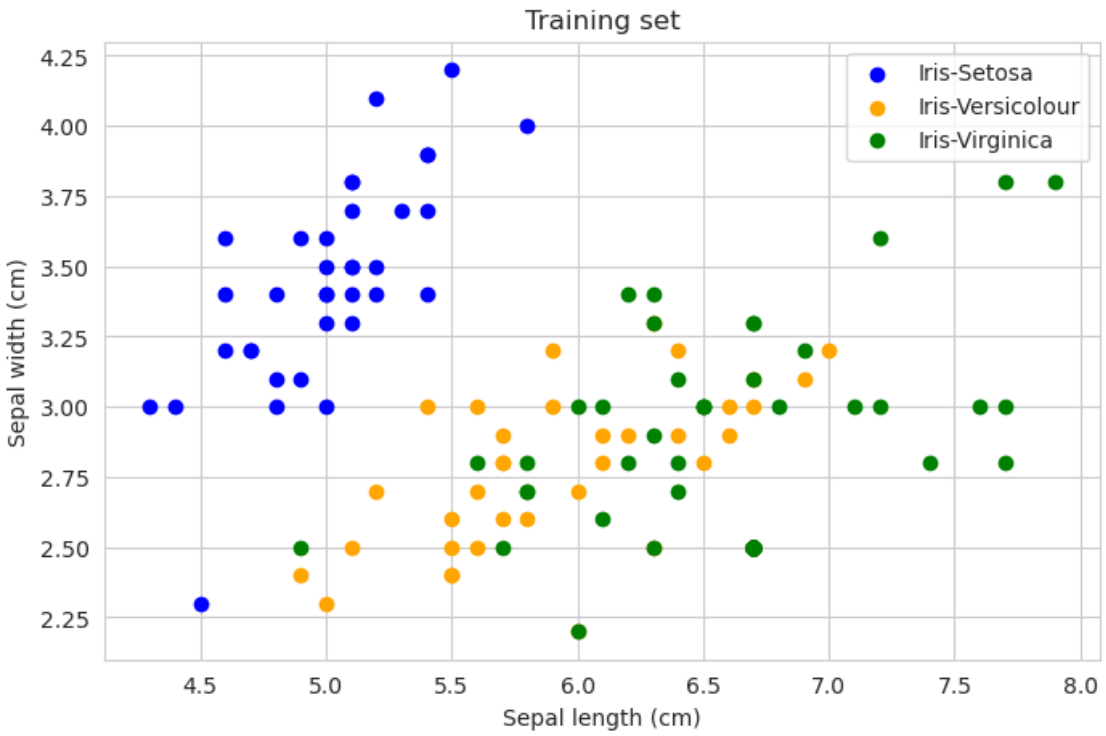
`sklearn` has a really helpful function called `train_test_split` that helps to randomly split your data and targets for training and testing. Please read the documentation on `train_test_split` before running the following cell. Note that there is an important keyword `test_size`. (Look at what random_seed can do for you as well, we will use that later.)

A short summary of terminology used below:

- **training** - data used in training the machine learning model
- **test** - data used to evaluate the performance of the model. Will use this information to further refine the model.
- **validation** - the data left aside in the beginning only to be used at the very end when you are finished with your model and want to test on data the model has never seeen.

[10]:
```
# Define a random seed for reproducibility
random_seed = 42

# Split the data into training + validation and test sets
x_train_all, x_validate, y_train_all, y_validate = model_selection.
 ↪train_test_split(
    data, targets, test_size=0.1, random_state=random_seed)

# Now split the training data further into training and testing sets
x_train, x_test, y_train, y_test = model_selection.train_test_split(
    x_train_all, y_train_all, test_size=0.2, random_state=random_seed)
```

[11]:
```
# Example of plotting the training data

labels = {0: 'Iris-Setosa', 1: 'Iris-Versicolour', 2: 'Iris-Virginica'}
label_colours = ['blue', 'orange', 'green']

def plot_data(x, y, labels, colours):
    for c in np.unique(y):
        inx = np.where(y == c)
        plt.scatter(x[inx, 0], x[inx, 1], label=labels[c], c=colours[c])
    plt.title("Training set")
    plt.xlabel("Sepal length (cm)")
    plt.ylabel("Sepal width (cm)")
    plt.legend()

y_train = y_train.astype(int)
plt.figure(figsize=(8, 5))
plot_data(x_train, y_train, labels, label_colours)
```



6.2 Question 6

[12]:
```
# (1 pt.) Create an array called priors that has the prior probability of␣
↪observing each class.
# Be sure to keep track of which class belongs to which element of the array!


y_train_flat = y_train.flatten() # Flatten y_train from (108, 1) to (108) for␣
↪bincount

# Calculate the priors based on training set
class_counts = np.bincount(y_train_flat)
priors = class_counts / y_train_flat.size
print(priors)

# Check that priors sum to 1
np.sum(priors)
```

```
[0.33333333 0.32407407 0.34259259]
```

[12]:
```
1.0
```

6.3 5b Build the Naive Bayes Classifier

6.4 Question 7

Build the Naive Bayes Classifier using Gaussian distributions.

