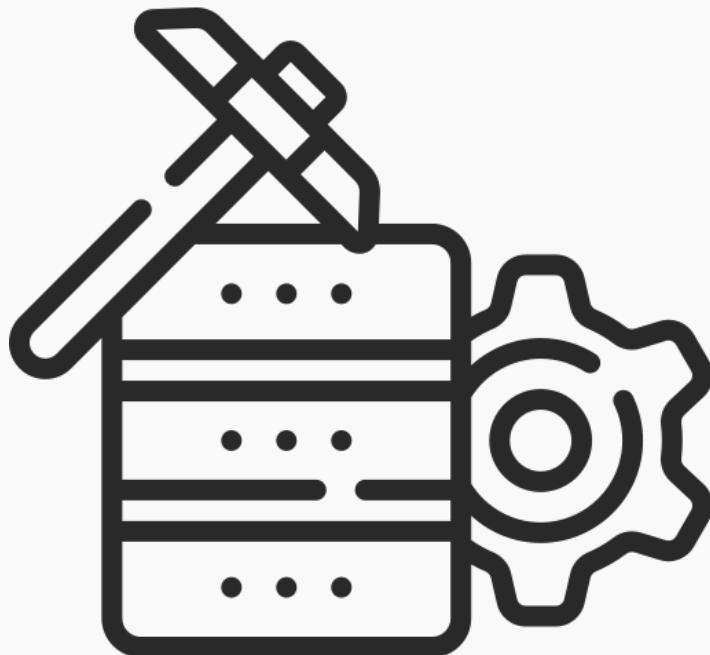


Εξόρυξη Δεδομένων και Αλγόριθμοι Μάθησης

ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ ΜΑΘΗΜΑΤΟΣ

«Ανάλυση δεδομένων επιταχυνσιομέτρων σε ανθρώπους»



- Στυλιανάκης Στυλιανός (1059713)
- Λεκαράκος Αλέξιος (1069367)

Πίνακας Περιεχομένων

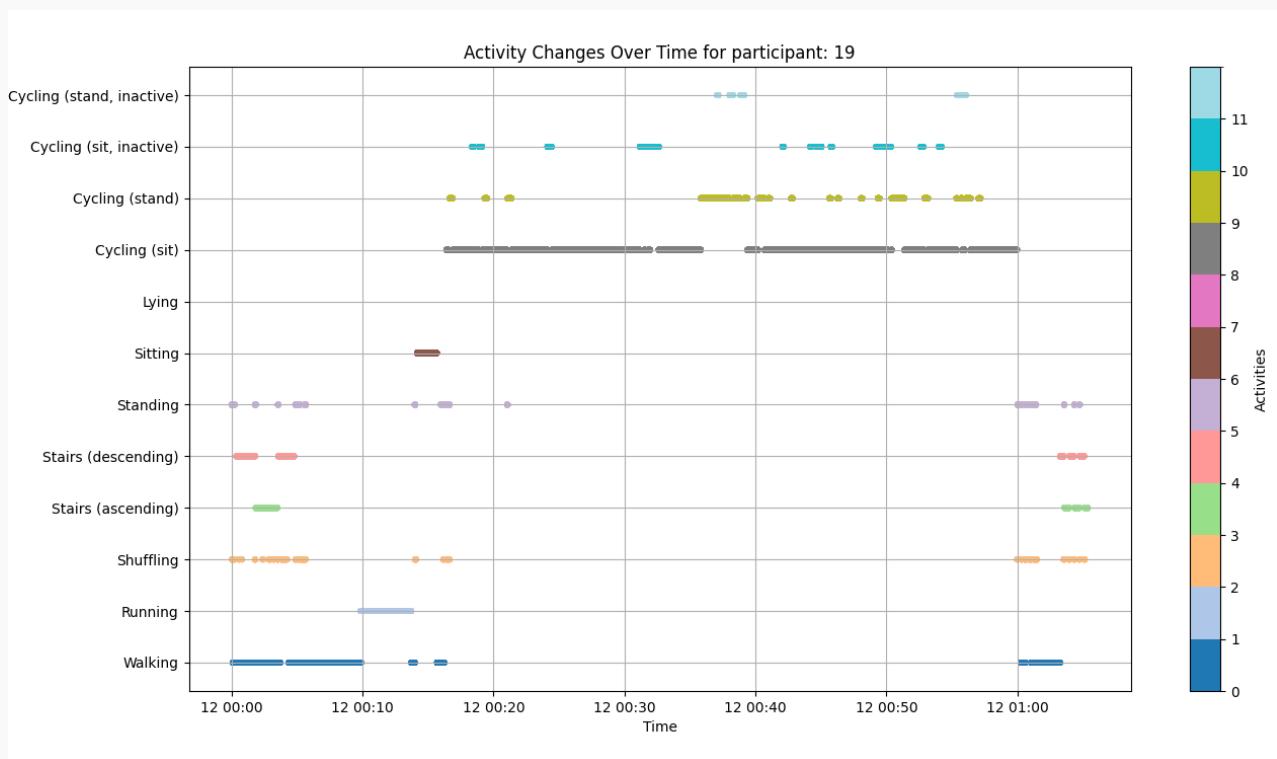
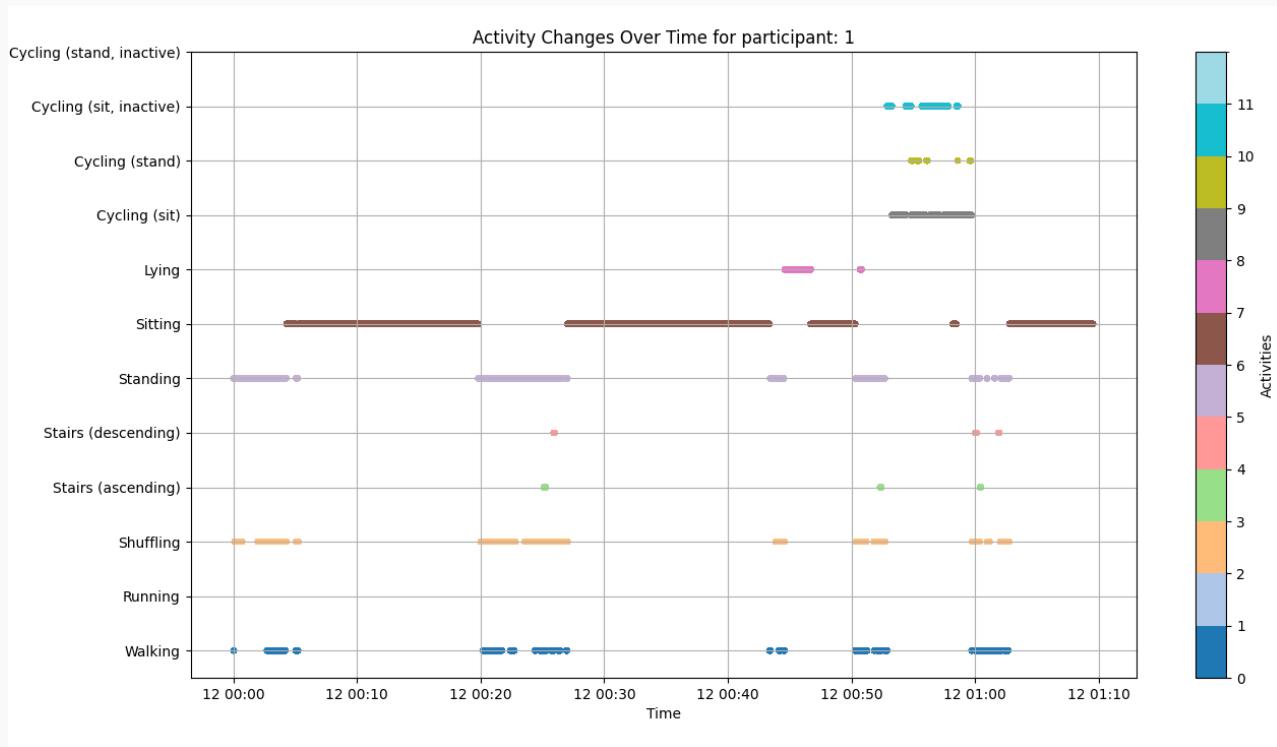
Ερώτημα 1: Αρχική ανάλυση συνόλου δεδομένων	3
1 ^ο Μέρος	3
Δραστηριότητα συμμετεχόντων κατά τη διάρκεια του πειράματος	3
Διαγράμματα για την μέση διάρκεια ενασχόλησης ανά δραστηριότητα	4
2 ^ο Μέρος: Ανάλυση μετρήσεων αισθητήρων για την εύρεση συσχέτισης τους και μοτίβων.	6
Ερώτημα 2: Εκπαίδευση Ταξινομητών	7
Προ-Επεξεργασία Δεδομένων	7
Διαχείριση μη ισορροπημένης κατανομής ετικετών συνόλου δεδομένων	10
Undersampling (υποδειγματοληψία)	10
Categorical Focal Loss	13
Εκπαίδευση & Αξιολόγηση Ταξινομητών	13
Εκπαίδευση σε όλο το σύνολο δεδομένων	13
Leave-One-Out Cross-Validation	16
Αρχιτεκτονική Νευρωνικού Δικτύου	17
Αποτελέσματα	18
1 ^η Μέθοδος (Όλο το σύνολο δεδομένων)	18
2 ^η Μέθοδος (Leave-One-Out Cross-Validation)	22
Συμπεράσματα	27
Undersampling	27
Leave-One-Out Cross-Validation	27
Σύγκριση ταξινομητών	27
Ερώτημα 3: Συσταδοποίηση συμμετεχόντων	27
Κατανομή ετικετών ανά συστάδα	29
Οπτικοποίηση χρηστών χρησιμοποιώντας PCA για μείωση διαστάσεων	30
Συμπέρασμα	30

