Introduction to Machine Learning

Hands-on

Brain Tumor Classification using Support Vector Machines

DEPARTAMENTO DE INGENIERÍA TELEMÁTICA Y ELECTRÓNICA

UNIVERSIDAD POLITÉCNICA DE MADRID

MIGUEL CHAVARRÍAS
EDUARDO JUÁREZ
ALBERTO MARTÍN-PÉREZ
ALEJANDRO MARTÍNEZ DE TERNERO

Index

1. Introduction

The aim of this first guided practice is to help the student become familiar with Python and the scikit-learn library by means of a practical example. By the end of this document, you would have implemented a script to detect brain tumoral tissue in hyperspectral images.

a. Key concepts

i. Python: «Python is an **interpreted, object-oriented, high-level programming language with dynamic semantics**. Its high-level built-in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. […] Python supports modules and packages, which encourages program modularity and code reuse. […] **Since there is no compilation step, the edit-test-debug cycle is incredibly fast**. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective» [1].

ii. Scikit-learn: is a free software machine learning library for the Python programming language (https://en.wikipedia.org/wiki/Scikit-learn). It features various classification, regression and clustering algorithms. The project was started in 2007 as a Google Summer of Code project leaded by David Cournapeau.

   URL: https://scikit-learn.org/stable/index.html

   Note: The former link may be also useful to clarify the concepts of Classification, regression, clustering, etc.

iii. Support vector machines (SVM): SVMs are a set of supervised learning methods used for classification, regression and outliers detection. An SVM predicts an optimal hyperplane in an n-dimensional space to divide the training set into multiple classes. Different kernel functions can be specified for the decision function depending on the problem. They could be used to implement a multi-class classification on a dataset, providing in advance a subset of labelled data needed for model training [2].

iv. Cross-validation: is a statistical method of evaluating generalization performance that is more stable and thorough than using a split into a training and a test set. In cross-validation, the data is instead split repeatedly and multiple models are trained. The most commonly used version of cross-validation is k-fold cross-validation, where k is a user-specified number, usually 5 or 10. When performing five-fold cross-validation, the data is first partitioned into five parts of (approximately) equal size, called folds. Next, a sequence of models is trained. The first model is trained using the first fold as the test set, and the remaining folds (2–5) are used as the training set. The model is built using the data in folds 2–5, and then the accuracy is evaluated on fold 1. Then another model is built, this time using fold 2 as the test set and the data in folds 1, 3, 4, and 5 as the training set. This process is repeated using folds 3, 4, and 5 as test sets. For each of these five splits of the data into training and test sets, we compute the accuracy. In the end, we have collected five accuracy values [3]. The process is illustrated in the figure below:

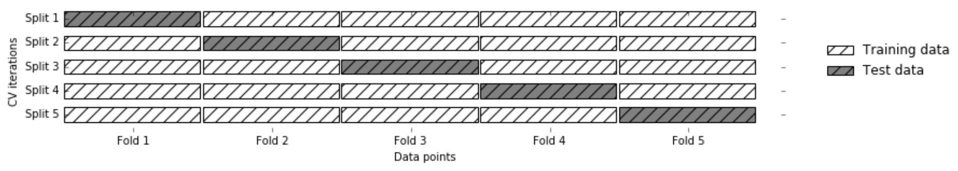


Figure 1: Data splitting in five-fold cross-validation

b. Data Science Workflow

The data scientist practitioners often follow 5 steps which are: data acquisition, data preparation, model analysis, reports and actions. These are described in Figure 1, and as can be seen, expert practitioners do all five steps. For this hands-on we will focus on steps 2 and 3, which are the main tasks people start doing when being introduced to data science. If you want to get into the details of what experts do in each step, we encourage you to visit this reference [8].

We will provide you with data that has been acquired in an operating room of patients suffering from brain tumor. Then, you will start preparing and loading the data for a SVM model to be trained. Later, you will write a Python script that trains a SVM to then classify brain images, generating what is known as brain classification images. Finally, you will optimize a SVM model to evaluate how well the model can classify new patients.