[13]:
```
# (3 pt.) Build Classifier here
# YOUR CODE HERE, Actually build don't use a sklearn model

# Function to calculate mean/variance
def calculate_mean_variance(X, y):
    classes = np.unique(y)
    means = {}
    variances = {}

    for cls in classes:
        X_cls = X[y == cls]
        means[cls] = X_cls.mean(axis=0)
        variances[cls] = X_cls.var(axis=0)

    return means, variances

# Function to calculate gaussian probability density function
def gaussian_pdf(x, mean, var):
    const = 1 / np.sqrt(2 * np.pi * var)
    pdf = const * np.exp(-((x - mean) ** 2) / (2 * var))
    return pdf

def naive_bayes_classifier(X, means, variances, priors):
    classes = np.unique(y_train_flat)
    n_features = X.shape[1]
    predictions = []

    for x in X:
        class_probs = {}
        for cls in classes:
            class_prob = priors[cls]
            for i in range(n_features):
                class_prob *= gaussian_pdf(x[i], means[cls][i],␣
 ↪variances[cls][i])
            class_probs[cls] = class_prob
        predictions.append(max(class_probs, key=class_probs.get))

    return np.array(predictions)
```

6.5 5c Train the model

Training the model in this case is to determine the parameters of the Gaussians describing the joint probability of observing the sepal length or width with respect to the class.

6.6 Question 8

(3 pt.) Using as many cells as you need, determine the parameters for the joint probability distribution.

[14]:
```
# Calculate the means and variances for the joint probability P(X_i|Y=y_k)
means, variances = calculate_mean_variance(x_train, y_train_flat)
```

6.7 5d Test the model predictions

`sklearn` comes with a lot of useful routines to compute metrics for machine learning applications. In this lab, you'll use `accuracy_score` from `sklearn.metrics` to compute the model accuracy.

6.8 Question 9

[15]:
```
# (1 pt.) make a array called pred that is your prediction for most likely␣
 ↪class given your test data
# Make predictions
pred = naive_bayes_classifier(x_test, means, variances, priors)
print(pred)
```

```
[2 0 1 0 0 1 2 1 1 1 1 1 2 2 0 1 1 1 2 0 0 2 0 0 2 2 1]
```

[16]:
```
# this line will give you your accuracy score for the test data
score = accuracy_score(y_test, pred)

y_test_flat = y_test.flatten().astype(np.int64)
print('True Label:')
print(y_test_flat)

print('\nPredicted Label:')
print(pred)

misses = (y_test_flat != pred).astype(int)
print('\nIncorrect Classifications:')
print(misses)

print('\nTest Accuracy Score')
print(f'{score*100:.4f}%')
```

```
True Label:
[2 0 1 0 0 1 2 1 1 2 1 1 1 2 0 2 1 2 2 0 0 1 0 0 2 2 2]

Predicted Label:
[2 0 1 0 0 1 2 1 1 1 1 1 2 2 0 1 1 1 2 0 0 2 0 0 2 2 1]

Incorrect Classifications:
[0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0 0 0 1 0 0 0 0 1]
```

```
Test Accuracy Score
77.7778%
```

6.9 Question 10

For this case, we will not modify our model, so we can just go ahead evaluate the accuracy of the validation data that you saved earlier.

Below, find the accuracy of the validation data.

[17]:
```
# (1pt.) determine the accuracy score of your validation data
val_pred = naive_bayes_classifier(x_validate, means, variances, priors)
val_score = accuracy_score(y_validate, val_pred)
print('\nValidation Accuracy Score')
print(f'{val_score*100:.4f}%')
```

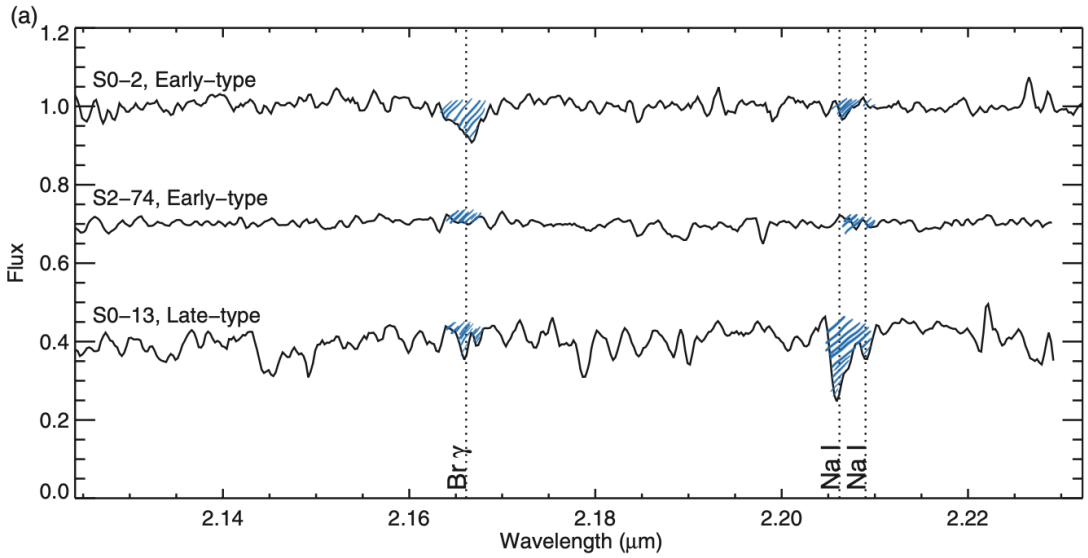
```
Validation Accuracy Score
93.3333%
```

Congratulations! You have now completed the iris classification. In this iris classification, we won't go through Steps 6 to 8 of the machine learning workflow, but we will in the next part of this lab.