Ερώτημα 1: Αρχική ανάλυση συνόλου δεδομένων

1° Μέρος

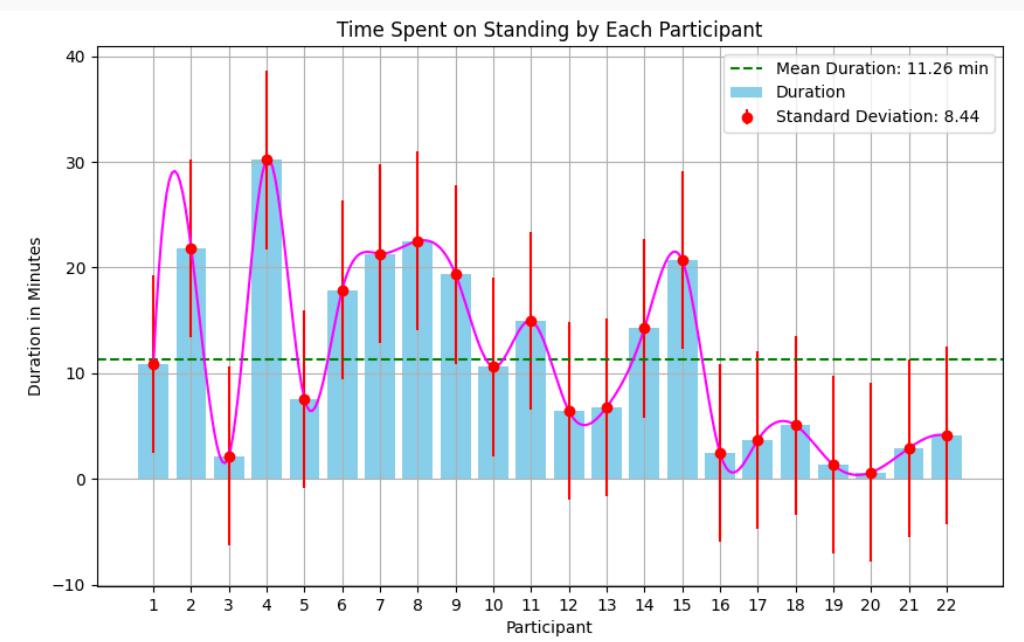
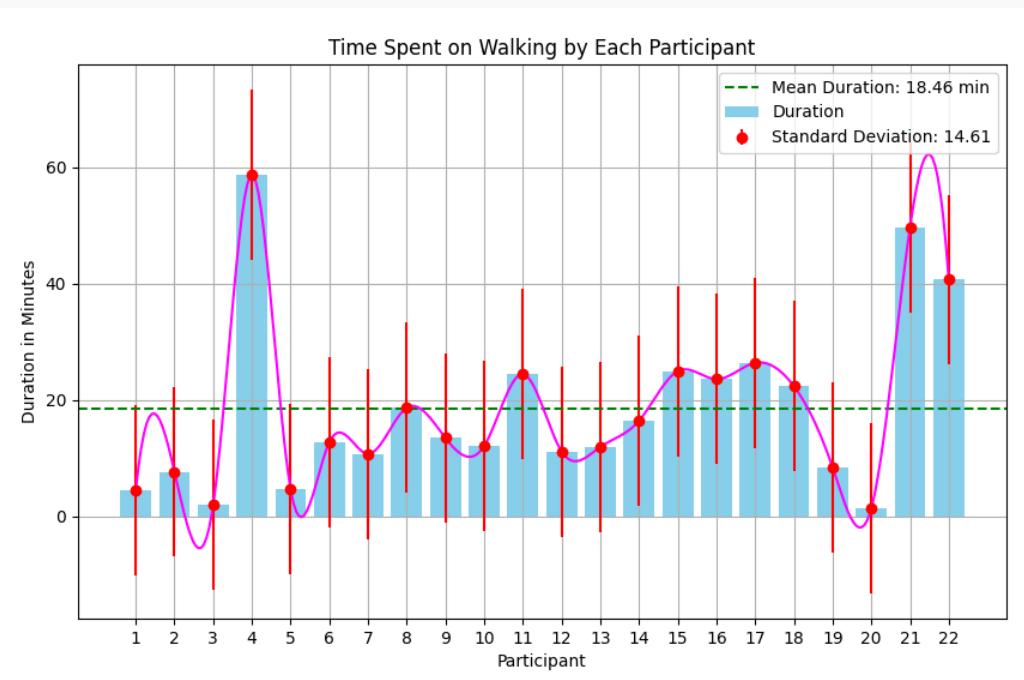
Δραστηριότητα συμμετεχόντων κατά τη διάρκεια του πειράματος

