Using Machine Learning for Functional Group Identification in Infrared Spectroscopy: Handout and Protocol

Summary:

In this experiment, you will learn how to use machine learning (ML) algorithms to analyze spectroscopic data. You will be provided with a dataset of computational infrared (IR) absorption spectra. First, you will train and test machine learning models to perform a binary classification task, in which you determine which spectra correspond to molecules containing a carbonyl functional group. You will explore different machine learning algorithms and will determine how changes in the data processing steps and the machine learning hyperparameters affect the results. Next, you will perform a multiclass classification task, in which you will train and test machine learning models to distinguish between the spectra of different types of carbonyl-containing molecules: ketones, carboxylic acids, and molecules with other carbonyl groups. This lab will also introduce you to the basics of writing and executing Python code in a Google Colaboratory notebook.

Theoretical background:

Vibrational Spectroscopy:

Molecular vibrations are motions of the relative interatomic distances that do not change the molecular center of mass. For a polyatomic molecule, these complex vibrational motions can be reduced to a set of *normal modes*. The energy of these normal modes is quantized and depends on the masses of the atoms and the strengths of the bonds involved in the vibration. Molecules can undergo changes in vibrational state upon interaction with photons, meaning that spectroscopy can be used to probe molecular vibrations. In particular, *infrared (IR) absorption spectroscopy* quantifies the absorption of light at different frequencies upon molecular excitation to a higher vibrational state.

Because the vibrations of a molecule depend on its structure, molecular vibrational spectra are unique. In practice, however, the vibrational modes corresponding to particular *functional groups*, specific arrangements of atoms and bonds, typically appear at similar frequencies across different molecules. Thus the presence of characteristic peaks in a vibrational spectrum can indicate the presence of a particular functional group. This experiment focuses on the characteristic vibrational features of the *carbonyl* functional group, in which a carbon atom is connected to an oxygen atom by a double bond. Carbonyl groups are a building block of several other functional groups, including *ketones* and *carboxylic acids*. The *carbonyl stretch* vibrational mode gives rise to a strong IR absorption peak that usually lies between 1660 – 1770 cm^{-1}, with the exact position depending on the type of carbonyl and other aspects of the molecular structure.

In addition to this carbonyl stretch, some carbonyl-containing functional groups contain other characteristic peaks; for example, the *hydroxyl stretch* in carboxylic acids gives rise to a broad peak in the range of 2500 – 3000 cm^{-1}. In this experiment, IR absorption spectra are used to distinguish between molecules with and without a carbonyl group and further to distinguish between ketones, carboxylic acids, and other carbonyl-containing molecules.

Machine Learning:

The term *machine learning* refers to a broad class of computer algorithms that can "learn" from data using statistical tools. Machine learning is widely used in applications ranging from image analysis to language translation to medical diagnosis. Within chemistry, machine learning has found applications in spectroscopy, materials discovery and property prediction, calculation of potential energy surfaces and force fields, and microscopy, among other areas.

Because machine learning models "learn" from data, they must first be provided with *training data* before they can be used for the analysis of new data. Sufficient training data must be provided to allow the model to generalize to new data. Machine learning algorithms are often categorized as *supervised* or *unsupervised*. In the supervised learning paradigm, training data consists of observations (*i.e.*, instances) and the desired outcome (*i.e.*, label) for each instance. The goal is to develop a model that can correctly predict the label for new data. In contrast to supervised learning, unsupervised learning entails discovering the data's underlying structure without reference to target labels; thus these approaches can be used for training data that does not contain labels, or for which the appropriate labels are not known. Supervised learning algorithms can be further categorized into *regression* and *classification* models. A regression model predicts numeric target values, whereas a classification model classifies data into distinct groups.

Classification tasks are described as *binary classification* if the learning goal is to assign the data into two distinct groups or *multiclass classification* if the data are to be assigned into three or more distinct groups. One common strategy for multiclass classification tasks is to carry out multiple *one-vs-all* binary classifications, in which the probability of membership in each class is separately determined for each instance. The class with the highest probability for each instance is then taken as the final predicted label.