7 Classifying Stars at the Galactic Center

The Galactic center contains a complex population of stars. Multiple generations of star formation have occured, including as recently as 4 to 6 million years ago. The most reliable way to determine the ages of the stars is measure their spectra to examine their spectral features. Generally stars near the supermassive black hole in Galactic center are old (> 1 billion years) or young (4-6 million years). The stars that are bright enough for us to see are young stars and massive (> 10 Msun) with high surface temperatures (>20,000K), or old stars in their red giant phase with lower mass (~1 Msun) and cool temperatures (3000 to 4000 K). This difference in temperature leads to very different spectral features.

In the spectra below, there are two examples of young stars and one example of an old star. The young star spectra are classified as 'early-type'. Because the temperature is high, many electrons have been ionized from atoms in its atmosphere so few absorption lines remain. The most prominent absorption line for early-type stars in this wavelength region is from hydrogen (specifically, the Brackett gamma line). Sometimes, the temperature is so high that even the electron in hydrogen is ionized, so there is very little hydrogen absorption. For the old red-giant stars, the temperatures are fairly low, so there are now many atomic absorption lines. The strongest lines in this wavelength range are from sodium (Na) atoms. In addition, because the temperature is cool, the hydrogen lines become less prominant because there is not enough high energy photons that are available to make those absorption lines.



7.1 Feature Engineering: Equivalent Widths and Cross-Correlation

To simplify and automate the process of classifying stars, we can reduce the spectrum (flux measurements at different wavelengths) into a more compact representation that hopefully still carries a lot of the information necessary for classification. In this case, we pick the strongest spectral lines (Br gamma and Na) and turn them into features for our machine learning models. We will use the concept of equilvalent width, which is related to the depth of the absorption features. In the figure above, the shaded blue regions represent the equivalent widths of the lines. The stronger the lines, the large the equivalent widths. For more details see: https://en.wikipedia.org/wiki/Equivalent_width. We will define positive equivalent widths as absorption lines (like in the figure above), and negative equivalent widths as emission lines.

We will also use the cross-correlation coefficient of the spectra with templates as a potential feature. This feature is the maximum from cross-correlating a template young star and old star spectrum with the observed spectrum. You can think of this correlation coefficient as how closely matched the observed spectrum is with a template spectrum. The correlation in principle should go from 0 to 1. With 0 meaning it is not at all alike and 1 being an identical match. The correlation coefficent given in the data for this lab varies widely because the observed spectra often have more noise the the spectrum above and we only cross correlate the observed spectra with one model young star and one model old star. There are also intrinsic variations in the spectra that can also reduce the correlations.

The dataset you'll be using has the equivalent widths and cross correlation coefficient measured using an automated method for spectra that have first been classified by humans, so there are labels.

8 Step 1: Look at the big picture

What is the problem you want to solve? How do you plan to use and benefit from the the machine learning model? What is your metric for success?

In this lab, we will give you the goal and the metric

Lab goal: classify stars at the Galactic center into young or old.

Metric: accuracy - defined as the number of correctly typed stars divided by the total number of stars

9 Step 2: Get the data

What data are available to you for training your model? How will you access and download all the data? The type and quantity of data available will often impact your choice in the subsequent stages.

Here, the data is straightforward. We have provided you with the training data in the file `galactic_center_stars_training.csv`

In the training data, there should be 8 columns: Na equivalent width, Na equivalent width error, correlation coefficient with an old star template, Br gamma equivalent width, Br gamma equivalent width uncertainty, signal-to-noise ratio of the spectrum, and young or old. The units of equilvalent widths and their uncertainties are in Angstroms. The correlation coefficients are unit-less.

There is also a file called `galactic_center_stars_eval.csv` that has no labels. We'll be using that file to grade this assignment.

Note: Also correlation coefficient with young star template is one of the 8 collumns in the csv file.

9.1 Question 11

[18]:
```
# (1 pt.) In this cell, load the training data and the data for grading into␣
 ↪two pandas data frames.
# and check that they are there by looking at each head.
# YOUR CODE HERE
gcs_train = pd.read_csv('galactic_center_stars_training.csv')
gcs_eval = pd.read_csv('galactic_center_stars_eval.csv')
```

[19]:
```
gcs_train.head()
```

[19]:

	Na	Na_err	Corr	Br	Br_err	Yng_corr	SNR	Type
0	1.11	0.91	0.81	2.57	0.53	0.54	17.69	old
1	5.40	0.78	0.77	0.98	0.26	0.36	20.74	old
2	5.71	0.80	0.81	0.96	0.22	0.34	20.23	old
3	2.54	1.45	0.93	2.94	0.87	0.11	10.92	old
4	7.13	1.42	0.83	1.33	0.46	6.28	11.13	old

[20]:
```
gcs_eval.head()
```

[20]:

	Na	Na_err	Corr	Br	Br_err	Yng_corr	SNR
0	5.30	0.67	0.89	0.45	0.13	0.29	24.41
1	5.53	1.73	0.85	1.00	0.45	31.12	9.06
2	0.75	1.90	0.53	-1.52	1.20	24.63	8.20
3	7.32	2.34	0.72	2.61	1.34	0.19	6.60

```
4  2.91    0.80  0.88  1.10    0.27      0.23  20.12
```

10 Step 3: Explore the data

Discover and visualize the data to gain insight. For example, how complete is the dataset? Are there more representatives of certain types of data than others?

10.1 Question 12

(6 pts)

For this lab, please answer the following questions about your data. Use as many cells and plots as you like.

