TECHNISCHE
UNIVERSITÄT
DARMSTADT

Clinical Applications of Brain Imaging, Stimulation, and Modeling

Special Exercise 2

Talha Bhatti (2667250)

Contents

List of Figures

1 Machine Learning in Neuroscience and Neuroimaging: Principles, Advantages, Challenges and Clinical Applications

In the recent years, Machine Learning methods have found many applications in the clinical domain, ncluding neuroscience and medical imaging. These techniques consist of sophisticated statistical algorithms which leverage large datasets and computational resoruces to discover hidden patterns in the data.

In this essay, two research articles are explored, providing an ML primer in neuroscience [1] and highlighting deep learning approaches to improve neuroimaging techniques [2]. The following is the summary of the methods discussed in these papers, followed by an examination of their advantages, clinical relevance and common pitfalls, as well as specific applications in the domain.

1.1 Core Principles of Machine Learning in Neuroscience and Imaging

1.1.1 Overview of ML Methods

As highlighted by [1], ML can be viewed as a subset of artificial intelligence centered around algorithms that learn patterns from data that cannot be found using regular statistical methods. Within ML, the two common approaches are supervised and unsupervised learning. In supervised learning, the algorithm is presented with labeled data, where each instance includes both an input, say an image and a known output label, say the presence of a disease. The goal is to learn a model that can accurately predict the label for unseen data. On the other hand, unsupervised learning involves unlabeled data where the goal of the algorithm is to infer hidden structures or patterns without any prior knowledge of the correct output.

1.1.2 Deep Learning and Neural Networks

Deep learning, more extensively covered in [2], is a specialized branch of ML employing artificial neural networks with multiple layers. Although these neural networks are looesly inspired by biological neurons as seen in 1, under the hood they perform mathematical functions rather than electrochemical operations in the brain.

Convolutional neural networks (CNNs) are especially relevant in image processing tasks, including neuroimaging, because they account for spatial and hierarchical features within images. These networks learn both low-level image features like edges and high-level features like organs, blood vessels. Over the years, various different CNN architectures have been developed for various medical and neuroimaging tasks like classification, detection, segmentation, registration, etc.

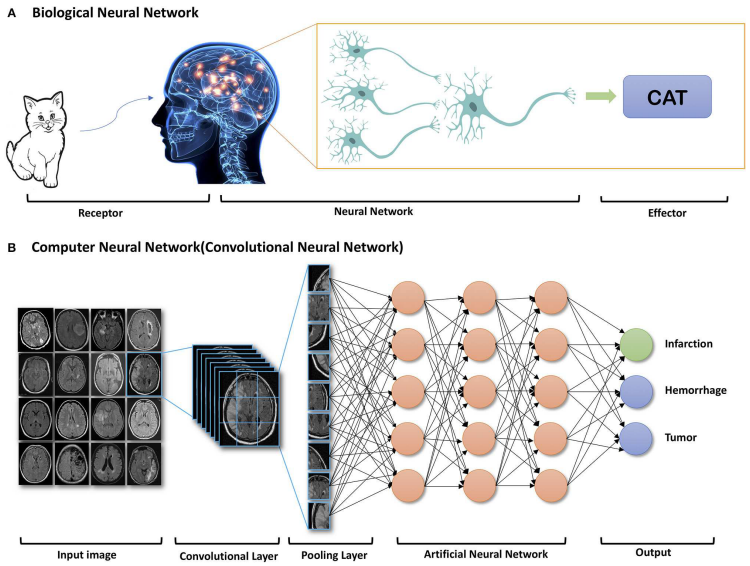

Figure 1: Example of components of Biologic Neural Network (**A**) and Computer Neural Network (**B**) [2]

1.1.3 Terminology and Model Validation

Both articles emphasize that ML models typically require at least two datasets.

- A "train" set to train the algorithm's parameters
- A "test" set to evaluate and validate the model's performance after training

Additionally, a valitation set is often used during training to guide the learning process. While some authors interchangeably use the term "validation set" and "test set", in terms of research and as evident by the terminology in both papers, it refers to the unseen dataset used to measure how well the model generalizes, as highlighted by 2.

Two additional terms to consider are overfitting and underfitting. Overfitting occurs when a model memorizes the training data but fails to perform adequately on new data, the high variance situation. Meanwhile, underfitting means that the model performs bad on both the test and train data, implying high bias.

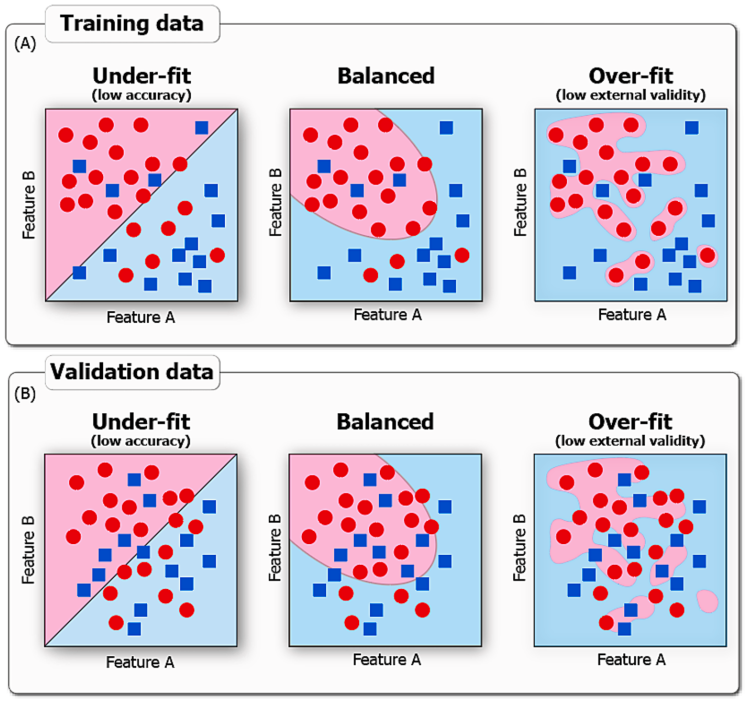

Figure 2: Underfitting and overfitting in training and test sets [1]