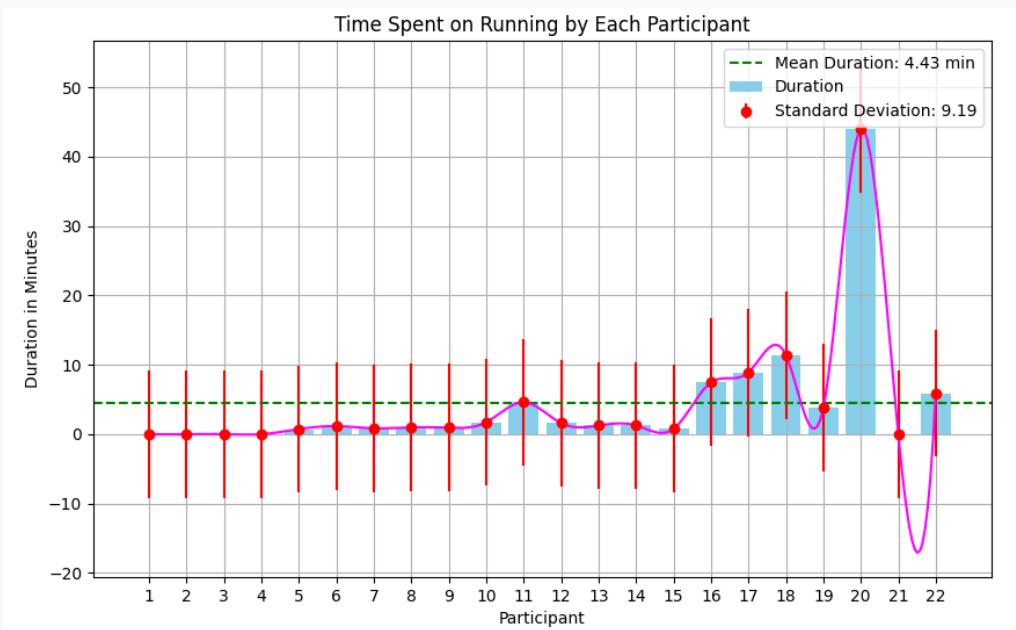
Για το πρώτο μέρος της ανάλυσης έγινε γραφική αναπαράσταση του πειράματος για κάθε συμμετέχοντα με βάση τις φυσικές δραστηριότητες που εκτέλεσε και την διάρκεια της κάθε μίας.



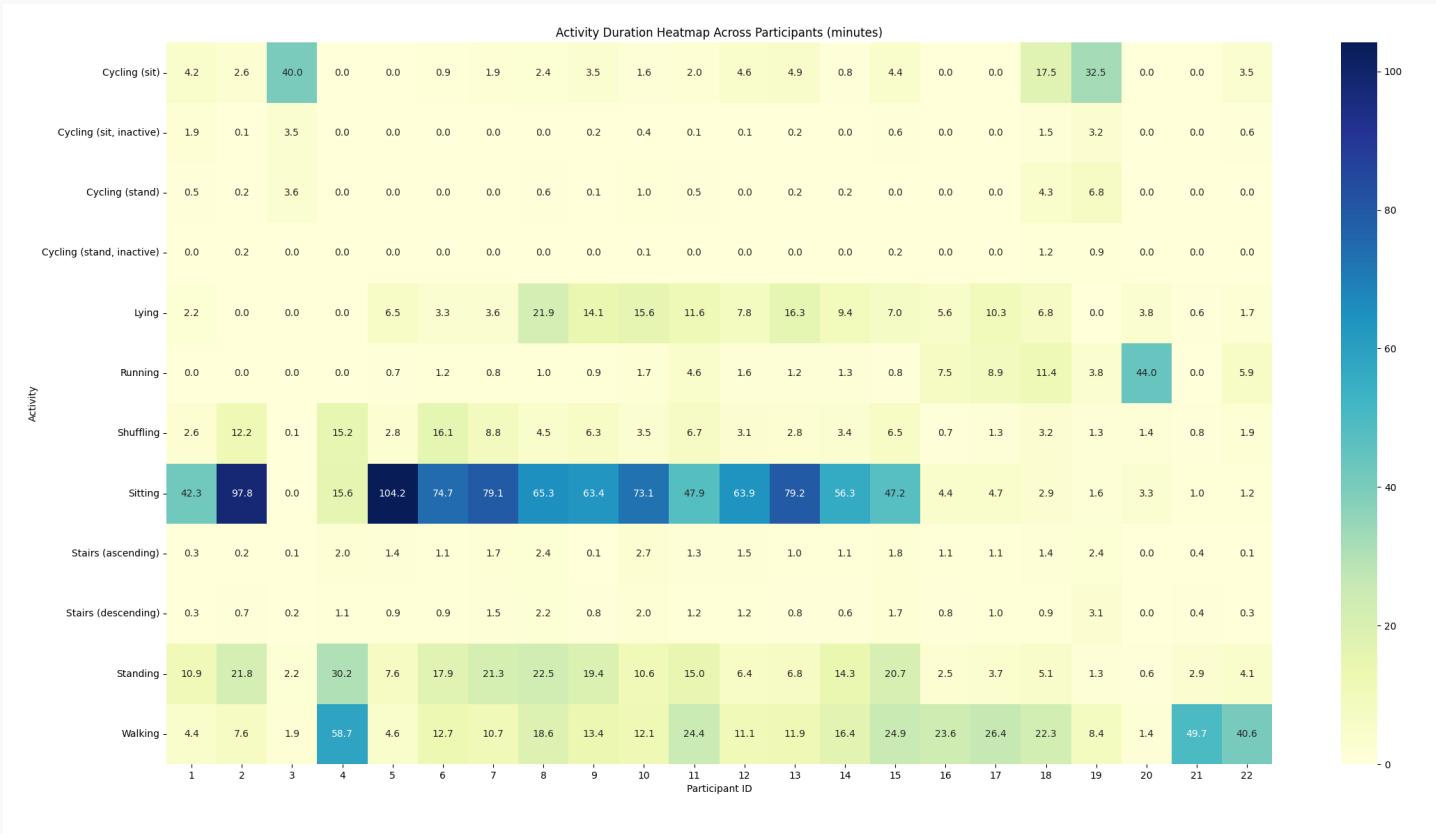
Διαγράμματα για την μέση διάρκεια ενασχόλησης ανά δραστηριότητα