1. What are the features? What is the label?
2. Compute some summary statistics about your dataset. For example, the number of samples, mean, median, dispersion of the features.
3. Plot some of the features to get a sense of the data.

1. The features are: 1. Na (Å) (equivalent width) 2. Na_err (Å) (equivalent width error) 3. Corr (correlation coefficient with an old star template) 4. Br (Å) (gamma equivalent width) 5. Br_err (Å) (gamma equivalent width uncertainty) 6. Yng_corr (correlation coefficient with a young star template) 7. SNR (signal-to-noise ratio of the spectrum)

The label is: 1. Type (young or old)

2.

[21]:
```
gcs_train.describe()
```

[21]:
```
                Na      Na_err        Corr            Br        Br_err  \
count  419.000000  417.000000  419.000000  4.190000e+02  4.190000e+02
mean     3.114630    1.341487    6.307780  9.379330e+05  8.194493e+08
std      2.893386    6.151558   42.377727  1.921929e+07  1.642226e+10
min     -2.560000    0.240000   -0.880000 -4.318304e+05  0.000000e+00
25%      1.020000    0.680000    0.560000  1.015000e+00  2.700000e-01
50%      3.440000    0.900000    0.700000  2.050000e+00  4.900000e-01
75%      4.670000    1.130000    0.830000  3.765000e+00  8.400000e-01
max     41.940000  125.390000  740.740000  3.934078e+08  3.360945e+11

           Yng_corr         SNR
count  4.190000e+02  419.000000
mean   8.085288e+03   20.035442
std    1.141227e+05   10.664952
min   -2.550000e+00   -0.110000
25%    3.000000e-01   14.085000
50%    4.500000e-01   17.780000
75%    7.650000e-01   23.960000
max    1.795982e+06   70.290000
```

[22]:
```
gcs_eval.describe()
```

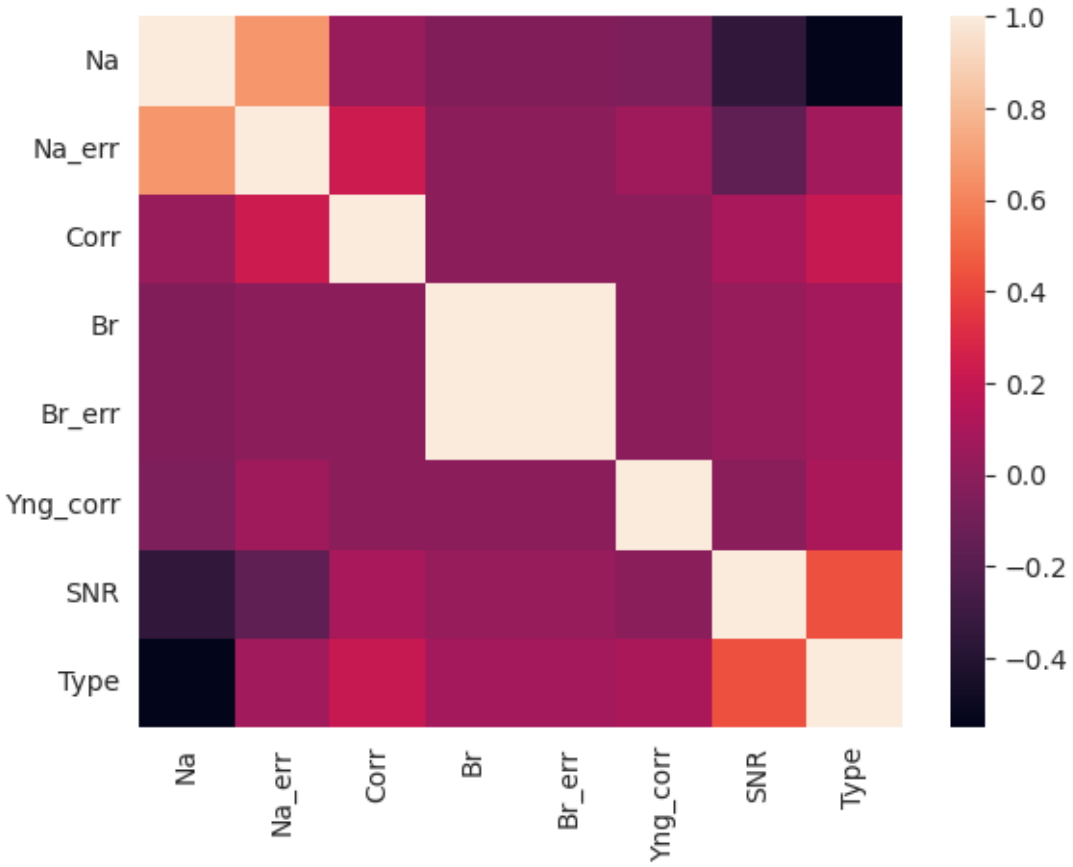
[22]:

	Na	Na_err	Corr	Br	Br_err	Yng_corr
count	90.000000	90.000000	90.000000	90.000000	90.000000	90.000000
mean	2.869778	1.027222	826.392000	2.125111	0.866111	1.939556
std	2.028248	0.518381	7730.325153	2.675019	1.279202	5.061681
min	-1.460000	0.200000	-162.320000	-5.480000	0.000000	-0.240000
25%	0.735000	0.760000	0.560000	0.950000	0.272500	0.290000
50%	3.310000	0.990000	0.670000	1.625000	0.450000	0.480000
75%	4.575000	1.185000	0.795000	3.337500	0.870000	0.647500
max	7.320000	4.120000	73340.070000	10.430000	8.020000	31.120000