1.2 Advantages of Machine Learning in Neuroimaging

1.2.1 Automated Feature Extraction and Enhanced Image Quality

A major appeal of ML in neuroimaging lies in its capacity to analyze high dimensional data, such as 3D MRI volumes or complex PET scans much faster and often more accurately than manual approaches. CNNs in particular are excellent at tasks like:

- **Lesion Detection:** ML algorithms can rapidly detect anomalies such as tumors, micro-bleeds in large sets of brain images.
- **Image Reconstruction:** Deep learning can reconstruct undersampled data, reducing acquisition time and computational cost in many cases.
- **Motion Artifact Reduction:** Especially in MRI, long acquisition times can introduce motion artifacts. ML algorithms can both detect and correct such artifacts quickly.

1.2.2 Improved Safety and Accessibility

By enhancing image quality from low-dose or undersampled data, ML techniques can lower radiation exposure in CT and PET, minimizing gadolinium doses in MRI and potentially reduce scan durations 3. These developments can not only improve the quality of life of many patients, but also provide economical benefits to health insurance companies and improve the overall healthcare infrastructure.

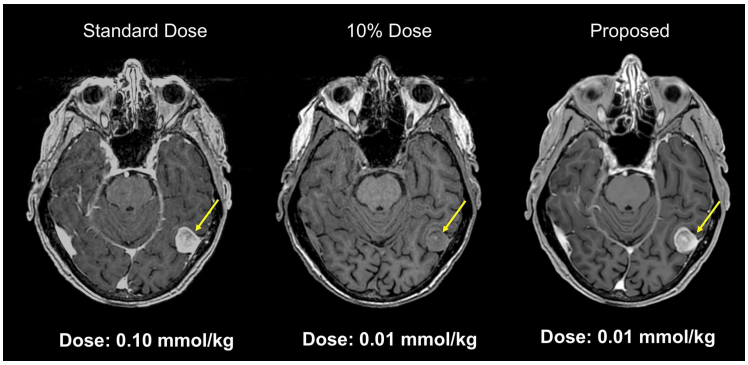

Figure 3: Example of low-dose contrast-enhaced MRI [2]

1.2.3 Personalized and Predictive Utility

ML algorithms can identify nuanced patterns in patient data that correlate with specific conditions, disease risks or response to therapy or medication. In both articles, there is discussion on the deployment of ML for predicting tumor malignancy, distinguishing neurological disorders and offering real-time insights for visualization, monitoring and planning in a clinical setting.

1.3 Challenges and Limitations

1.3.1 "Black Box" Complexity

While Deep learning networks are powerful tools, they are notoriously known as "black boxes" models. Which means, it can be difficult to interpret exactly why a network produces a particular output. This lack of transparency raises concerns in clinical contexts, where traceability of medical decisions is critical for ethical and legal reasons.

1.3.2 Overfitting and Data Bias

The bias variance trade-off is the at the core of many ML and DL alorithms and need to be addressed in model development to ensure optimal generalization. Moreover, many datasets, especially in medical and neuroimaging context suffer from the class imbalance problem, where model generalizes better on the majority class and has poor performance on minority class. Although, this class imbalance represents real-life scenarios (less people are likely to be suffering from a certain disease in a global population), it needs to be handled correctly to enhance DL model performance.

1.3.3 Data Requirements and Diversity

Creating sufficiently large and diverse datasets for ML model training remains a major bottleneck. Patient privacy regulations, high labeling costs and variations in scanner hardware across different institutions can limit the pooling of data. Models must be thoroughly tested across multiple centers to demonstrate true generalizability.

1.3.4 Validation Strategies

Models must be assessed on thoroughly independent test sets, ideally collected in separate settings or institutions. This is especially necessary for clinical adoption, as any subtle differences in image acquisition protocols can negatively impact model performance. In most clinical applications, datasets are heterogeneous and test or validation data might be out-of-distribution from the training data, severely limiting the model's performance.

1.4 Clinical Applications and Future Directions

1.4.1 Brain Tumor and Stroke Analysis

One of the earliest successful ML applications in neuroimaging is tumor characterization, including the detection of glioblastomas and metastases. ML algorithms can also be used to quantify ischemic regions in stroke patients quickly, facilitating timely interventions.

1.4.2 Low-Dose PET and MR Imaging

The practical advantage of reconstructing high-quality images from greatly reduced PET tracer doses or undersampled MRI acquisitions is highlighted by [2] as seen in 4. These methods are already showing promise in clinical research settings to lower radiation risk and shorten scan times. In some centers, low-dose PET imaging guided by deep learning is moving closer to routine use, particularly in research protocols for vulnerable populations.

