Tutorial 1 – Unsupervised Machine Learning

Janis Witmer	janis.witmer@unibe.ch	office: 210a
Prof. Lucia Kleint:	lucia.kleint@unibe.ch	office: 234
Deadline:	19.11.2025 14:15	

The goal of this exercise is to get familiar with *unsupervised machine learning* techniques introduced in the lecture and apply them to a real dataset containing spectra of stars. You will learn the strengths and weaknesses of different methods, while also exploring their limitations and the challenge of interpreting high-dimensional results. Understanding these boundaries will help you choose appropriate techniques for different types of datasets and scientific questions.

1 Preparing the Data

1.1) Download `tut1_datasetparquet.sec` from Ilias.

In order to read the data, rename it to `tut1_dataset.parquet`. The data set is 139 MB in size and contains 2210 spectra of stars of different spectral classes and luminosity classes. The data are taken from the *Sloan Digital Sky Survey* (SDSS), a large multi-spectral imaging and redshift spectroscopic survey using a dedicated 2.5 m wide-angle optical telescope at the *Apache Point Observatory* in New Mexico (USA). The data are structured as follows:

wavelength	**flux**	**spectral-type**	**luminosity**
`list` of wavelength points in Å	`list` of corresponding fluxes	`str` of spectral type	`str` of luminosity class

Note that the length of the *wavelength* and *flux* `lists` vary between different observations and that the *wavelength* grid is not exactly the same through all observations. This is a problem to apply machine learning, we therefore interpolated all fluxes to a common wavelength grid (the columns common_wavelengths & interpolated_fluxes)

1.2) Load and optionally sort the data with `pandas` and display it. You can use the following lines of code:

```python
import pandas as pd

#load the data
skyserver = pd.read_parquet('tut1_dataset.parquet')

#create dictionary for sorting
spectral_classes = [f"{letter}{number}" for letter in 'OBAFGKM' for number in range(10)]
sort_dict = {s: i for i, s in enumerate(spectral_classes)}

#sort the spectral classes
skyserver = skyserver.sort_values(by = ['spectral_type'], key=lambda x: x.map(sort_dict))

#gather some statistics
skyserver.describe()
```

Note that in order to read the file, it must be in the same directory as your jupyter notebook. Alternatively you can exchange `tut1_dataset.parquet` with the absolute path to the file.

1.3) Find a spectrum for each spectral type (for luminosity class V stars) and plot the original fluxes overlaid with the interpolated fluxes and check if the interpolation worked as expected. Carefully observe the plotted spectra for each spectral type and describe the shape and characteristics of the spectra. Do you notice any trends? In the following exercises the features that we want to work with will be the *interpolated fluxes*.

2 Dimensionality Reduction – PCA

2.1) For the next exercises, we need to make sure that all the features (interpolated fluxes) are on the same scale.

2.1.1) Since we now only work with the interpolated fluxes and it is easier to handle `np.arrays` than `pd.DataFrames`, we suggest extracting the relevant data from the dataframe and convert it to a `np.array`. You should now have a `np.array` of shape `(n_samples, n_features)` = `(2210, 4645)`. You can use the following code:

```python
# extract the relevant data from the dataframe and convert it to a np.array
# assuming skyserver is the dataframe
features = np.vstack(skyserver['interpolated_fluxes'].values)
print(f"the shape of the feature array is {features.shape}")
```

2.1.2) Normalize each spectrum to have a maximum of 1 and a minimum of 0. Ensure that the normalization worked by e.g. checking the minimum and maximum of the first spectrum. Why do we perform this normalization?

```python
#assuming features_normalized is your array after normalization of shape (2210, 4645)
print(normalized_features.min(axis=1), normalized_features.max(axis=1))
```

Output:

```
[0. 0. 0. ... 0. 0. 0.] [1. 1. 1. ... 1. 1. 1.]
```

2.1.3) Would you standardize your features, such that the mean and standard deviation is 0 and 1 for all features respectively? Explain your reasoning.