	SNR
count	90.000000
mean	19.376111
std	11.204594
min	3.670000
25%	13.400000
50%	16.225000
75%	21.402500
max	86.520000

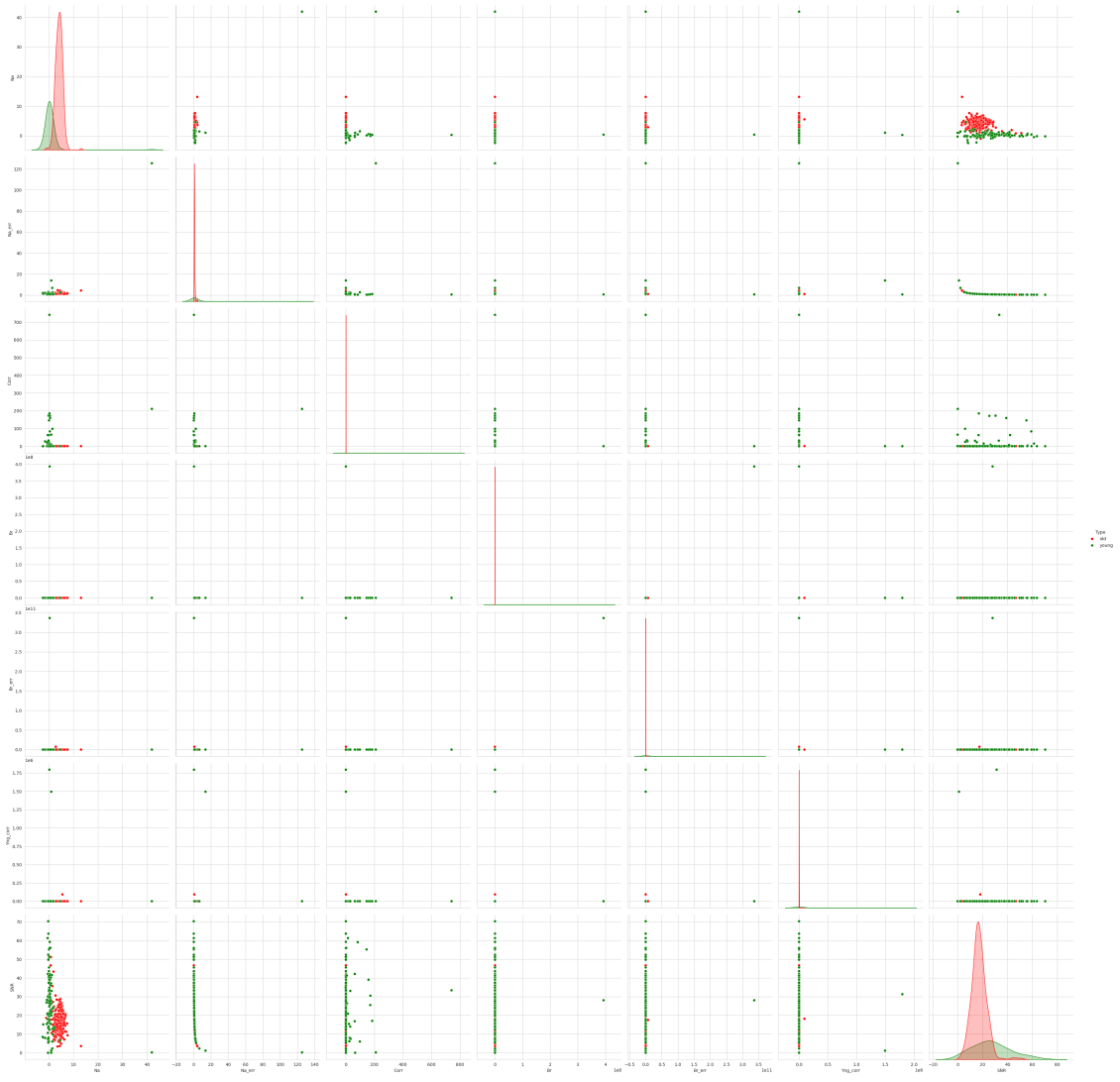
3.

[23]:
```
# Corr matrix

# Copy of DataFrame to avoid modifying the original data
gcs_train_encoded = gcs_train.copy()

# Encode the 'Type' column
gcs_train_encoded['Type'], _ = pd.factorize(gcs_train_encoded['Type'])

corr = gcs_train_encoded.corr()  # Compute the correlation matrix
sns.heatmap(corr)  # Annot=True to print the values inside the square
plt.show()
```



[24]:
```
# Pair plot
palette = {'old': 'red', 'young': 'green'}
sns.pairplot(gcs_train, hue='Type', palette=palette, height=5)
```

[24]: <seaborn.axisgrid.PairGrid at 0x78003e9b47d0>



[ ]:

11 Step 4: Prepare the data for ML algorithms

Machine learning algorithms often require data to be pre-processed in a certain way, such as scaling numerical values or map categories to other representations. You will also need to decide what data to use as training and as testing. You may also need to develop a strategy to deal with missing data.

For this lab, we have already done the feature engineering to get the equivalent widths, their uncertainties, and the correlation coefficients.