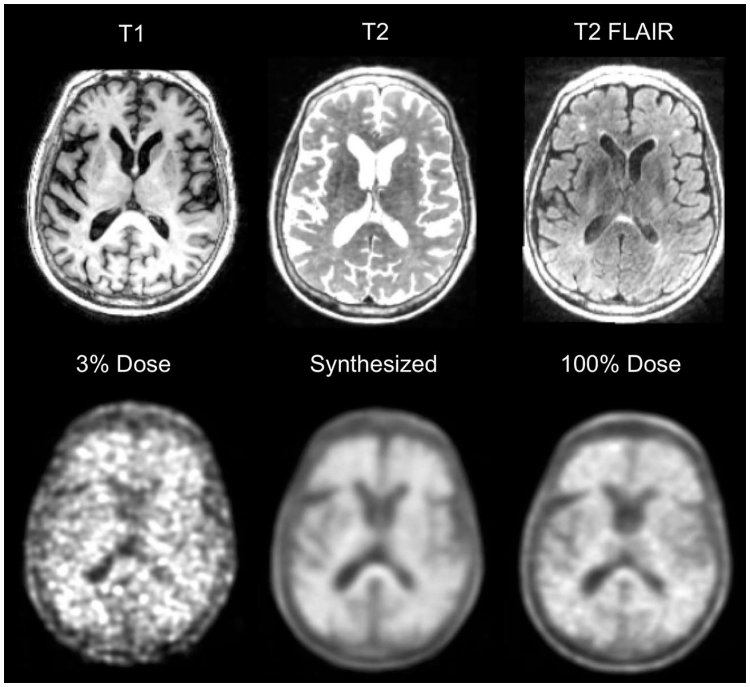

Figure 4: Example of ultra-low dose 18F-florbetaben PET/MRI [2]

1.4.3 Dementia and Degenerative Disease

There is growing interest in combining ML based analysis of both PET and MRI scans for earlier and more accurate detection of neurodegenerative disorders like Alzheimer's disease. The ability to reliably generate "synthetic" full-dose images or to enhance subtle anatomical features can help spot disease biomarkers at earlier stages, thereby improving patient outcomes.

1.4.4 Motion-Corrected and Super-Resolved Imaging

Clinical interest in super-resolution techniques is particularly high, as these might allow MRI scanners to acquire data more rapidly while still achieving the resolution necessary for complex diagnoses 5. This reduces patient discomfort and could mitigate motion artifacts, making scans more feasible for pediatric or critically ill patients.

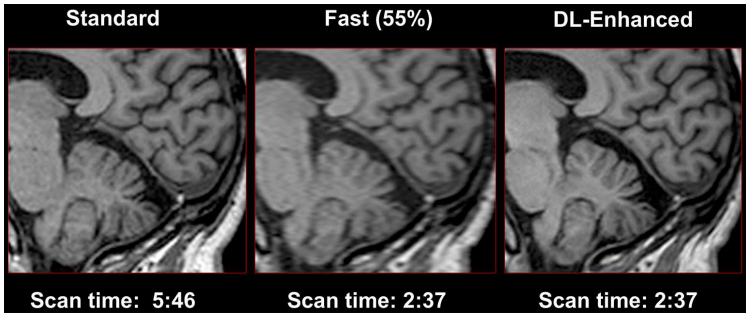

Figure 5: Use of CNNs to perform super-resolution [2]

Overall, many of these applications are still in research phases but they have high potential. Certain AI-driven denoising and super-resolution modules have even started to become commercially available, offering integrated solutions on modern scanner platforms. With sufficient validation, these methods may soon become mainstream in clinical practice.

1.5 Conclusion

In conclusion, ML and particularly DL techniques are reshaping neuroscience and neuroimaging by enhancing diagnostic accuracy, improving image quality and enabling personalized medicine. While challenges such as interpretability, data bias and regulatory constraints remain, ongoing research and validation efforts are paving the way for broader clinical integration. As ML technologies continue to evolve, they have the potential to revolutionize medical imaging, ultimately leading to more efficient, accessible and effective patient care.

References

[1] Fakhirah Badrulhisham et al. "Machine learning and artificial intelligence in neuroscience: A primer for researchers". In: *Brain, Behavior, and Immunity* 115 (2024), pp. 470–479. ISSN: 0889-1591. DOI: https://doi.org/10.1016/j.bbi.2023.11.005. URL: https://www.sciencedirect.com/science/article/pii/S0889159123003380.

[2] Guangming Zhu et al. “Applications of Deep Learning to Neuro-Imaging Techniques”. In: *Frontiers in Neurology* 10 (2019). ISSN: 1664-2295. DOI: 10.3389/fneur.2019.00869. URL: https://www.frontiersin.org/journals/neurology/articles/10.3389/fneur.2019.00869.