There are a number of different classification algorithms; this experiment uses four common ones. The *Decision Tree* algorithm models the data in a tree structure, in which each leaf node corresponds to a class label and attributes are the tree's internal nodes. Better performance is generally obtained with the *Random Forest* algorithm, which builds a collection of decision trees, where each tree is trained with a random subset of training instances and a random subset of attributes. By pooling predictions from multiple decision trees, Random Forest aims to reduce the variance of each individual tree and thereby achieve a more robust and superior performance. An alternative but often effective approach is the *k-Nearest Neighbors* (k-NN) algorithm. Instead of training a model, this algorithm groups the training data according to a similarity metric. A new instance is then classified by taking the most common label among the *k*

nearest neighbors (*i.e.*, the *k* most similar members of the training data set), where *k* is an adjustable parameter. Finally, the *Naive Bayes* classification algorithm generates a probability model of the data, derived from Bayes' theorem. It is called "naive" because it assumes that the data features are independent. Although this assumption is often violated, the model can still perform well.

One common issue in classification tasks is *class imbalance*, meaning that the different classes are not equally represented in the training data. Class imbalance can sometimes lead to poor classification performance because the model may simply learn to ignore the less common (or *minority*) class. Several approaches are available to account for class imbalance in the training data. This experiment uses a technique called *synthetic minority oversampling technique* (*SMOTE*) which generates new instances of the minority class by interpolating between the existing instances. Another important step in classification tasks, and other machine learning analyses, is *data preprocessing*. This term refers to the organization and standardization of any raw data that will be used to train or test the machine learning model. In this experiment, the raw vibrational spectra must first be put into a uniform format before any analysis can be performed; this process is described in the protocol.

After developing a machine learning binary classification model, several metrics can be used to evaluate the model performance. *Error cases* are instances for which the model's prediction was incorrect. A *false positive* (FP) is an outcome where the model incorrectly predicts the positive class. A *false negative* (FN) is an outcome where the model incorrectly predicts the negative class. An overall performance metric is the *accuracy*, defined as the proportion of the total number of predictions that were correct (*i.e.*, the number of correct predictions divided by the total number of instances in the test data). The ability to identify members of the class is quantified by the sensitivity, defined as the proportion of actual positive cases which are correctly identified (*i.e.*, the number of correct positive predictions divided by the number of positive instances in the test data). The corresponding metric is the specificity, defined as the proportion of actual negative cases which are correctly identified (*i.e.*, the number of correct negative predictions divided by the number of negative instances in the test data).

Another brief introduction to machine learning can be found here: https://chem.libretexts.org/Courses/Intercollegiate_Courses/Cheminformatics_OLCC_(2019)/8%3A_Machine-learning_Basics/8.1%3A_Machine_Learning_Basics

Dataset and Implementation:

Several comprehensive databases of experimental and computational vibrational spectra are available. This experiment uses the Alexandria Library, a computational dataset containing a range of molecular properties calculated for 2,704 different compounds using different model chemistries (https://www.nature.com/articles/sdata201862). The particular dataset used in this experiment contains the vibrational frequencies of 2,337 molecules calculated using density functional theory (DFT) with the B3LYP hybrid functional and the aug-cc-pVTZ basis set, designated as B3LYP-aug-cc-pVTZ. The calculated vibrational frequencies are a series of lines

with intensities corresponding to the oscillator strength of each vibrational mode. To simulate IR absorption spectra, these vibrational frequencies must be convoluted with a function representing the peak shape; this experiment uses a Lorentzian function with width equal to 40 cm^{-1}. Of the resulting spectra, 90% (2,104 spectra) are used as training data and 10% (233 spectra) are used as test data. For the multiclass classification task in Part III of this experiment, the total dataset contains the spectra of 351 carbonyl-containing molecules, of which 90% (316 spectra) are used as training data and 10% (35 spectra) are used as test data. (We have done this for you already, so that you can focus on the machine learning aspects of the project.)

This experiment is implemented in the programming language *Python*, which is a popular general-purpose programming language. The experiment will walk you through some of the basic features of Python; if you already know another programming language, many of these features will be familiar to you already. Python code can be implemented in a convenient *Jupyter notebook* format, which can include code, text, graphics, and hyperlinks. These notebooks can be run on *JupyterHub* or on *Google Colaboratory*.

Protocol:

Overview:

The protocol is divided into three sections:

I. *Part I* walks you through the "Carbonyl Binary Classification" notebook, in which you will carry out a binary classification analysis to distinguish between molecules with or without a carbonyl group using two different machine learning models. All the code that you will need for this section is provided in the notebook and you simply need to execute the notebook in order. As you proceed through the notebook, familiarize yourself with the analysis workflow and the basics of the Python code.