11.1 Question 12

(6 pts)

While the features are already chosen, there are a number of other questions that need to be answered:

1. Are there bad data points or outliers?
2. If there are outliers, what is your strategy for dealing with them?
3. Apply your strategy for cleaning and dealing with the data, then split your training data into three parts: training, testing, and validation

[25]:
```
# use as many cells as you need two answer Question 12
```

1. Yes, as we saw in the description of the dataset and in the pair plot above some features have data that deviates wildly from the mean and can safely be considered an outlier.

2. To deal with the outliers I want to try clipping out rows which have very egregious entries for one or more features. There may be a more inteligent way to handle the outliers in the data but I don't have a deep understanding of the underlying physics so its difficult to have intuitions about what should and should not be removed/modified.

I'll create a function to remove data that falls beyond some threshold number of standard deviations and I'll experiment with that threshold number to see what seems to improve model accuracy.

[26]:
```
# YOUR CODE HERE
# raise NotImplementedError()
```

[27]:
```
# Describe your process, clean the data, and show clean_training.describe()
```

3. I'd like to try various methods and test how they change the acuracy of the classifier but for an initial pass at cleaning the data, I'll try clipping out outliers since when scrolling through the csv I see some very egregious entries, and the plots show very clear outliers for various features.

[28]:
```
# Data Cleaning Method 1, clip outliers beyond a certain # of standard␣
↪deviations
def remove_outliers(data, z_threshold=5):
     # Select only numerical columns for outlier detection
    numerical_data = data.select_dtypes(include=[np.number])

    # Calculate mean and standard deviation
    means = numerical_data.mean()
    stds = numerical_data.std()

    # Calculate Z-scores
    z_scores = (numerical_data - means) / stds

    # Keep rows where all values are within the threshold
    filtered_entries = (z_scores.abs() < z_threshold).all(axis=1)
    return data[filtered_entries]
```

[29]:
```
gcs_clean_train = remove_outliers(gcs_train)
gcs_clean_train.describe()
```

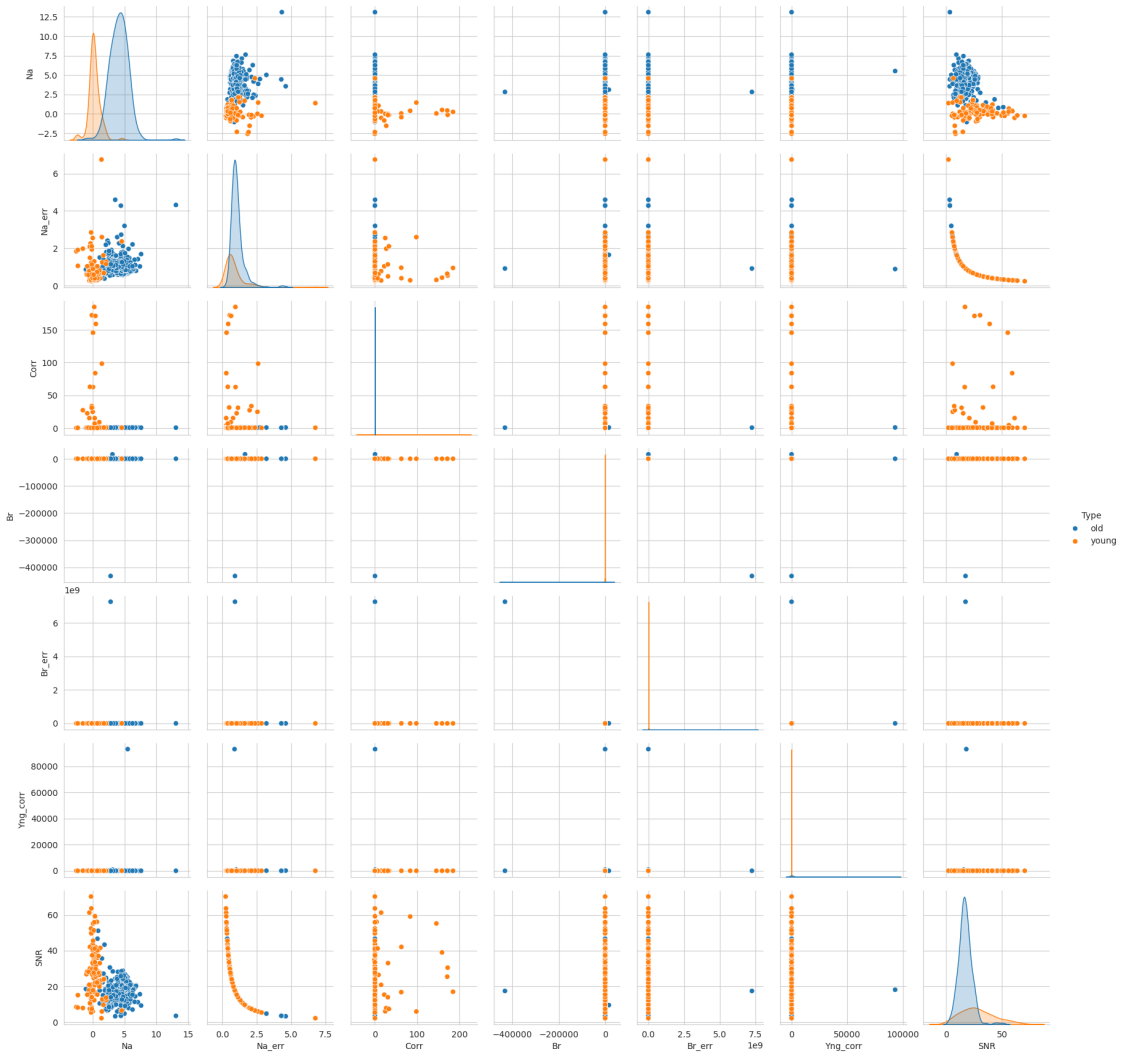
[29]:

	Na	Na_err	Corr	Br	Br_err	\
count	412.000000	412.000000	412.000000	412.000000	4.120000e+02	
mean	3.060000	1.015947	3.942840	-1004.703617	1.760864e+07	
std	2.175442	0.608656	19.959335	21291.863589	3.572741e+08	
min	-2.560000	0.240000	-0.880000	-431830.410000	0.000000e+00	
25%	1.117500	0.680000	0.560000	1.010000	2.675000e-01	
50%	3.455000	0.905000	0.700000	2.040000	4.900000e-01	
75%	4.677500	1.130000	0.830000	3.717500	8.400000e-01	
max	13.100000	6.750000	185.010000	16200.080000	7.251879e+09	

	Yng_corr	SNR
count	412.000000	412.000000
mean	232.722670	20.148592
std	4587.869614	10.533494
min	-2.550000	2.190000
25%	0.300000	14.137500
50%	0.450000	17.790000
75%	0.742500	23.857500
max	93125.150000	70.290000

[30]:
```
# Before splitting the training data into trainig, testing, and validation sets,
# plot clean training with sns.pairplot(clean_training, hue="Type")
sns.pairplot(gcs_clean_train, hue='Type')
```

[30]:
```
<seaborn.axisgrid.PairGrid at 0x780032030b10>
```



[31]:
```
# After cleaning, split data into training, testing, and validation
# We already have a validation split -> gcs_eval it was given as a csv file

# Seed for reproducibility
random_seed = 42

# Split the data into train and test sets with the seed
X_train, X_test, y_train, y_test = model_selection.train_test_split(
    gcs_clean_train.drop('Type', axis=1),
    gcs_clean_train['Type'],
    test_size=0.2,
    random_state=random_seed
)
```

12 Step 5: Select a model and train it

Based on the problem and the data, there are often a handful of algorithms to try. Here, experimentation and knowledge of the strengths and weaknesses of machine learning models will help you choose a model and train it.

12.1 Question 13

(10 pts)

In this lab, we will specifically use the Naive Bayes model.

In the following cells, please implement the Naive Bayes classifier to deterimine whether a star is young or old based on the features you've selected in Step 4.
At the end of this step, report the accuracy of your model on the test data.

[32]:
```
# Build Model
class NaiveBayesClassifier:
    def fit(self, X, y):
        self.classes = np.unique(y)
        self.parameters = {}

        # Calculate parameters for each class
        for c in self.classes:
            X_c = X[y == c]
            self.parameters[c] = {
                'mean': X_c.mean(),
                'var': X_c.var(),
                'prior': len(X_c) / len(X)
            }

    def _calculate_likelihood(self, class_, x):
        # Gaussian likelihood of the data x given a class
        mean = self.parameters[class_]['mean']
        var = self.parameters[class_]['var']

        numerator = np.exp(- (x - mean) ** 2 / (2 * var))
        denominator = np.sqrt(2 * np.pi * var)
        return numerator / denominator

    def _calculate_posterior(self, x):
        posteriors = []

        # Calculate posterior probability for each class
        for class_ in self.classes:
            prior = self.parameters[class_]['prior']
            likelihood = self._calculate_likelihood(class_, x).prod()
            posterior = likelihood * prior
            posteriors.append(posterior)
```

```
        return self.classes[np.argmax(posteriors)]

    def predict(self, X):
        # Predict class for each sample in X
        return [self._calculate_posterior(x) for x in X.to_numpy()]
```

[33]:
```
# Train the model
model = NaiveBayesClassifier()
model.fit(X_train, y_train)

# Make predictions
predictions = model.predict(X_test)
```

[34]:
```
# Uncomment these and run when ready
from sklearn.metrics import accuracy_score
score = accuracy_score(y_test, predictions)
print(f'{score*100:.4f}%')
```

```
27.7108%
```

This accuracy is very poor but we will improve it through tuning.

13 Step 6: Fine tune the model

Most machine learning algorithms will have hyperparameters that can be tuned to improve the performance of the model given the specific dataset. You may also want to visualize the results of model predictions to aid in tuning the model.

13.1 Question 14

(9 pts)

For this lab, please do the following:
1. One of the hyperparameters in this case is the types of features to use. Try subtracting or adding a feature to see how it changes your accuracy.
2. Examine how changes in your training data might affect your results. For example, what if you use only measurements with small uncertainties for training?
3. Report your final accuracy on your test dataset after selecting the optimal model. Explain why you think this is the optimal model.

[35]:
```
# use as many cells as you need to do Step 6. We will be grading manually
```

I have few intuiations about this dataset so I'll attempt to tune by brute force, iterating through some configurations of ways to compose the data. I notice this dataset is not very large so this tactic should be feasible.