Επίσης παρέχουμε διαγράμματα για την μέση διάρκεια ενασχόλησης ανά φυσική δραστηριότητα για κάθε συμμετέχοντα. Για παράδειγμα για τις δραστηριότητες Walking, Standing και Running:





Και το αντίστοιχο heatmap:

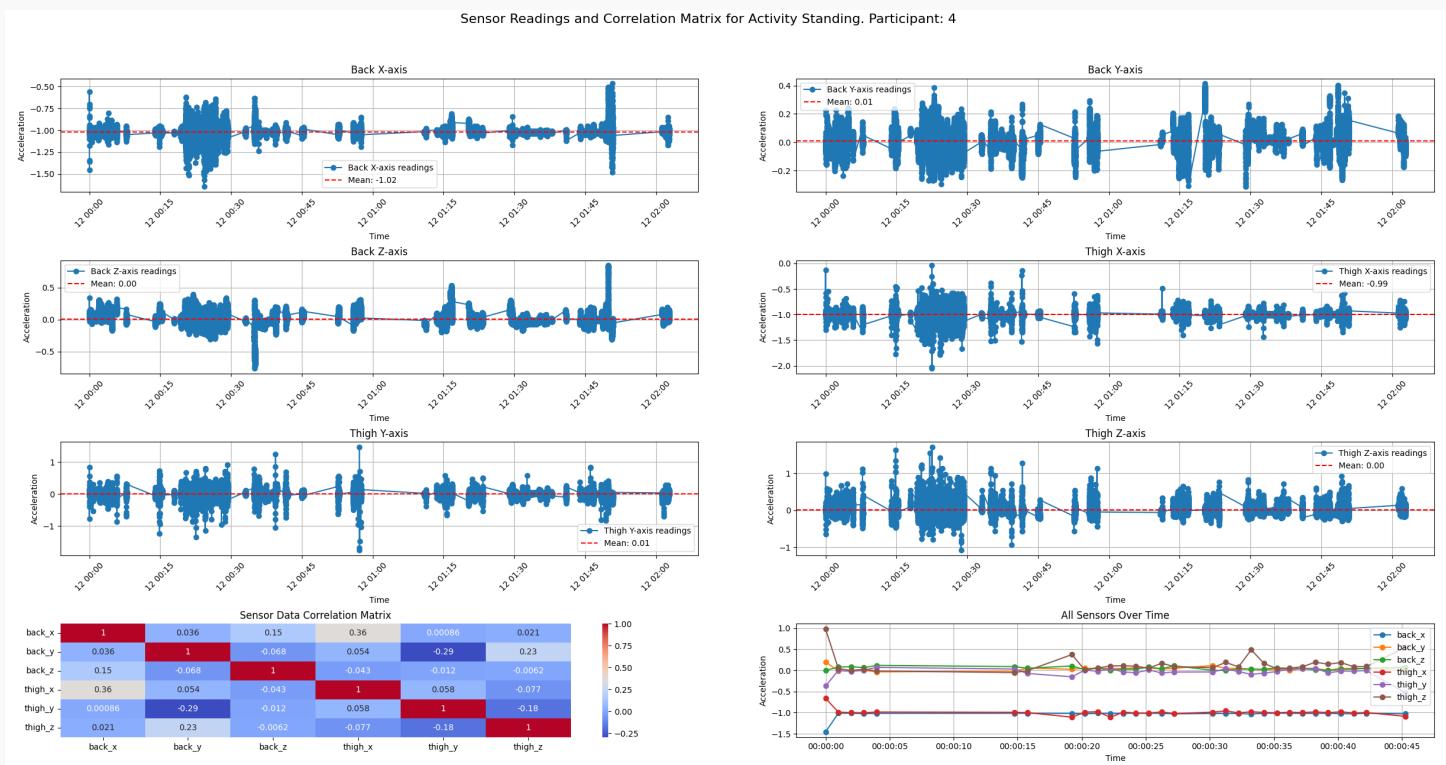


Το πρώτο μέρος της ανάλυσης μας βοηθάει σε τρία σημεία.