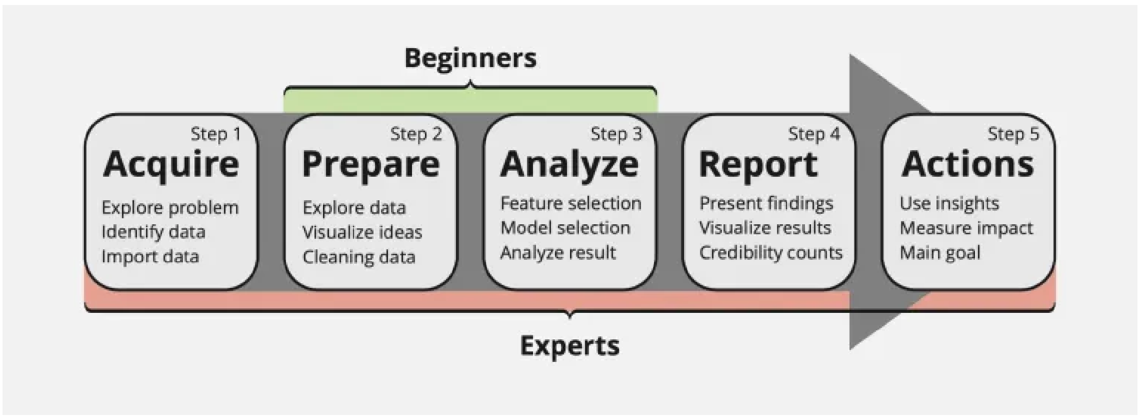


FIGURE 1. DATA SCIENTIST WORKFLOW [8].

c. Brain tumor detection and hyperspectral imaging

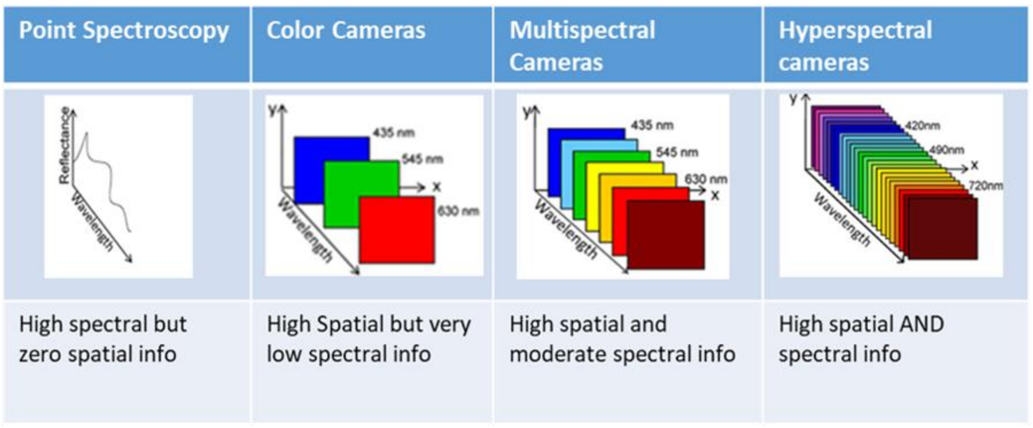
i. The importance of brain tumour detection

Cancer is caused by the transformation of normal cells into tumor cells. They grow uncontrollably, forming masses called tumors that destroy healthy tissue. Today, cancer is one of the most common causes of death. By 2018, this disease had caused 9.6 million deaths, with more than 18.1 million new cases, and this figure is expected to reach 29.5 million patients per year by 2040. In the case of brain tumors, glioma is the most common tumor in adults. Unfortunately, glioblastoma multiforme (GBM) or grade 4 glioma (GIV) does not permit long-term survival among patients since it is the most aggressive glioma type. Despite the risks involved, GBM surgery is an unavoidable phase in the treatment of this type of tumor. Surgical intervention in GBM improves survival and patients' quality of life; nonetheless, it is important to remove as much tumor tissue and as little healthy tissue as possible [2].