2 Python Exercises

2.1 Task 1

```
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from scipy.stats import f_oneway, spearmanr
from statsmodels.stats.multitest import multipletests
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
    ConfusionMatrixDisplay, classification_report
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.dummy import DummyClassifier


plt.style.use('default')



#%% Task 1: Load and inspect node strength data (10 points)

"""
Here, we are gonna look at a clinical dataset (modified for this task).
In this dataset, MEG data has been recorded from subjects labeled as either:
   1. healthy
   2. suffering from major depressive disorder (MDD)
   3. suffering from post-traumatic stress disorder (PTSD)

From these subjects, MEG data was recorded, source-reconstructed into 400
    parcels,
    and phase synchrony between parcels was estimated in 32 frequencies using
    the iPLV.
To reduce data size, you are given the node strength of iPLV rather than
    the edge strengths.

Load the node strength data and the labels that you downloaded from the shared
     folder.


Run a statistical test that is suited for comparing 3 (or more) groups at once
    .
Correct the p_values for multiple comparisons with Benjamini-Hochberg method.
```

```
Plot the mean strength for each of the three groups as a function of frequency
   (with logarithmic x-axis). Add a legend.
Indicate on the plot for which frequencies the test was significant (p<0.05)
    after correction.

"""

DATA_DIR = 'data'
NODE_STRENGTH_DIR = os.path.join(DATA_DIR, 'node_strength.npy')
LABELS_DIR = os.path.join(DATA_DIR, 'labels.csv')
SYMPTOMS_DIR = os.path.join(DATA_DIR, 'symptoms.csv')

# Load the data
node_strength = np.load(NODE_STRENGTH_DIR)
labels = pd.read_csv(LABELS_DIR, header=None).squeeze().values
symptoms = pd.read_csv(SYMPTOMS_DIR, sep=';')

freqs = np.array([
        2.1,  2.5,  2.9,  3.3,  3.7,  4.15, 4.8,  5.4,  5.9,
        6.6,  7.4,  8.1,  9.0,  9.8, 10.9, 11.9, 13.1, 14.8,
       16.3, 17.8, 19.7, 21.6, 23.7, 26.6, 28.7, 31.8, 34.5 ,
       37.9, 42.5, 46.9, 52.1, 59.3 ])

group_names = ['HC', 'MDD', 'PTSD']
node_strength_avg = node_strength.mean(axis=2)

# ANOVA for each frequency
p_values = []
for i in range(node_strength_avg.shape[1]):
    group_data = [node_strength_avg[labels == j, i] for j in range(3)]
    stat, p = f_oneway(*group_data)
    p_values.append(p)

p_values = np.array(p_values)

# Correct for multiple comparisons using Benjamini-Hochberg
_, p_corrected, _, _ = multipletests(p_values, method='fdr_bh')

plt.figure(figsize=(12, 6))
for i, group in enumerate(group_names):
    mean_strength = node_strength_avg[labels == i].mean(axis=0)
    plt.plot(freqs, mean_strength, label=group)

significant_freqs = freqs[p_corrected < 0.05]
print("Significant frequencies after correction (p < 0.05):",
    significant_freqs)
significant_y = node_strength_avg.mean(axis=0)[p_corrected < 0.05]
plt.scatter(significant_freqs, significant_y, color='red', marker='*', label='
    p < 0.05')

plt.xscale('log')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Mean Node Strength')
```

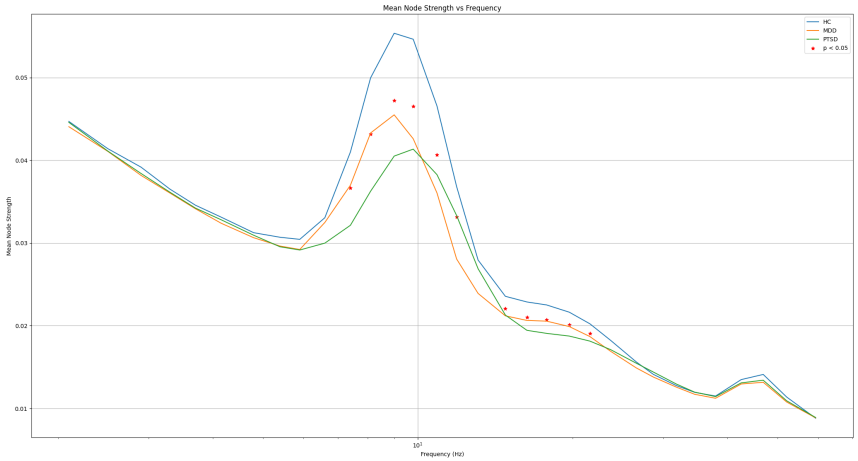

Figure 6: Mean Node Strength vs Frequency

```
plt.title('Mean Node Strength vs Frequency')
plt.legend()
plt.grid(True)
plt.show()
```

```
Significant frequencies after correction (p < 0.05):
[ 7.4 8.1 9.0 9.8 10.9 11.9 14.8 16.3 17.8 19.7 21.6]
```

2.2 Task 2

```
#%% Task 2: Inspect symptoms (6 points)


"""
Load the symptom data.
For each of the two symptoms, make violinplots for the 3 groups.

"""

symptoms['Group'] = labels
symptoms['Group'] = symptoms['Group'].map({0: 'HC', 1: 'MDD', 2: 'PTSD'})

plt.figure(figsize=(14, 6))
# Color palette
palette = sns.color_palette("Set2", n_colors=3)

# BDI
plt.subplot(1, 2, 1)
sns.violinplot(x='Group', y='BDI', data=symptoms, hue='Group', palette=palette
    , legend=False)
plt.title('BDI Scores by Group')
```

```
plt.xlabel('Group')
plt.ylabel('BDI Score')

# PCL
plt.subplot(1, 2, 2)
sns.violinplot(x='Group', y='PCL', data=symptoms, hue='Group', palette=palette
    , legend=False)
plt.title('PCL Scores by Group')
plt.xlabel('Group')
plt.ylabel('PCL Score')

plt.tight_layout()
plt.show()
```