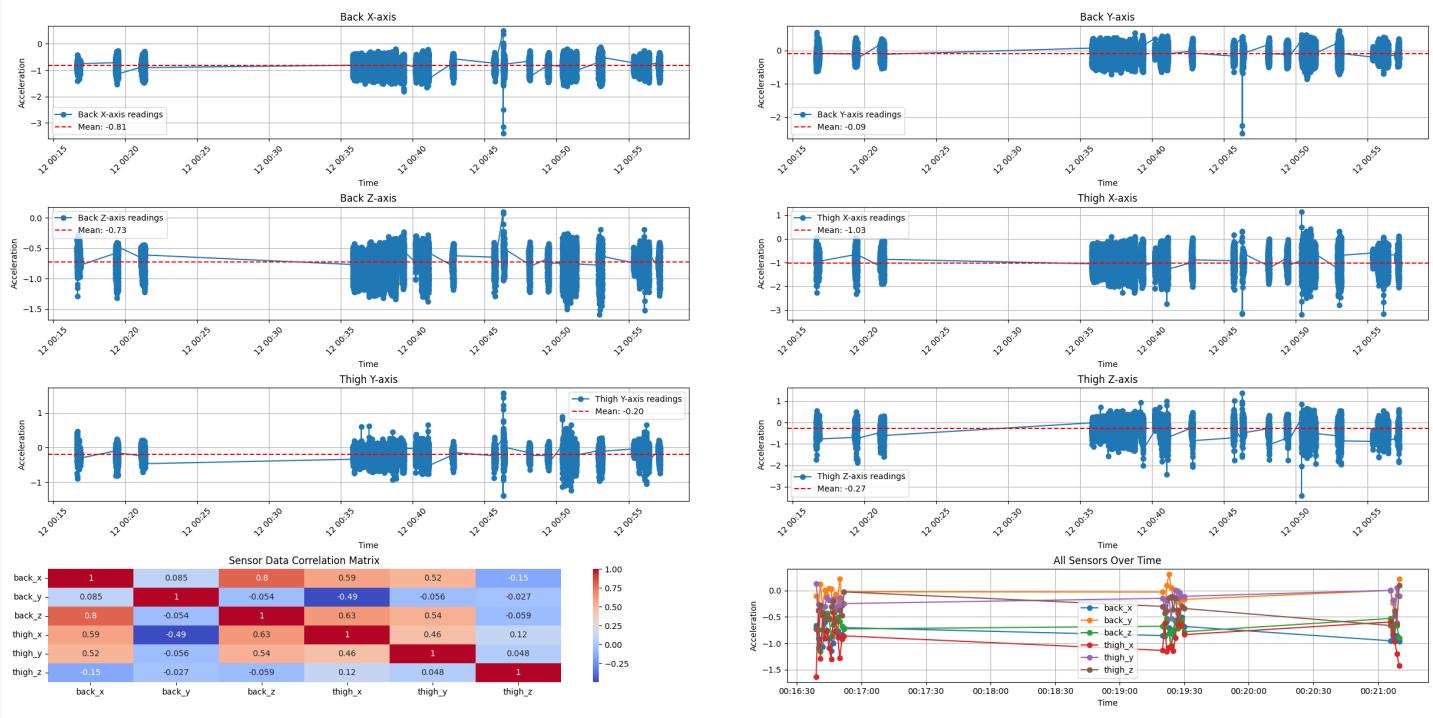
- Οπτικοποίηση της συμπεριφοράς του κάθε συμμετέχοντα κατά την διάρκεια του πειράματος.
- Δυνατότητα σύγκρισης συμμετεχόντων μετά την συσταδοποίηση.
- Επιλογή κατάλληλου δείγματος στην ανάλυση των τιμών των αισθητήρων για κάθε δραστηριότητα. Επιλέγουμε τις μετρήσεις του πιο ενεργού χρήστη για κάθε δραστηριότητα καθώς υπάρχουν περισσότερα δεδομένα και μπορούμε έτσι να ανιχνεύσουμε καλύτερα σχέσεις και μοτίβα μεταξύ των αισθητήρων.

2^ο Μέρος: Ανάλυση μετρήσεων αισθητήρων για την εύρεση συσχέτισης τους και μοτίβων.

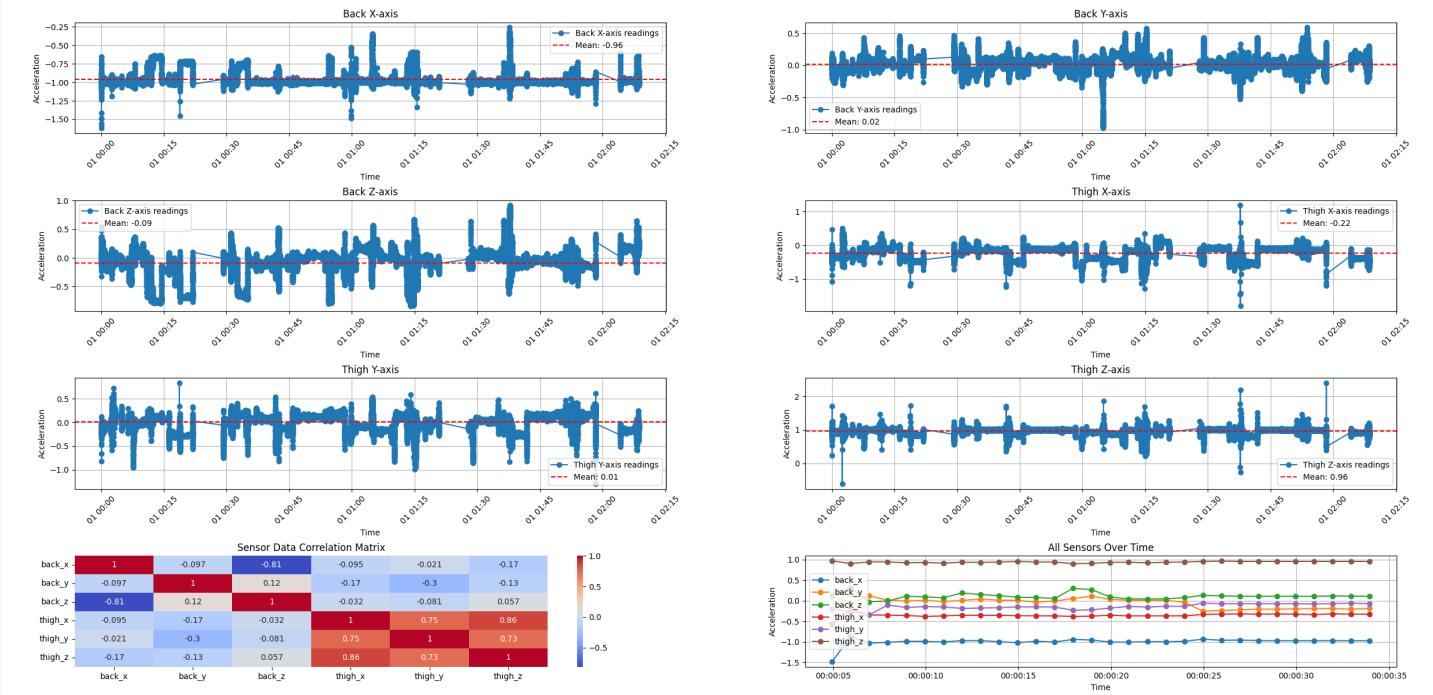
Για κάθε φυσική δραστηριότητα επιλέγουμε τον συμμετέχοντα που την εκτέλεσε για το μεγαλύτερο συνολικό χρονικό διάστημα. Για κάθε φυσική δραστηριότητα παρουσιάζουμε 8 γραφήματα. Τα 6 πρώτα δείχνουν όλες τις τιμές των αισθητήρων και την μέση τιμή του κάθε ένα. Το 7ο γράφημα παρουσιάζει ένα heatmap του πίνακα συσχέτισης των τιμών των αισθητήρων. Τιμές κοντά στο 1 δείχνει μεγάλη συσχέτιση και εξάρτηση και στο -1 αντίθετη συσχέτιση. Τιμές κοντά στο 0 υποδηλώνουν την έλλειψη συσχέτισης. Τέλος, το 8ο γράφημα παρουσιάζει ένα timestamp των 30 sec με τις τιμές των αισθητήρων ως δείγμα των μοτίβων που μπορεί να παρουσιάζουν. Ενδεικτικές εικόνες για τις δραστηριότητες Standing, Cycling Standing και Sitting.