2.2) Apply *PCA* to your spectra and set `n_components` = 2. Plot your projected data and color it depending on the spectral type and luminosity class. Hint: You first have to convert your *'spectral-types'* and *'luminosity'* to a numerical value, which you can do with the following lines of code:

```python
#convert spectral type to a numerical value
skyserver['spectral_type_code'] = pd.factorize(skyserver['spectral_type'])[0]
# convert luminosity to a numerical value
skyserver['luminosity_code'] = pd.factorize(skyserver['luminosity'])[0]

#plot projected data and color it depending on spectral type
scatter1 = ax1.scatter(projected_data[:, 0], projected_data[:, 1], c=skyserver['
    spectral_type_code'], s=5, cmap='coolwarm')

#Create legend handles for each unique spectral class
unique_classes = skyserver['spectral_type'].unique()
handles = [mpatches.Patch(color=scatter1.cmap(scatter1.norm(skyserver['spectral_type_code
    '][skyserver['spectral_type'] == cls].iloc[0])), label=cls) for cls in unique_classes]

#Add the legend to the plot
ax1.legend(handles=handles, title="Spectral Class", ncol =10, fontsize=6)

#plot projected data and color it depending on luminosity class
scatter2 = ax2.scatter(projected_data[:, 0], projected_data[:, 1], c=skyserver['
    luminosity_code'], s=5, cmap='Accent')

#Create legend handles for each unique luminosity class
unique_classes = skyserver['luminosity'].unique()
handles = [mpatches.Patch(color=scatter2.cmap(scatter2.norm(skyserver['luminosity_code'][
    skyserver['luminosity'] == cls].iloc[0])), label=cls) for cls in unique_classes]

# Add the legend to the plot
ax2.legend(handles=handles, title="Luminosity Class", ncol =5, fontsize=6)
```

2.1.1) Are the different spectral types/luminosity classes well separated? Discuss you observations. Is there some physical explanation for your findings?

2.1.2) Do you think this projection represents your data well? Explain your reasoning.

2.1.3) Would it make sense to add a 3rd component?

3 Clustering

3.1) Before you start this exercise, ensure that your data is normalized as described in 2.1.2).

3.2) Fit multiple `KMeans`-models to your spectra with different numbers of clusters. Play around with the arguments of `KMeans` to optimize your results. Hint: Set the `random_state` parameter to some specific value, to ensure the reproducibility.

- 3.1.1) Plot different metrics to assess the quality of your `KMeans` model as a function of the number of clusters and describe its behavior. Does it look as expected?
- 3.1.2) Based on the plot, what can you tell about your data?
- 3.1.3) How do you find the best number of clusters?
- 3.1.4) Is the best number of cluster unambiguous?

3.3) Fit different `DBSCAN` models to your spectra and vary the hyperparameters to optimize your result.

- 3.2.1) Assess the quality of your `DBSCAN` models by plotting the *silhouette score* as a function of the number of clusters. What is the optimal number of clusters?
- 3.2.2) Is the best number of clusters unambiguous?

3.4) Reinitialize `KMeans` and `DBSCAN` with the best hyperparameters you found in exercises 3.2) and 3.3). Project your spectra into two dimensions using `PCA`. Plot your projected data and color it according to the predicted cluster labels.

- 3.3.1) Compare the plot to the plot made in exercise 2.2). What are the similarities/differences between the clusters found by `KMeans` and `DBSCAN`? Do the clusters have physical meaning?
- 3.3.2) Find a metric that allows you to quantify how well the clustering worked, knowing the true labels. Which clustering method worked better? Try to find an explanation for why the clustering worked well/badly.*
- 3.3.3) How could you artificially force either `KMeans` or `DBSCAN` to perform better?*

4 Outlier Detection*

4.1) For this exercise, we need to make sure that all spectra are normalized such, that the minimum and maximum value of each spectrum is 0 and 1 respectively.

- 4.1.1) Since we now only work with the interpolated fluxes and it is easier to handle `np.array`s than `pd.DataFrame`s, we suggest extracting the relevant data from the dataframe and convert it to a `np.array`. You should now have a `np.array` of shape `(n_samples, n_features)` = `(2210, 4645)`.
- 4.1.2) Before you start this exercise, ensure that your data is normalized as described in 2.1.2).

4.2) Use the *z-score* method introduced in the lecture to identify outlier spectra. Ensure that you are identifying entire outlier spectra, rather than outliers within individual spectra. Plot several examples of these outlier spectra. Analyze and discuss your findings. What physical insights can be inferred from these outliers?

4.3) Repeat exercise 4.2) with *Isolation Forest*. Describe any differences or similarities you observe.

4.4) Repeat the previous two exercises but apply *Local Outlier Factor*. Describe differences and similarities you observe to the previous methods.

*optional exercises – not necessary to reach maximum points.