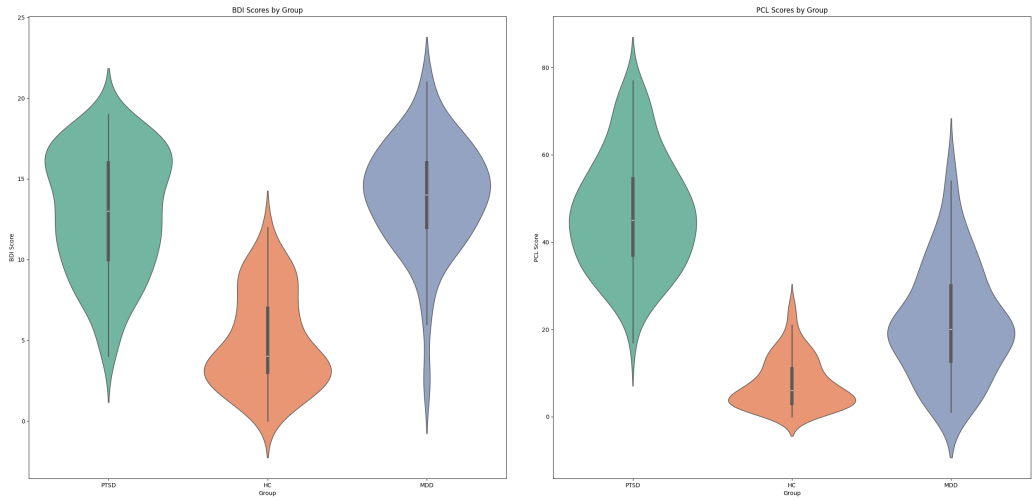

Figure 7: BDI and PCL scores

2.3 Task 3

```
#%% Task 3: Correlate symptoms with node strength. (10 points)

"""
For each frequency, compute the correlation of subjects' mean node strength
   with each of the two symptoms. Which test is more appropriate here?

Plot the correlations with the symptoms as a function of frequency (log x-axis
    ).
For each symptom, mark the frequencies where the test was significant (p <
    0.05).

Use the Benjaminini-Hochberg method to control for multiple comparisons
  (for each symptom separately) and repeat the plot with the new significants.

Describe your findings.
```

```
"""


correlations_bdi = []
p_values_bdi = []
correlations_pcl = []
p_values_pcl = []

# Spearman correlation for each frequency
for i in range(node_strength_avg.shape[1]):
    mean_strength = node_strength_avg[:, i]

    # BDI
    corr_bdi, p_bdi = spearmanr(mean_strength, symptoms['BDI'])
    correlations_bdi.append(corr_bdi)
    p_values_bdi.append(p_bdi)

    # PCL
    corr_pcl, p_pcl = spearmanr(mean_strength, symptoms['PCL'])
    correlations_pcl.append(corr_pcl)
    p_values_pcl.append(p_pcl)

correlations_bdi = np.array(correlations_bdi)
p_values_bdi = np.array(p_values_bdi)
correlations_pcl = np.array(correlations_pcl)
p_values_pcl = np.array(p_values_pcl)

# Benjamini-Hochberg correction
_, p_corrected_bdi, _, _ = multipletests(p_values_bdi, method='fdr_bh')
_, p_corrected_pcl, _, _ = multipletests(p_values_pcl, method='fdr_bh')

plt.figure(figsize=(14, 6))

# BDI
plt.subplot(1, 2, 1)
plt.plot(freqs, correlations_bdi, label='BDI Correlation')
plt.scatter(freqs[p_values_bdi < 0.05], correlations_bdi[p_values_bdi < 0.05],
            color='red', marker='*', label='Significant (p < 0.05)')
plt.scatter(freqs[p_corrected_bdi < 0.05], correlations_bdi[p_corrected_bdi <
    0.05],
            color='blue', marker='o', label='Corrected Significant (p < 0.05)'
    )
plt.xscale('log')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Spearman Correlation')
plt.title('Correlation with BDI vs Frequency')
plt.legend()
plt.grid(True)

# PCL
plt.subplot(1, 2, 2)
plt.plot(freqs, correlations_pcl, label='PCL Correlation')
plt.scatter(freqs[p_values_pcl < 0.05], correlations_pcl[p_values_pcl < 0.05],
            color='red', marker='*', label='Significant (p < 0.05)')
```

```
plt.scatter(freqs[p_corrected_pcl < 0.05], correlations_pcl[p_corrected_pcl <
    0.05],
            color='blue', marker='o', label='Corrected Significant (p < 0.05)'
    )
plt.xscale('log')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Spearman Correlation')
plt.title('Correlation with PCL vs Frequency')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```

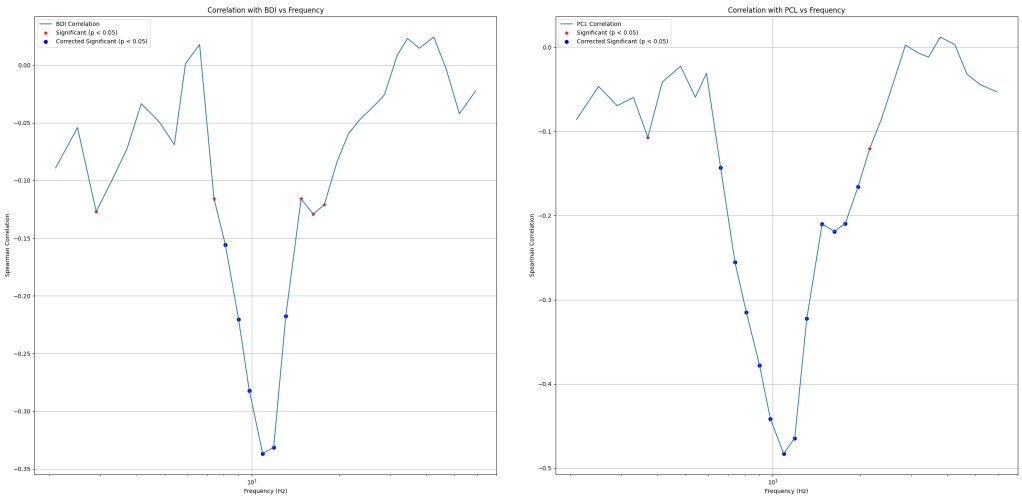

Figure 8: Correlation with BDI and PCL with Frequency

2.4 Task 4

```
#%% Task 4: Supervised learning: Classification (14 points)


"""
Now we will run a classifier on our data.
Use the node strength values (or a subset and/or means of it) as features.