Sensor Readings and Correlation Matrix for Activity Cycling (stand). Participant: 19



Sensor Readings and Correlation Matrix for Activity Sitting. Participant: 5



Ερώτημα 2: Εκπαίδευση Ταξινομητών

Προ-Επεξεργασία Δεδομένων

Τα δεδομένα εισόδου μας αποτελούν δεδομένα χρονοσειρών. Αυτό σημαίνει πως δε θέλουμε να ξέρουμε μόνο τις στιγμιαίες τιμές των αισθητήρων, αλλά και κάποιες τιμές προηγούμενων στιγμών.

Για αυτόν το σκοπό δημιουργούμε χρονικά «παράθυρα» όπου αποθηκεύουμε όλες τις τιμές των αισθητήρων μέσα στο χρονικό περιθώριο του κάθε παραθύρου. Παράθυρα που περιέχουν έστω κι ένα

μεγάλο χρονικό κενό (πάνω από 20ms που είναι ο ρυθμός δειγματοληψίας) **απορρίπτονται**. Για την ετικέτα του κάθε παραθύρου χρησιμοποιείται η ετικέτα **πλειοψηφίας** (αυτή δηλαδή που εμφανίζεται περισσότερες φορές στα δείγματα του κάθε παραθύρου).

Αυτός ο χωρισμός παραθύρων γίνεται με τη μέθοδο **segment_time_series**, η οποία δέχεται σαν είσοδο το **dataframe** ενός ατόμου, το **μήκος του παραθύρου** (σε ms) και το **ποσοστό επικάλυψης παραθύρων** και επιστρέφει 2 numpy arrays.

Ο 1^{ος} πίνακας αποτελεί την είσοδο των ταξινομητών και είναι 3 διαστάσεων. Το σχήμα του είναι της μορφής: (αριθμός_παραθύρων, δείγματα_ανά_παράθυρο, τιμές_αισθητήρων). Για παράδειγμα, αν ένα άτομο έχει 10.000 παράθυρα και ο αριθμός των δειγμάτων ανά παράθυρο είναι 20, το σχήμα του πίνακα θα είναι: (10.000, 20, 6), δεδομένου ότι έχουμε 6 τιμές αισθητήρων (2 αισθητήρες με x y z άξονες).

Ο 2^{ος} πίνακας περιέχει τις ετικέτες του κάθε παραθύρου και είναι 1 διάσταση, 1 ετικέτα ανά παράθυρο.

```
def segment_time_series(df, window_length_ms, overlap):
    """This function creates windows of a given length and overlap fit for time-series data.

    # Time the function
    start = time()

    # Calculate samples_per_window and step_size
    samples_per_window = window_length_ms // 20 # 20 ms intervals
    step_size = int(np.ceil(samples_per_window * (1 - overlap)))

    # Calculate the number of windows without overlap
    num_windows = len(df) // step_size - samples_per_window

    # Generate array indices for each window
    window_indices = np.arange(num_windows) * step_size
    window_end_indices = window_indices + samples_per_window

    # Extract necessary values
    timestamps = pd.to_datetime(df["timestamp"]).values
    labels = df["label"].values
    feature_data = df[FEATURES].values

    # Extract segments and labels for each window
    segments = [
        feature_data[i:end] for i, end in zip(window_indices, window_end_indices)
    ]
    segment_labels = [
        pd.Series(labels[i:end]).mode()[0]
        for i, end in zip(window_indices, window_end_indices)
    ]

    # Drop windows with maximum time difference > 20ms
    valid_windows = [
        npamax(np.diff(timestamps[i:end])).astype("timedelta64[ms]").astype(int)) <= 20
        for i, end in zip(window_indices, window_end_indices)
    ]
```

```

# Filter out invalid windows
segments = np.array(segments)[valid_windows]
segment_labels = np.array(segment_labels)[valid_windows]

end = time()

print(f"Segmenting took {round(end-start)} seconds")
print(f"Original samples: {len(df)}\n")
print(f"Step size: {step_size}")
print(f"Number of windows: {num_windows}")
print(f"Windows dropped: {num_windows - np.sum(valid_windows)}")
print(f"Valid windows: {segments.shape[0]}")

return shuffle(segments, segment_labels, random_state=7)

```

Με τη συνάρτηση `read_and_preprocess_data`, διαβάζουμε τα csv αρχεία ένα-ένα και δημιουργούμε 2 λίστες. Η 1 λίστα περιέχει τα δεδομένα εισόδου των ταξινομητών και η 2 τις ετικέτες. Οι λίστες περιέχουν 1 πίνακα ανά άτομο (άρα έχουν μήκος 22 αν διαβάσουμε όλα τα αρχεία). Επειδή αυτή η διαδικασία (πιο συγκεκριμένα η συνάρτηση `segment_timeseries` χρειάζεται αρκετό χρόνο για να ολοκληρωθεί, κατά την επεξεργασία **αποθηκεύονται** τα προ-επεξεργασμένα δεδομένα (οι 2 πίνακες) ανά χρήστη σε ένα .pkl αρχείο. Αυτά τα αρχεία μπορούν να φορτωθούν γρήγορα στο μέλλον χρησιμοποιώντας τη συνάρτηση `load_preprocessed_data`.

```

def read_and_preprocess_data(
    folder_path="harth/",
    subjects=22,
    window_length_ms=400,
    overlap=0.99,
) -> list:
    """Reads original csv files and loads them into a dataframe. Appropriate pre-processing
    is applied to the dataset. Finally, 2 lists containing the inputs and the labels of
    classifiers are returned."""

    print("Reading dataset...")
    files = os.listdir(folder_path)
    csv_files = [file for file in files if file.endswith(".csv")]
    X = []
    y = []
    # Iterate over each CSV file and read it into a DataFrame
    for idx, csv_file in enumerate(csv_files):
        if idx >= subjects:
            break
        print(f"\nPreprocessing subject {idx}...")
        file_path = os.path.join(folder_path, csv_file)
        df = pd.read_csv(file_path, quoting=csv.QUOTE_NONE)

        # For subject 20, keep every other row to be consistent with 20ms sampling of the
        # other subjects
        if idx == 20:
            df = df.iloc[::2]

        # Convert timestamp column to datetime datatype