To offer a solution to these problems, in recent years, it has been demonstrated that technologies based on hyperspectral imaging (HSI) in combination with machine learning (ML) are able to distinguish between in-vivo tumour and healthy tissue during neurosurgery. HSI is an appropriate technology for medical use since it is non-invasive, non-ionizing and does not require any contact with the patient. In addition to neurosurgery, HSI has been used in different types of image-guided surgery to help the surgeon to identify the lesion and its margins, in cases such as abdominal surgeries, cholecystectomy and renal surgeries. Furthermore, HSI has been applied in other medical applications related to the diagnosis of different kinds of cancer, retinal diseases, diabetes and cardiac diseases, among others. [2]

ii. **Hyperspectral imaging background**

While RGB images only have three discrete wavelengths in the visible spectrum, red, green, and blue, hyperspectral images can gather hundreds of diffuse reflectance wavelengths including the ultraviolet (250-380 nm), visible (380-750 nm) and some of the infrared spectrum (780-2500 nm). Hyperspectral images, also known as hyperspectral cubes, include the spatial dimension represented by the height and width of the image and the depth dimension of all wavelengths gathered. The spectral information of a pixel extracted from the hyperspectral cube is known as the spectral signature of a material or tissue related with the captured pixel. These spectral signatures can be used to differentiate materials in a single hyperspectral image such as healthy brain and tumour tissue.



Point Spectroscopy
Color Cameras
Multispectral Cameras
Hyperspectral cameras
Reflectance
Wavelength
y
x
435 nm
545 nm
630 nm
420nm
490nm
720nm
High spectral but zero spatial info
High Spatial but very low spectral info
High spatial and moderate spectral info
High spatial AND spectral info

Figure 2. Difference between point spectroscopy, colour cameras, multispectral and hyperspectral cameras [9].

2. Previous considerations

The exercises proposed in this document consist of a series of small programs written in Python, intended to get familiar with SVM algorithm. For testing, it is possible to use both

1) Visual Studio IDE, installed in the virtual machine available in the Moodle's course site. The installed package already has Python installed and it can be straightforwardly used,

and

2) Google Colab. Although the use of the previous option is recommended, in case of difficulties with the virtual machine or if you want to make specific tests out of the normal working environment, you can use the Google's online coding tool, Colab [5].

Please note that the Virtual Machine password is: ***imlstudent2022***

3. Brain Tumor Classification using Support vector machines

Previous tips:

- Note that directory separator slashes vary on Windows (\\) to Linux (/). You may have to fix them when copying the code.
- Remember that each time you make a modification in the source code you must save the file before executing it.

a. Previous knowledge on Support Vector Machines

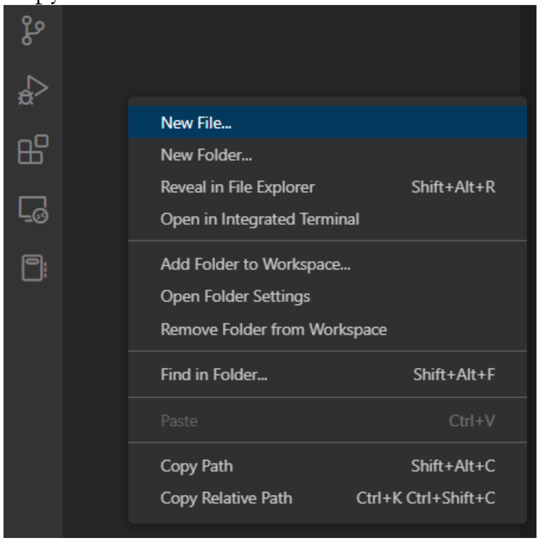
For this exercise we will follow the guidance provided by scikit-learn on the use of SVM. Both the information and the examples are available at the following link, URL: https://scikit-learn.org/stable/modules/svm.html

Subsections 1.4.1 to 1.4.4 on the previous link cover the most basic concepts and examples of SVM and its implementation with Python programming language. Section 1.4.5 contains some recommendations of especial interest when facing the design of SVMs as a solution to problems that require an efficient, fast or specially optimised solution for a limited hardware or a specific working environment. Finally, sections 1.4.6 and 1.4.7 go into more advanced concepts about SVM handling.

It is recommended to read each section carefully, paying attention to the different example codes. Sufficient time should be taken to understand the functionality of the executed code. For troubleshooting, it is recommended to refer, in the first instance, to the troubleshooting section, available below.

b. Loading and understanding data

1. Open Visual Studio Code (or your text editor of preference) to create and open a folder available in your computer (e.g. Desktop/Brain_SVM/).
2. Download the files available in Moodle in the created folder and then unzip it.
3. Create a new Python file in the Brain_SVM folder. To do so, right click > "New file..." and give it the name main.py.