In this task, it is sufficient to split the data into training and
    test sets, no validation set will be used.

Choose a classifier from sklearn library and train it on the training set,
    then test it on the test set. Report the classification accuracy on the
    training and test sets.
Plot and annotate the 3-way confusion matrix.

Try to select features so that you get > 80% accuracy on test data.
```

```
If this were real data, how useful would the classifier be for diagnosing
    patients?

"""

X = node_strength_avg
y = labels

# 80-20 split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=42, stratify=y)

# Random Forest Classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# Predictions
y_train_pred = clf.predict(X_train)
y_test_pred = clf.predict(X_test)

# Calculate accuracy
train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)

# Classification report
report = classification_report(y_test, y_test_pred, target_names=group_names)

# Confusion matrix
cm = confusion_matrix(y_test, y_test_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=group_names)
plt.figure(figsize=(8, 6))
disp.plot(cmap=plt.cm.Blues, values_format='d')
plt.title('3-Way Confusion Matrix')
plt.show()

print(f"Training Accuracy: {train_accuracy * 100:.2f}%")
print(f"Test Accuracy: {test_accuracy * 100:.2f}%\n")
print("Classification Report:\n")
print(report)
```

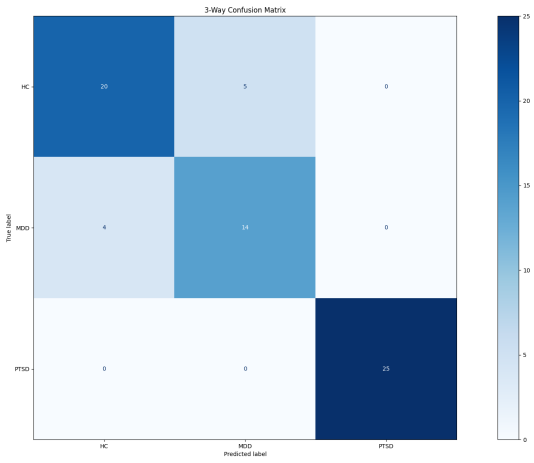

Figure 9: 3-Way Confusion Matrix for Random Forest Classifier

```
Random Forest performance:
    Training Accuracy: 100.00%
    Test Accuracy: 86.76%

Classification Report:
```

	precision	recall	f1-score	support
HC	0.83	0.80	0.82	25
MDD	0.74	0.78	0.76	18
PTSD	1.00	1.00	1.00	25
accuracy			0.87	68
macro avg	0.86	0.86	0.86	68
weighted avg	0.87	0.87	0.87	68

2.5 Task 5

```
#%% Task 5: Systematic evaluation of classifiers (10 points)

"""
In addition to the classifier you used in task 4, choose two others.
Run all 3 classifiers at least 20 times, using the same features,
    then plot the mean and stdev of accuracy (on test set) for each classifier
    .
Which one performs best?

"""
```

```
# Define classifiers
classifiers = {
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42)
    ,
    'SVM': SVC(kernel='rbf', random_state=42),
    'Logistic Regression': LogisticRegression(max_iter=1000, random_state=42)
}

# Store results
train_results = {name: [] for name in classifiers.keys()}
test_results = {name: [] for name in classifiers.keys()}
classification_reports = {name: [] for name in classifiers.keys()}

# Run each classifier 20 times
for _ in range(20):
    # Split the data with random shuffling
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, stratify=y
    )

    for name, clf in classifiers.items():
        clf.fit(X_train, y_train)
        y_train_pred = clf.predict(X_train)
        y_test_pred = clf.predict(X_test)
        train_accuracy = accuracy_score(y_train, y_train_pred)
        test_accuracy = accuracy_score(y_test, y_test_pred)
        train_results[name].append(train_accuracy)
        test_results[name].append(test_accuracy)
        report = classification_report(y_test, y_test_pred, target_names=
    group_names, output_dict=True, zero_division=0)
        classification_reports[name].append(report)

# Calculate mean and std of accuracies
mean_train_accuracies = {name: np.mean(accs) for name, accs in train_results.
    items()}
mean_test_accuracies = {name: np.mean(accs) for name, accs in test_results.
    items()}

mean_classification_reports = {}
for name, reports in classification_reports.items():
    mean_report = {}
    # Average metrics for each class and overall
    for key in reports[0].keys():
        if isinstance(reports[0][key], dict):
            mean_report[key] = {
                metric: np.mean([rep[key][metric] for rep in reports]) for
    metric in reports[0][key].keys()
            }
        else:
            mean_report[key] = np.mean([rep[key] for rep in reports])
    mean_classification_reports[name] = mean_report
```