```

```

df["timestamp"] = pd.to_datetime(df["timestamp"])
df["label"] = df["label"].map(LABEL_MAPPING)

# Apply preprocessing on data and prepare input and labels
X_temp, y_temp = segment_time_series(df, window_length_ms, overlap)

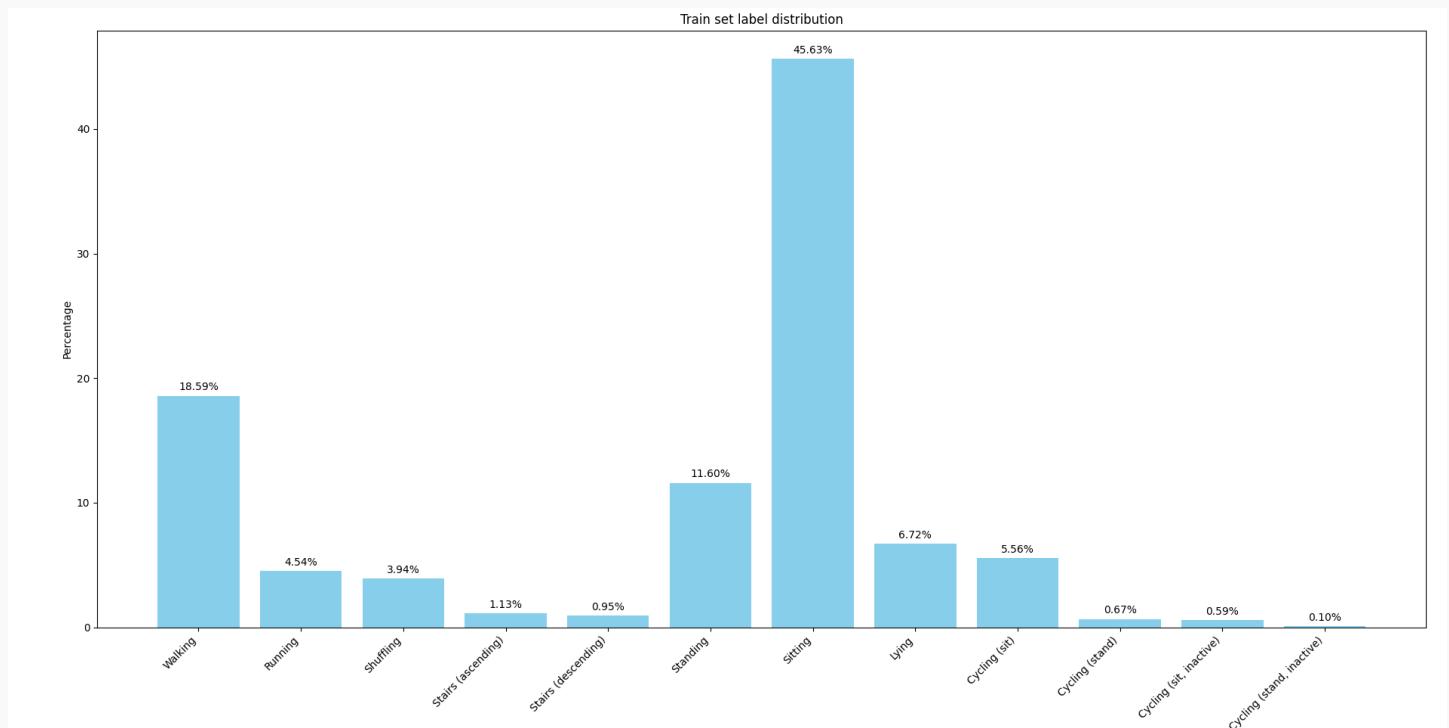
# Append current subject's input and labels to the total list
X.append(X_temp)
y.append(y_temp)

with open(PP_DATA_FP + f"subject_{idx}.pkl", "wb") as f:
    pickle.dump((X_temp, y_temp), f)

return X, y

```

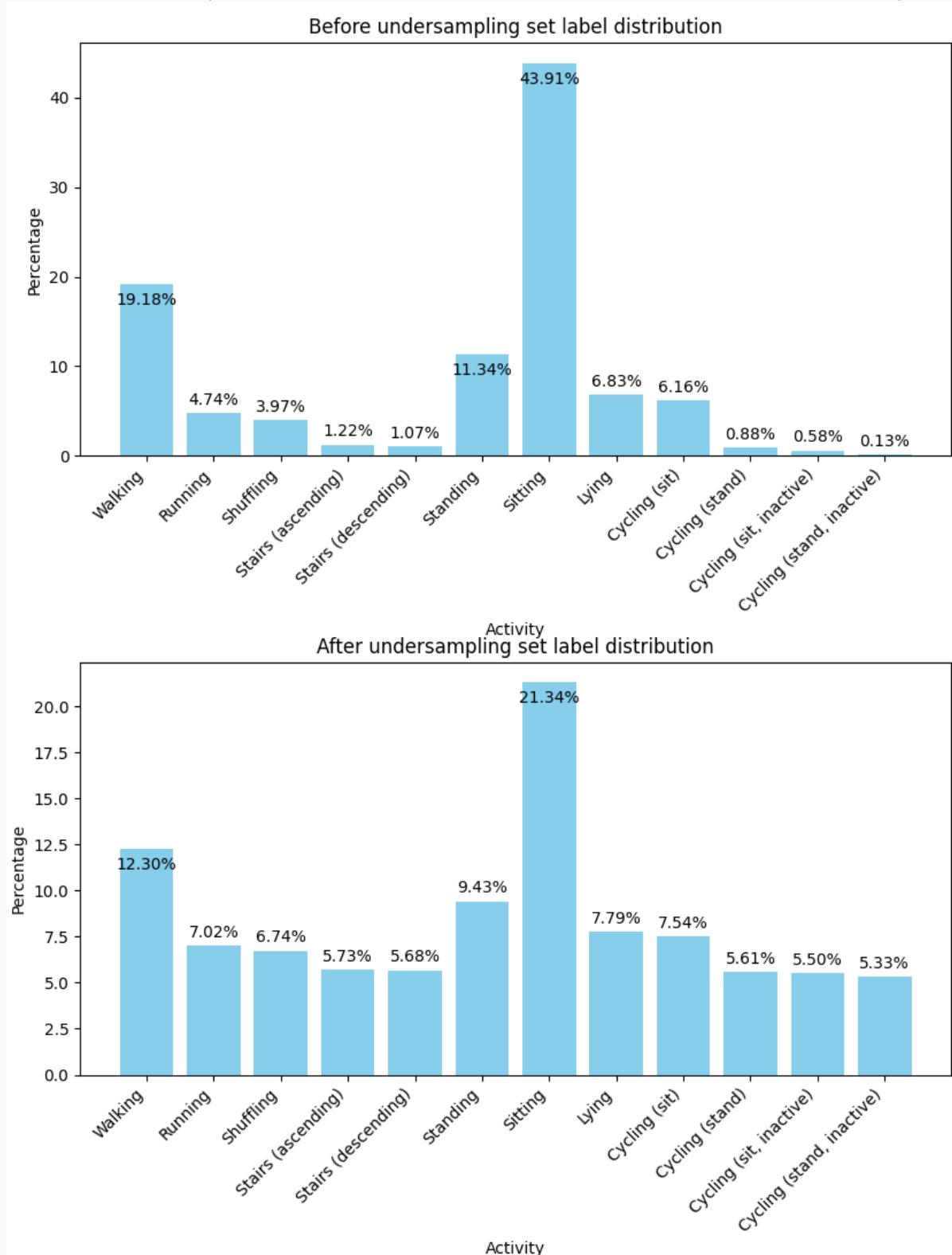
Διαχείριση μη ισορροπημένης κατανομής ετικετών συνόλου δεδομένων