New File...
New Folder...
Reveal in File Explorer Shift+Alt+R
Open in Integrated Terminal
Add Folder to Workspace...
Open Folder Settings
Remove Folder from Workspace
Find in Folder... Shift+Alt+F
Paste Ctrl+V
Copy Path Shift+Alt+C
Copy Relative Path Ctrl+K Ctrl+Shift+C

4. Now you can start with the code itself. First, we will set a seed for the experiments to be repeatable. This way everyone will obtain the same results. Open main.py and add the following seed, later on we will come back to this parameter:

```
seed = 2022
```

5. The first step in every machine learning problem is to load the data and manipulated in such a way that the algorithm can understand it. The data files provided are in *.mat* extension. To load the data stored in the .mat files, we need to use the Scipy [6] Python, specifically the 'loadmat()' function. To do so, you need to import it such as:

```
from scipy.io import loadmat
```

6. To load the data and save it in a variable, you need to pass to 'loadmat()' the path with the file name included as an argument such as:

```
# Load the .mat file to a variable
patient_1_dataset = loadmat(r".\data\dataset\ID0065C01_dataset")
```

7. But… what is really inside the "dataset" variable? To know this, you can print in the command prompt the type of the variable like:

```
# Understand what loadmat returns
print(type(patient_1_dataset))
```

Which will return:

```
<class 'dict'>
```

Therefore, we have a Python dictionary and, if you don't know it, Python dictionaries contains a key associated with every value such that:

```
dictionary = {
    <key>: <value>,
    <key>: <value>,
      .
      .
      .
    <key>: <value>
}
```

Learn more about Python dictionaries in reference [7].

8. Let's see what keys are included in the dataset variable by using the "keys()" method:

```
# See what elements are contained in dataset
print(patient_1_dataset.keys())
```

Which returns:

```
dict_keys(['__header__', '__version__', '__globals__', 'data', 'label',
'label4Classes'])
```

The keys of interest are 'data', 'label' and 'label4Classes'.

Exercise 1: Return the value of the 'data' and 'label' keys in two new variables with names 'data' and 'labels'.
Tip: Use this link to know how to access values in a Python dictionary.

9. The values in 'data' and 'label' correspond to the labelled pixels provided by the neurosurgeons. The 'data' value includes each pixel (or sample) for every row with all their features in the columns. On the other hand, 'label' corresponds to the classes labelled to every single pixel. To see how many pixels we will use to train our SVM model, we need to know the type of the 'data' and 'labels' variables:

```
# What are the data and labels variables?
print(f"'data' type: {type(data)}")
print(f"'labels' type: {type(labels)}")
```

Which returns:

```
'data' type: <class 'numpy.ndarray'>
'labels' type: <class 'numpy.ndarray'>
```

These variables are Numpy arrays, which have the **'shape'** property to see their dimensions. (Numpy is another Python library commonly used for scientific computing. If you want to know more, use this link).
To see the dimensions, use:

```
# See the dimensions of "data" and "labels"
print(f"Samples array size: {data.shape}")
print(f"Labels array size: {labels.shape}")
```

10. Now, instead of only loading a single patient dataset, we will load 3 and merge their pixels and labels together. To do so, we are going to load 2 more patients and merge the 'data' and 'labels' as:

```
# Load datasets from other patients and mix all together
import numpy as np

patient_2_dataset = loadmat(r".\data\dataset\ID0067C01_dataset")
patient_3_dataset = loadmat(r".\data\dataset\ID0070C02_dataset")

data = np.concatenate(
    [
        patient_1_dataset["data"],
        patient_2_dataset["data"],
        patient_3_dataset["data"],
    ],
    axis=0,
)

labels = np.concatenate(
    [
        patient_1_dataset["label"],
        patient_2_dataset["label"],
        patient_3_dataset["label"],
    ],
    axis=0,
)

# See the dimensions of "data" and "labels"
print(f"Samples array size: {data.shape}")
print(f"Labels array size: {labels.shape}")
```

11. Finally, lets understand the different classes we are working with. Since the 'labels' variable is a Numpy array, we can use some functions in Numpy to determine this. Specifically, **'np.unique()'** and the built-in Python function **'len()'.** This way we can see the number of labelled pixels for every class as well as the number of different classes we are working with by using:

```
# How many labels do we have?
print(f"Unique labels: {np.unique(labels, return_counts=True)}")
print(f"Different number of labels: {len(np.unique(labels))}")
```

Which returns:

```
Unique  labels:  (array([101,  200,  301,  302,  320],  dtype=uint16),
array([5896,  2387,  350,  339, 2399], dtype=int64))
Different number of labels: 5
```

(Note that the values might differ from yours since the data used might be slightly different.)

c. Train and evaluate an SVM model

Now we will train a simple SVM model on the brain labelled pixels and determine how well it performs when predicting data.