```
# Plotting the results
plt.figure(figsize=(10, 6))
plt.bar(mean_test_accuracies.keys(), mean_test_accuracies.values(),
        yerr=[np.std(accs) for accs in test_results.values()], capsize=10,
    color=['red', 'green', 'blue'])
plt.ylabel('Test Accuracy')
plt.title('Mean and Standard Deviation of Test Accuracy (20 Runs)')
plt.grid(axis='y')
plt.show()

# Train/test accuracy and mean classification report
for name in classifiers.keys():
    print(f"\n{name}")
    print(f"Mean Training Accuracy: {mean_train_accuracies[name] * 100:.2f}%")
    print(f"Mean Test Accuracy: {mean_test_accuracies[name] * 100:.2f}%\n")

    print("Mean Classification Report:")
    report = mean_classification_reports[name]
    for label, metrics in report.items():
        if isinstance(metrics, dict):
            print(f"  {label}:")
            for metric, value in metrics.items():
                print(f"    {metric}: {value:.2f}")
        else:
            print(f"  {label}: {metrics:.2f}")
```

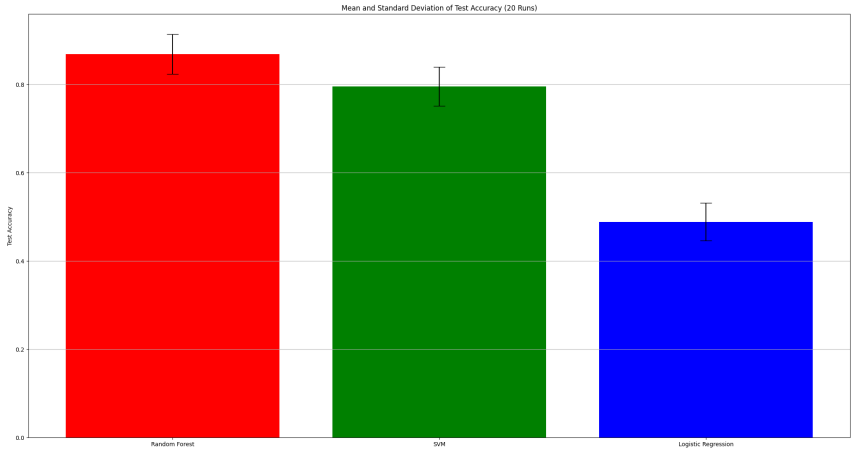

Figure 10: Mean Test Accuracies of 3 Classifiers

```
Random Forest
Mean Training Accuracy: 100.00%
Mean Test Accuracy: 86.84%

Mean Classification Report:
```

```
  HC:
    precision: 0.82
    recall: 0.86
    f1-score: 0.84
    support: 25.00
  MDD:
    precision: 0.79
    recall: 0.72
    f1-score: 0.75
    support: 18.00
  PTSD:
    precision: 0.97
    recall: 0.99
    f1-score: 0.98
    support: 25.00
  accuracy: 0.87
  macro avg:
    precision: 0.86
    recall: 0.85
    f1-score: 0.86
    support: 68.00
  weighted avg:
    precision: 0.87
    recall: 0.87
    f1-score: 0.87
    support: 68.00

SVM
Mean Training Accuracy: 83.40%
Mean Test Accuracy: 80.81%

Mean Classification Report:
  HC:
    precision: 0.83
    recall: 0.79
    f1-score: 0.81
    support: 25.00
  MDD:
    precision: 0.70
    recall: 0.65
    f1-score: 0.67
    support: 18.00
  PTSD:
```

```
    precision: 0.87
    recall: 0.94
    f1-score: 0.90
    support: 25.00
  accuracy: 0.81
  macro avg:
    precision: 0.80
    recall: 0.79
    f1-score: 0.79
    support: 68.00
  weighted avg:
    precision: 0.81
    recall: 0.81
    f1-score: 0.81
    support: 68.00

Logistic Regression
Mean Training Accuracy: 48.33%
Mean Test Accuracy: 46.76%

Mean Classification Report:
  HC:
    precision: 0.56
    recall: 0.49
    f1-score: 0.52
    support: 25.00
  MDD:
    precision: 0.00
    recall: 0.00
    f1-score: 0.00
    support: 18.00
  PTSD:
    precision: 0.42
    recall: 0.78
    f1-score: 0.55
    support: 25.00
  accuracy: 0.47
  macro avg:
    precision: 0.33
    recall: 0.42
    f1-score: 0.36
    support: 68.00
  weighted avg:
```

```
precision: 0.36
recall: 0.47
f1-score: 0.39
support: 68.00
```

2.6 Task 6

```
#%% Baseline Approaches

"""
This is an additional task. I want to implement some non-ML baseline
    classifiers for comparison.
Scikit-learn offers the following 2 strategies for baseline DummyClassifier
1. 'stratified': Predicts labels by respecting the training sets class
    distribution (randomly).
2. 'most_frequent': Always predicts the most frequent class from the training
    set.
3. 'uniform': Predicts each class randomly with equal probability.
4. 'constant': Always predicts a constant class label (specified via the '
    constant' parameter).

resource: https://medium.com/@preethi_prakash/understanding-baseline-models-in
    -machine-learning-3ed94f03d645

"""

dummy_strategies = {
    'stratified': DummyClassifier(strategy='stratified', random_state=42),
    'most_frequent': DummyClassifier(strategy='most_frequent'),
    'uniform': DummyClassifier(strategy='uniform', random_state=42),
    'constant(HC)': DummyClassifier(strategy='constant', constant=0)    # For
    constant, I will make the algo always predict HC.
}

train_accuracies = []
test_accuracies = []
strategy_names = []

for name, dummy_clf in dummy_strategies.items():
    dummy_clf.fit(X_train, y_train)