II. *Part II* asks you to explore the binary classification analysis in more detail by testing different machine learning models, changing analysis parameters, and analyzing different test data. To carry out these tasks, you will need to make modifications and add small sections of your own code to the "Carbonyl Binary Classification" notebook.

III. *Part III* walks you through the steps needed to implement a multiclass classification analysis to distinguish between three different classes of molecules: ketones, carboxylic acids, and other carbonyl-containing molecules. This part of the protocol uses the code framework provided in the "Carbonyl Multiclass Classification" notebook. You will need to write larger chunks of code in this part of the protocol, although everything should be an extension of the code from the binary classification notebook.

Part I: Implementing Carbonyl Binary Classification

1. Save a copy of the "Carbonyl_Binary_Classification.ipynb" notebook and open it in Google Colaboratory. You can do this by uploading the file to Google Drive and then opening it in Google Colaboratory.
2. Run the "Getting Started" section to load the necessary Python code libraries
3. Run the "Get Data" section to import CSV files containing vibrational spectra from the GitHub repository:
    a. This code imports two different sets of data:
        i. Training data (train): dataset used to train the machine learning models
        ii. Test data (test): the primary test dataset used to test the machine learning models
    b. Also, notice that for each spectrum, there is a label designating whether a carbonyl group is present (1) or absent (0)
4. Run the "Data Preprocessing" section to format the training and test data for the machine learning analysis:
    a. The "Normalization" module normalizes each spectrum so that the minimum intensity is 0 and the maximum intensity is 1
    b. The "Apply Threshold" module sets intensity values below a certain threshold to 0
        i. Use the default threshold of 0.2 to start
        ii. Look at the plotted spectra and think about whether or not this threshold seems like a reasonable choice
    c. The "Split Attribute and Label" module separates the label indicating whether a carbonyl group is present or absent from the intensities for each molecule
    d. The "Data Balancing" module balances the data using the Synthetic Minority Over-sampling Technique (SMOTE)
        i. Look at the plotted spectra and observe how the synthetic carbonyl-containing spectrum compares to the real carbonyl-containing spectrum. Would you guess that the synthetic spectrum was a real carbonyl-containing molecule?
5. Run the "Building Machine Learning Models" section to train two machine learning models, a Decision Tree model and a Random Forest model, using the training data set
6. Run the "Testing Machine Learning Models" section to test the Decision Tree and Random Forest model on the test data and display the performance results
    a. How good is the performance of the two models? Compare the accuracy (total fraction of correct predictions), sensitivity (correct identification of carbonyl-containing compounds), and specificity (correct identification of compounds without carbonyls).
7. Run the "FP/FN Group Analysis" section to examine the error cases, the false positives (molecules for which the model incorrectly predicted the presence of a carbonyl group)

and the false negatives (molecules for which the model incorrectly predicted the absence of a carbonyl group)

   a. First look at the results for the Decision Tree model. Examine the molecular structures and spectra for a few of the false negative and false positive results. Can you identify any features that might explain the errors?
   b. Now repeat the analysis for the Random Forest model. Are there any false positives or false negatives in common for the two models?

8. Run the "Assessing Overall Model Performance" section to display a table with the different performance metrics for the Decision Tree and Random Forest models

<u>Part II: Exploring Carbonyl Binary Classification</u>

9. Now develop a new code module to train and test a new machine learning model using the Gaussian Naive Bayes algorithm:
   a. The easiest way to do this is to make a copy of the code used for the Decision Tree or Random Forest training and testing and modify the variable names and functions
   b. Instead of the `RandomForestClassifier()` function, use the `GaussianNB(var_smoothing= 0.03511)` function
      i. The value of the `var_smoothing` parameter affects the performance of the model. Optional: experiment with different values to see how the performance changes.
   c. How does the performance of the Gaussian Naive Bayes model compare to that of the Decision Tree and Random Forest models? (It might help to add the Gaussian Naive Bayes performance metrics to the summary table.) Can you think of reasons why this might be the case?
   d. Inspect the false negative and false positive cases for this model. How do they compare to the Decision Tree and Random Forest models?
10. Another widely used classifier is the k-nearest neighbors (k-NN) algorithm. In brief, it searches for the *k* members of the training set that are closest, by some metric, to the test data. It then classifies the test data based on the classes of those nearest neighbors. Write a new code module to implement the k-NN algorithm for the binary carbonyl classification task.
    a. First use the `KNeighborsClassifier(n_neighbors= 5)` function to implement this algorithm. Here the number of neighbors is set to the default value of 5. How does the performance compare to the other models?
    b. Next, test different values of the `n_neighbors` parameter. How sensitive is the performance to this value? What happens to the performance for very large values of `n_neighbors` (*e.g.*, > 100). Why do you think that this is the case?
    c. Inspect the false negative and false positive cases for this model. How do they compare to the other models?