1. To train an SVM, we need to import the 'SVC' class from 'scikit-learn' (or sklearn), create an instance, and train (or fit) the model. Do you remember the 'seed' we set at the beginning of the hands-on? Now it's time to pass it to the SVC class constructor. Additionally, we will set the 'probability' parameter to True.

   Exercise 2: Why do we set the 'probability' parameter to true?
   Tip: Read the SVM documentation in scikit-learn

   ```python
   from sklearn.svm import SVC

   # Create an instance of the model
   model = SVC(kernel="linear", probability=True, random_state=seed)
   # Train the model with the dataset
   model.fit(X=data, y=labels)
   ```

   As you can see, we have encountered a warning message:

   ```
   DataConversionWarning: A column-vector y was passed when a 1d array was
   expected. Please change the shape of y to (n_samples, ), for example
   using ravel().
   ```

   This type of issues is common when first working with data. Some messages are more descriptive than others, but luckily the one presented is very easy to solve. The problem is due to the dimensions of the 'labels' Numpy array passed to the 'y' parameter of the 'fit()' method. If you remember, the shape of 'labels' is (N, 1) but the message suggests to change the shape to (N,). To do so, we just need to use the 'ravel()' method to 'labels' as:

   ```python
   # To change the shape of labels from (X, 1) to (X, ), we do:
   labels = labels.ravel()
   print(f"New shape of labels array: {labels.shape}")

   # Then we can fit the model without any warning
   model.fit(X=data, y=labels)
   ```

2. To predict data with the trained SVM model, we can use two methods "predict()" and "predict_proba()". In this case, we will indicate the model to predict **the same data used**

during the training step *(note that this is never done in real scenarios since you always want to predict new data unseen by the model. Later we will learn how to properly evaluate a model!):*
As you can see, the difference between the results provided by methods "predict()" and "predict_proba()" methods are the shape of the arrays. On one hand, "predict()" provides a single predicted class (either healthy, tumor, blood or meninges tissue), whereas "predict_proba()" gives the probability estimated for each of the classes.

3. Now we need to evaluate how well a model has predicted the data. To do so, **we will always need the reference (true) labels to be compared with the predicted labels.** Otherwise, we won't be able to compute certain metrics. For simplicity purposes, we will evaluate the model using the accuracy score (but there are other metrics that will help evaluate the model better, such as the ROC AUC score):

```
from sklearn.metrics import accuracy_score

# Compute the accuracy of the predictions
acc = accuracy_score(y_true=labels, y_pred=predictions)
print(f"ACCURACY: {100*acc:.2f}%")
```

Exercise 3: Use the same trained SVM model but predict data **not used** during training (patient ID0071C02). Evaluate how well the model has performed. Does the accuracy increase or decrease? Why?

Tip: Read the "a. Loading and understanding data" section to load new data and save it with its classes .

d. Generate visualizations

It is a common practice to generate useful visualizations to subjectively evaluate the ML model. For instance, in the computer vision area one can obtain classification maps to check how the model classifies an image. This helps better understand the behaviour of the model since you not only focus on the objective score obtained (such as de accuracy previously obtained). Therefore, we are now going to classify a brain with the model to obtain its classification map. Will we be able to classify the tumour in the brain image? Let's find out!

1. First, load a hyperspectral cube and obtain the necessary data to be classified. In this case we will classify the brain image with "ID0071C02"

```
# Load entire hyperspectral cube
patient_id = "ID0071C02"
preprocessed_mat                                                        =
loadmat(rf".\data\cubes\SNAPimages{patient_id}_cropped_Pre-
processed.mat")
cube = preprocessed_mat["preProcessedImage"]
```

2. To predict the 3D cube using the "predict_proba()" method, we need to reshape it into a 2D matrix. The method needs an input array of shape (n_samples, n_features) but we have a 3D array of shape (width, height, bands). Reshaping the 3D cube to then predicted can be accomplished by using:

```
# Predict method needs an input array of shape (n_samples  n_features)
# but we have (width, height, bands). Therefore, we need to reshape
the
# 3D cube into a 2D cube and then predict
cube_reshaped  =  cube.reshape((cube.shape[0]  *  cube.shape[1]),
cube.shape[2])
pred_map = model.predict_proba(X=cube_reshaped)
```