    # Predictions on both train and test
    y_train_pred = dummy_clf.predict(X_train)
    y_test_pred = dummy_clf.predict(X_test)

    # Accuracy
    train_acc = accuracy_score(y_train, y_train_pred)
    test_acc = accuracy_score(y_test, y_test_pred)

    # Store results
    train_accuracies.append(train_acc)
```

```
    test_accuracies.append(test_acc)
    strategy_names.append(name)

    # Print results
    print(f"\nDummy strategy: {name}")
    print(f"  Training Accuracy: {train_acc * 100:.2f}%")
    print(f"  Test Accuracy:     {test_acc * 100:.2f}%")

    print("  Classification Report (Test):")
    print(classification_report(y_test, y_test_pred, target_names=group_names,
     zero_division=0))

plt.figure(figsize=(12, 5))

# Training Accuracy
plt.subplot(1, 2, 1)
plt.bar(strategy_names, train_accuracies, color=['red', 'green', 'blue', '
    yellow'])
plt.title("Dummy Classifiers: Training Accuracy")
plt.ylabel("Accuracy")
plt.ylim([0, 1.0])
for i, acc in enumerate(train_accuracies):
    plt.text(i, acc+0.01, f"{acc*100:.1f}%", ha='center')

# Test Accuracy
plt.subplot(1, 2, 2)
plt.bar(strategy_names, test_accuracies, color=['red', 'green', 'blue', '
    yellow'])
plt.title("Dummy Classifiers: Test Accuracy")
plt.ylabel("Accuracy")
plt.ylim([0, 1.0])
for i, acc in enumerate(test_accuracies):
    plt.text(i, acc+0.01, f"{acc*100:.1f}%", ha='center')

plt.tight_layout()
plt.show()
```

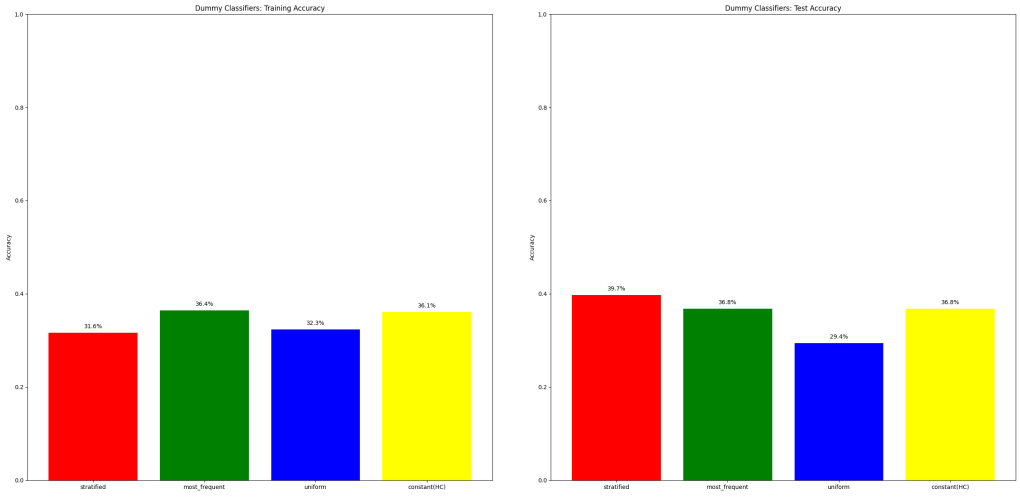

Figure 11: Baseline Approach Accuracies

```
Dummy strategy: stratified
  Training Accuracy: 31.60%
  Test Accuracy:     39.71%
  Classification Report (Test):
              precision    recall  f1-score   support

          HC       0.43      0.40      0.42        25
         MDD       0.31      0.22      0.26        18
        PTSD       0.41      0.52      0.46        25

    accuracy                           0.40        68
   macro avg       0.38      0.38      0.38        68
weighted avg       0.39      0.40      0.39        68


Dummy strategy: most_frequent
  Training Accuracy: 36.43%
  Test Accuracy:     36.76%
  Classification Report (Test):
              precision    recall  f1-score   support

          HC       0.00      0.00      0.00        25
         MDD       0.00      0.00      0.00        18
        PTSD       0.37      1.00      0.54        25

    accuracy                           0.37        68
   macro avg       0.12      0.33      0.18        68
```

```
weighted avg       0.14      0.37      0.20        68


Dummy strategy: uniform
  Training Accuracy: 32.34%
  Test Accuracy:     29.41%
  Classification Report (Test):
              precision    recall  f1-score   support

          HC       0.26      0.20      0.23        25
         MDD       0.20      0.28      0.23        18
        PTSD       0.42      0.40      0.41        25

    accuracy                           0.29        68
   macro avg       0.29      0.29      0.29        68
weighted avg       0.30      0.29      0.30        68


Dummy strategy: constant(HC)
  Training Accuracy: 36.06%
  Test Accuracy:     36.76%
  Classification Report (Test):
              precision    recall  f1-score   support

          HC       0.37      1.00      0.54        25
         MDD       0.00      0.00      0.00        18
        PTSD       0.00      0.00      0.00        25

    accuracy                           0.37        68
   macro avg       0.12      0.33      0.18        68
weighted avg       0.14      0.37      0.20        68
```