11. Recall that as part of the data preprocessing, a thresholding step was used to set low intensity values to 0. Increasing the value of this threshold means that less intense spectral features will not be used in the classification. Repeat the training and testing using the Random Forest model with the threshold set to 0.5. (Optional: test one or two other values, too.)
    a. How do the results change? What does this result tell you about the importance that the model places on less-intense spectral features? Is this consistent with the way that you might analyze a spectrum yourself?
12. Data preprocessing is an important step in machine learning; all test data must conform to the format expected by the trained model. To see this, try analyzing a "bad" test data in which the data format is not consistent.
    a. Load this data set from the GitHub repository: https://raw.githubusercontent.com/elizabeththrall/MLforPChem/main/MLforvibspectroscopy/Data/binary_test_baddata.csv
    b. This test data contains four spectra for the carbonyl-containing molecule N-methylacetamide. The first one is the spectrum in the correct format; the other three are formatted incorrectly.
    c. Remember that before running the analysis, you will need to go through the same data preprocessing steps as for the first test data set (normalization, thresholding, and splitting attribute and label).
    d. Use the Random Forest model to analyze these "bad" test data. Can the model correctly predict that they are carbonyl-containing spectra?
    a. Plot the different spectra, either in the notebook or by downloading the data from GitHub and plotting in another plotting program. Can you identify how the three incorrectly formatted spectra different from the format expected by the model?
13. Optional: In a classification task, if the data are not balanced between the different classes (here, carbonyl-containing vs. non-carbonyl-containing molecules), the model may simply learn to ignore the minority class. If the classes are distinct enough, however, data balancing may not be necessary. Try running the Random Forest classification analysis again without using SMOTE to balance the training data.
    a. Compare these results to the results obtained after data balancing. How much does data balancing affect the performance? What do you conclude from these results?
14. Optional: Your trained machine learning models can be used to classify spectra other than the ones provided in this experiment. Try using the Random Forest model to classify another spectrum obtained experimentally or computationally.
    a. Notes:
        i. Your new test data should not include molecules contained in the training dataset, because the model has seen those spectra already, so you must

check to be sure that all molecules are unique. To check, download the training data CSV file and search the list of molecules by SMILES string.

ii. The new test spectra must be processed to be in the same format as the dataset used to train the model. Inspect the format of the training data displayed in the "Get Data" code module. Your test data must match the training set data in the following ways: reported as absorbance or intensity instead of transmission, spanning the same frequency range, and using the same frequency spacing of data points. If the test spectra do not match this format, you must modify them manually before running the analysis. (For example, you can use interpolation to change the data point spacing.)

iii. Computational vibrational frequencies are systematically higher than experimental values due to approximations made to simplify the calculations. To address this discrepancy, computational frequencies are typically scaled for comparison to experimental data. Consensus scaling factors for different model chemistries are available from the NIST (https://cccbdb.nist.gov/vibscalejust.asp). For the B3LYP-aug-cc-pVTZ data used here, the consensus scaling factor is 0.968. Thus experimental frequencies must be divided by this value before data preprocessing. Further, computational frequencies obtained with a different model chemistry must be scaled by the ratio of the two scaling factors.

iv. Before loading the data into Google Colaboratory, you will need to save it in a CSV file with the same format as the test data, including the header row, the row number, the SMILES string, the molecule name, and the label. To get a template, follow the link to the test data CSV file in the GitHub repository and save it to your computer by right-clicking and selecting "Save Page As…"

v. To load your data file into Google Colaboratory, go to the "Files" menu by clicking on the folder icon at the top left of the page, then click on the page with arrow icon to select and upload the file. You can then read the file using the `pd.read_csv` command and the file name (without a file path or URL).

vi. Remember that before running the analysis, you will need to go through the same data preprocessing steps as for the first test data set (normalization, thresholding, and splitting attribute and label).

b. Choose a molecule not represented in the test or training dataset and test the model performance:

    a. Obtain a new computational or experimental spectrum for the molecule:

        i. Computational: Use the Psi4 computational engine in WebMO to do a geometry optimization and vibrational frequencies calculation ("Optimize and Frequencies") using a basis set of your choice.

ii. Experimental: Find experimental IR spectral data in the NIST Chemistry WebBook (https://webbook.nist.gov/chemistry/).

b. Download the spectrum and process the data to get the correct format as described above before loading it into Google Colaboratory.

c. Use the Random Forest model to determine whether or not your molecule contains a carbonyl. Was the prediction correct? If not, double check the test data format. If the test data format is correct, can you identify any spectral features that might explain the error?