[36]:
```
gcs_train = pd.read_csv('galactic_center_stars_training.csv')

# First split: 10% for evaluation, 90% for further splitting into train and test
X_remaining, X_eval, y_remaining, y_eval = model_selection.train_test_split(
    gcs_train.drop('Type', axis=1),
    gcs_train['Type'],
    test_size=0.1,
    random_state=42)
```

[37]:
```
%%time
from itertools import combinations

def evaluate_model(data, feature_combination, z_threshold, results_list,␣
 ↪print_intermediate):
    # Outlier removal based on z_threshold
    if z_threshold is not None:
        data = remove_outliers(data, z_threshold=z_threshold)

    # Split data into features and labels
    X = data.drop('Type', axis=1)
    y = data['Type']

    # Split data into train and test sets
    X_train, X_test, y_train, y_test = model_selection.train_test_split(
        X, y, test_size=0.2, random_state=42)

    # Train and evaluate the model
    model = NaiveBayesClassifier()
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    accuracy = accuracy_score(y_test, predictions)

    # Add the result to the results list
    results_list.append({
        'features_dropped': ', '.join(feature_combination),
        'z_threshold': z_threshold,
        'accuracy': accuracy
    })

    # Optionally print the intermediate result
    if print_intermediate:
        print(f"Evaluating: Features Dropped - {', '.join(feature_combination)}␣
 ↪|| Z-Threshold = {z_threshold} || Accuracy = {accuracy*100:.3f}%")

# Load dataset
gcs_train_data = pd.concat([X_remaining, y_remaining], axis=1)
```

```
# List of all features
all_features = gcs_train_data.drop('Type', axis=1).columns

# List to store the results of each model configuration
results = []

# Flag to control printing of intermediate results
print_intermediate = False

# Generate combinations of features to drop (from 1 to all features)
for r in range(1, len(all_features) + 1):
    for combo in combinations(all_features, r):
        # Create datasets for each combination of dropped features
        data_with_dropped_features = gcs_train_data.drop(list(combo), axis=1)

        # Evaluate model with different z_thresholds and the current␣
 ↪combination of dropped features
        for z_threshold in [2, 3, 4, 5, None]:
            evaluate_model(data_with_dropped_features, combo, z_threshold,␣
 ↪results, print_intermediate)

# Find the top 5 model configurations based on accuracy
top_results = sorted(results, key=lambda x: x['accuracy'], reverse=True)[:1]

# Print the top 5 results
print('------------------------')
print('Top Configurations:')
for result in top_results:
    print(f"Features Dropped - {result['features_dropped']} || Z-Threshold =␣
 ↪{result['z_threshold']} || Accuracy = {result['accuracy']*100:.1f}%")
print('------------------------')
```

```
------------------------
Top Configurations:
Features Dropped - Na_err, Br, Br_err, SNR || Z-Threshold = 3 || Accuracy =
98.6%
------------------------
CPU times: user 58 s, sys: 105 ms, total: 58.1 s
Wall time: 58.1 s
```

[ ]:

14 Step 7: Present your solution and apply it

Here you will evaluate your model against the test set that was set aside in Step 2 to determine the final performance metrics. Your model is now ready to be applied to new data.

14.1 Question 15

(2 pts)

For this lab, please do the following:
1. Report your accuracy for the validation data that you saved in Step 4. How does this score compare to when you were training and fine tuning your model?
2. Apply your model to the data for the unlabeled grading data. Note that you should explore these inputs to see if there are differences in the features between your training data and this dataset. Beware of outliers in this dataset as well. Create a set of predictions called pred_eval.

[38]:
```
# use as many cells as you need to do Step 7.
```

[39]:
```
# Load the evaluation data
gcs_eval_data = pd.concat([X_eval, y_eval], axis=1)

def retrain_and_evaluate(top_model_config, train_data, eval_data):
    # Outlier removal from training data based on z_threshold
    if top_model_config['z_threshold'] is not None:
        train_data = remove_outliers(train_data,␣
 ↪z_threshold=top_model_config['z_threshold'])

    # Drop the specified features from both training and evaluation data
    features_to_drop = top_model_config['features_dropped'].split(', ')
    X_train = train_data.drop(features_to_drop + ['Type'], axis=1)
    y_train = train_data['Type']
    X_eval = eval_data.drop(features_to_drop + ['Type'], axis=1)
    y_eval = eval_data['Type']

    # Train the model
    model = NaiveBayesClassifier()
    model.fit(X_train, y_train)

    # Evaluate on the evaluation set
    predictions = model.predict(X_eval)
    accuracy = accuracy_score(y_eval, predictions)

    return accuracy

# Evaluate the top 5 models on the evaluation set
for i, model_config in enumerate(top_results, 1):
    eval_accuracy = retrain_and_evaluate(model_config, gcs_train_data,␣
 ↪gcs_eval_data)
    print(f"Model {i}: Features Dropped - {model_config['features_dropped']} ||␣
 ↪Z-Threshold = {model_config['z_threshold']} || Evaluation Accuracy =␣
 ↪{eval_accuracy*100:.1f}%")
```

```
Model 1: Features Dropped - Na_err, Br, Br_err, SNR || Z-Threshold = 3 ||
Evaluation Accuracy = 92.9%
```

To find top preforming model I itterated through combinations of features with clipping at various z-thresholds. Increasing beyond a z-threshold of 6 did not change the number of datapoints when compared to a threshold of 5 so 5 was chosen as the max value for z-threshold. One way to further experiment could be by trying different seeds to try different test and training splits.

If you would like to submit your group's solution to the unlabeled dataset for the competition to see who gets the accurate predictions, please send your TA and Prof. Do a csv file with a single column predicting whether each star is 'young' or 'old'. Below is an example code of how to create this file. You will send ONE file per team.