3. Now, we will generate a classification map for us to see how the pixels have been labelled. Using the provided functions in the given files and the code below, you will be able to save a classification map with and without probabilities:

```
# Generate a classification map
from classification_maps import ClassificationMap

cls_map = ClassificationMap(
    map=pred_map,
    cube_shape=cube.shape,
    unique_labels=np.unique(labels),
)

# * Save the classification map
cls_map.plot(
    title=f"Patient classified with {type(model).__name__}",
    show_axis=False,
    path_="./outputs/",
    file_suffix=f"{patient_id}",
    file_format="png",
)
```

4. In order to compare how well we have classified the pixels, we can use the ground-truth map with the pixels labelled by a neurosurgeon at the hospital. To save the ground-truth map as an image to be visualized we can use:

```
# Generate the ground-truth map image
from ground_truth_maps import GroundTruthMap

gt = GroundTruthMap(r".\data\ground-truth\\", patient_id)

# Save the ground truth map
gt.plot(
    title=f"Ground truth from patient {patient_id}",
    show_axis=False,
    path_="./outputs/",
    file_suffix=f"GT_{patient_id}",
    file_format="png",
)
```

Exercise 4: Compare the classification maps with the ground-truth map. Has the model classified the pixels properly? Can you think what happened and why?

e. Optimize a ML model by tuning its hyperparameters

Although incrementing the data certainly helps a ML to perform better, it is also common practice to tune or optimize the hyperparameters of the model. The hyperparameter and the parameters of a model are different things. The parameters are **variables that are changed during the training process**. But the hyperparameters are **variables that have a fixed value and guide the model during the training process.** Therefore, we can only select the values we want on the **hyperparameters**. By tuning or optimizing the hyperparameters we can boost the model to perform better. Let's find out how to do that with Scikit-learn!

1. To optimize our SVM model we are going to use the GridSearchCV class from Scikit-learn. But before, we must define the hyperparameters we want to optimize and the values we want to try. The hyperparameters for SVM can be found in its documentation page at Scikit-learn, but the ones we will optimize are "kernel", "C" and "gamma":

```
from sklearn.model_selection import GridSearchCV

hyperparameters = {"kernel": ("linear", "rbf"), "C": [1], 
"gamma":[1]}

model = GridSearchCV(estimator=SVC(probability=True, 
random_state=seed), param_grid=hyperparameters, verbose=4)
```

2. Now we are ready to find the best hyperparameters and train the SVM with the best ones. To so, we will use the code below, which will conduct a 5-fold cross-validation to train multiple SVMs with slightly different data (learn more about cross-validation in this Scikit-learn documentation page). During the execution you will see the hyperparameters used, the score obtained as well as the time took for the SVM to be trained. **This may take few minutes!**

```
# Perform 5-fold cross validation to obtain the best parameters
# With those, automatically train the model with the same data passed
model.fit(X=data, y=labels)
print(f"Best parameters found: {model.best_params_}")
```

3. As done before, we will now predict the data from the patient not used during training (ID0071C02). To see how well it classifies the image and its data:

```
# Predict data from the patient not used during training (as done 
before)
new_predictions = model.predict(patient_new_dataset["data"])

acc                                                              = 
accuracy_score(y_true=patient_new_dataset["label"],y_pred=new_predi
ctions)
print(f"ACCURACY (on new data with optimized SVM): {100*acc:.2f}%")

# Classify image with optimized SVM
pred_map = model.predict_proba(X=cube_reshaped)

cls_map = ClassificationMap(
    map=pred_map,
    cube_shape=cube.shape,
    unique_labels=np.unique(labels),
)

# Save the classification map
cls_map.plot(
             title=f"Patient     classified     with     optimized
{type(model.estimator).__name__}",
    show_axis=False,
    path_="./outputs/",
    file_suffix=f"{patient_id}_optimized",
    file_format="png",
)
```