Part III: Carbonyl Multiclass Classification

15. Save a copy of the "Carbonyl_Multiclass_Classification.ipynb" notebook and open it in Google Colaboratory
16. The code will walk you through the steps needed to carry out the multiclass classification analysis:
    a. Note that you will set up this classification as a series of three binary classifications. There are three copies of the same training data with different labels corresponding to whether or not each molecule contains a ketone, a carboxylic acid, or another carbonyl group. You will train three different Random Forest machine learning models—one for classifying ketones, one for classifying carboxylic acids, and one for classifying other carbonyl groups. You will then apply each of these models to the test data. For each model, you will get a prediction (positive or negative) and a probability (essentially how much confidence the model has in the prediction). The final label will be the predicted label with the highest probability. (If this approach sounds complicated, don't worry—the notebook will carry out the probability analysis for you.)
    b. How does the performance for the multiclass classifier compare to the binary classifier? Does the performance vary for the three different classes?
    c. Plot the spectra of several ketones and carboxylic acids. What spectral features do you think might be used to distinguish between the two types of molecules?
17. Optional: Try something different! Try different analysis parameters, a different machine learning model, or a different test data spectrum.

Lab Report:

Binary Classification:

- Show a table with the performance metrics for all tested models with all tested parameters.
- Discuss the performance of the four different types of machine learning models:

    - Which one achieved the best performance? Which one achieved the worst performance?
    - Were there any outliers (*i.e.*, any models that performed much better or worse than the others)? If there were, can you think of any explanation for why that might be the case?
    - For the k-nearest neighbors model, discuss how the number of nearest neighbors *k* affects performance. What did you observe for a large (*e.g.*, > 100) value of *k*? Can you explain this observation?
- Discuss the false positives and false negatives in the analysis:
    - Plot representative spectra of at least one false positive and one false negative from the Decision Tree analysis; also show the corresponding molecular structures. Can you observe any spectral and/or structural features that might explain the errors?
    - Were there any common false positives and false negatives between the different models? If so, can you rationalize why these molecules might be hard to classify?
- Discuss the effect of the threshold value on the performance of the Random Forest model:
    - How was the model performance affected by using the high threshold value of 0.5 instead of the default value of 0.2? Plot the spectra of a few representative molecules to show what they looked like after thresholding with those two values?
    - What does this result tell you about the importance that the model places on less-intense spectral features? Is this consistent with the way that you might analyze a spectrum yourself?
    - If you tested other threshold values, discuss how they affected the performance.
- Discuss the performance of the model for the "bad" data set:
    - Was it able to identify the presence of a carbonyl group for the correctly formatted spectrum #1? What about for the incorrectly formatted spectra #2 – 4? If not, can you identify the problems with the format of these spectra?
    - Discuss what these results mean for the importance of data preprocessing in machine learning analysis.
- Optional: If you ran the analysis without using SMOTE to balance the training set data, how did the performance change? On the basis of these results, do you think that data balancing is necessary for this particular classification task? Explain why or why not.
- Optional: If you tested the analysis on a different experimental or computational spectrum, plot the spectrum and show the molecular structure. Was the model able to correctly identify whether or not a carbonyl group was present? If not, can you identify any reasons why that might be the case?

Multiclass Classification:

- Show a table with all the different classification outcomes (both correct and incorrect predictions). Also show a summary table with the overall classification accuracy and the accuracy for the three different classes:
    - How good was the model performance? Was it better or worse for some classes in comparison to others? If so, can you think of any explanation for why that might be the case?
    - What spectral features do you think might be used to distinguish between ketones and carboxylic acids?
    - How was the overall model accuracy for the multiclass classification in comparison to the binary classification? Can you think of an explanation for why this might be the case?
- Optional: If you tried something different in the analysis, discuss the results! What did you learn?