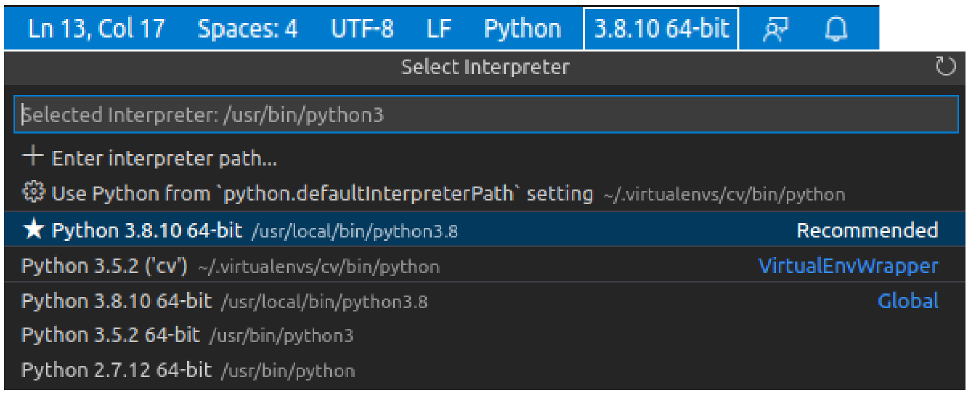
Exercise 5: Has the accuracy increased from the non-optimized model? Can you optimize furthermore the model?

4. Troubleshooting

The most common errors are listed below, and the corresponding suggested workarounds are proposed.

Visual Studio Code says ImportError: No module named 'sklearn' or 'ImportError: cannot import name 'DecisionBoundaryDisplay'

- To solve this issue, you need to ensure that the Python version used to run the program have the 'sklearn' package installed. We recommend to use the already installed Python 3.8.10 version. To do so, make sure that in the bottom right corner of Visual Studio Code the version used is 3.8.10 64-bit. Changing the version is as simply as clicking in the version and then select the indicated interpreter:


Ln 13, Col 17
Spaces: 4
UTF-8
LF
Python
3.8.10 64-bit
Select Interpreter
Selected Interpreter: /usr/bin/python3
+ Enter interpreter path...
Use Python from `python.defaultInterpreterPath` setting ~/.virtualenvs/cv/bin/python
★ Python 3.8.10 64-bit /usr/local/bin/python3.8 Recommended
Python 3.5.2 ('cv') ~/.virtualenvs/cv/bin/python VirtualEnvWrapper
Python 3.8.10 64-bit /usr/local/bin/python3.8 Global
Python 3.5.2 64-bit /usr/bin/python3
Python 2.7.12 64-bit /usr/bin/python

I can't show a figure when running an experiment.

- If you are using the Python 3.8.10 64-bit Python interpreter probably you encounter the "*UserWarning: Matplotlib is currently using agg, which is a non-GUI backend, so cannot show the figure.*" error. If so, this has to do with the line of code "**plt.plot()**". This problem has to do with not having a GUI backend that allows the Matplotlib python package to display a figure. The easiest way to solve this problem is saving the figure instead of plotting it. To do so, just replace the "plt.show()" function call with "**plt.savefig('name_of_my_figure.png')**" call. For more information regarding saving figures with Matplotlib, you can visit the documentation using this link.

5. References

[1] Python, available in URL: https://www.python.org/doc/essays/blurb/

[2] Urbanos, Gemma, Alberto Martín, Guillermo Vázquez, Marta Villanueva, Manuel Villa, Luis Jimenez-Roldan, Miguel Chavarrías, Alfonso Lagares, Eduardo Juárez, and César Sanz. 2021. "Supervised Machine Learning Methods and Hyperspectral Imaging Techniques Jointly Applied for Brain Cancer Classification" *Sensors* 21, no. 11: 3827. https://doi.org/10.3390/s21113827

[3] Müller, Andreas C., and Sarah Guido. Introduction to machine learning with Python: a guide for data scientists. " O'Reilly Media, Inc.", 2016.

[4] SVM implementation with scikit-learn, available in URL: https://scikit-learn.org/stable/modules/svm.html

[5] Google Colab, available in URL: https://colab.research.google.com/

[6] Scipy, available in URL: https://scipy.org/

[7] Python dictionaries, available in URL: https://realpython.com/python-dicts/

[8] The Ultimate Data Science Workflow Template, available in URL: https://www.learnpythonwithrune.org/the-ultimate-data-science-workflow-template/

[9] Spectral Imaging, available in URL: https://www.spectricon.com/spectral-imaging/