Όπως φαίνεται στο παραπάνω γράφημα, η κατανομή των ετικετών στο σύνολο δεδομένων δεν είναι καθόλου ισορροπημένη. Η πιο συνηθισμένη κλάση (sitting) έχει πάνω από 400 φορές περισσότερα δείγματα από την πιο ασυνήθιστη (cycling (stand, inactive)). Για αυτό το λόγο πρέπει να προσέξουμε πώς θα εκπαιδεύσουμε τους ταξινομητές μας ώστε να μην εστιάζουν μόνο στις κλάσεις με πολλά δείγματα, αγνοώντας τις υπόλοιπες και να μην κάνουν overfit τα δεδομένα εκπαίδευσης. Για αυτό το πρόβλημα έχουμε 2 λύσεις:

Undersampling (υποδειγματοληψία)

Αρχικά, χρησιμοποιούμε την τεχνική **undersampling**, παίρνοντας λιγότερα δείγματα από τις κλάσεις με πολλά. Παρακάτω φαίνεται η κατανομή ετικετών πριν και μετά την υποδειγματοληψία:



Η υλοποίηση της υποδειγματοληψίας στον κώδικα φαίνεται παρακάτω:

```
# Calculate the number of labels
values, value_counts = np.unique(y_train, return_counts=True)

# Calculate the desired number of labels depending on us_factor
scaled_value_counts = scale_array(value_counts, us_factor)
```

```

# Select calculated desired samples per label
X_train, y_train = custom_undersample(
    X, y, dict(zip(values, scaled_value_counts))
)

```

Αρχικά υπολογίζουμε τις εμφανίσεις της κάθε ετικέτας. Για να υπολογίσουμε πόσα δείγματα να πάρουμε από κάθε ετικέτα, χρησιμοποιούμε την συνάρτηση **scale_array**, η οποία δέχεται ως είσοδο τις αρχικές εμφανίσεις και το undersampling_factor, το οποίο καθορίζει πόσο μεγάλη διαφορά να έχουν η πιο σπάνια με την πιο συνηθισμένη θέση. Ουσιαστικά, οι εμφανίσεις των ετικετών περιορίζονται στο διάστημα [min_count, min_count*us_factor]. Παρακάτω φαίνεται η συνάρτηση **scale_array**:

```

def scale_array(original_array, factor):
    """Scale the values of a 1-d numpy array to a new range defined by a factor of its
    minimum value."""

    old_min = np.min(original_array)
    old_max = np.max(original_array)
    new_max = old_min * factor

    scaled_array = ((original_array - old_min) / (old_max - old_min)) * (
        new_max - old_min
    ) + old_min
    return scaled_array.astype(int)

```

Τέλος, χρησιμοποιούμε τη συνάρτηση **custom_undersample** για να εφαρμόσουμε υποδειγματοληψία στο σύνολο δεδομένων μας, η οποία φαίνεται παρακάτω:

```

def custom_undersample(X, y, sample_count):
    """Undersample the dataset based on the specified sample counts for each class."""

    unique_classes = np.unique(y)
    resampled_indices = []

    for cls in unique_classes:
        cls_indices = np.where(y == cls)[0]
        num_samples = sample_count.get(cls, len(cls_indices))

        if num_samples > len(cls_indices):
            raise ValueError(
                f"Requested more samples ({num_samples}) than available ({len(cls_indices)})"
                f"for class {cls}"
            )

        # Randomly sample indices without replacement
        sampled_indices = np.random.choice(cls_indices, size=num_samples, replace=False)
        resampled_indices.extend(sampled_indices)

    # Shuffle the resampled indices to mix the classes
    np.random.shuffle(resampled_indices)

```

```

# Create the resampled X and y
X_resampled = X[resampled_indices]
y_resampled = y[resampled_indices]

return X_resampled, y_resampled

```

Categorical Focal Loss

Ο 2^{ος} τρόπος με τον οποίο μπορεί να αποφευχθεί το overfit του νευρωνικού δικτύου συγκεκριμένα σε ένα μη ισορροπημένο σετ εκπαίδευσης είναι η αξιοποίηση της συνάρτησης κόστους **Categorical Focal Loss**.

Η συνάρτηση αυτή δίνει μεγαλύτερη έμφαση στα δύσκολο δείγματα (δηλαδή τα λάθος ταξινομημένα) και λιγότερη έμφαση στα εύκολα. Η παράμετρος **gamma** ελέγχει το βαθμό εστίασης σε αυτά τα παραδείγματα. Παράλληλα, η παράμετρος **alpha** ελέγχει το βάρος της κάθε κλάσης. Για $\alpha=1$ η απώλεια είναι ίδια για όλες τις κατηγορίες. Για τον υπολογισμό της χρησιμοποιούμε τη συνάρτηση **calculate_alpha**, η οποία παίρνει σαν είσοδο την κατανομή των ετικετών στο σετ εκπαίδευσης και επιστρέφει την παράμετρο alpha, η οποία είναι μία λίστα με μία τιμή (βάρος) ανά κλάση. Παρακάτω φαίνεται η συνάρτηση αυτή:

```

def calculate_alpha(label_distribution: list[float]) -> list[float]:
    """Calculate the alpha parameter for each label based on the label distribution."""

    # Convert percentages to proportions (sum should be 1)
    total = sum(label_distribution)
    proportions = [x / total for x in label_distribution]

    # Calculate the inverse of the proportions
    inverse_proportions = [1 / p if p > 0 else 0 for p in proportions]

    # Normalize the inverses to sum to 1
    sum_inverse_proportions = sum(inverse_proportions)
    alpha = [x / sum_inverse_proportions for x in inverse_proportions]

    return alpha

```

Εκπαίδευση & Αξιολόγηση Ταξινομητών

Για την εκπαίδευση και την αξιολόγηση των ταξινομητών, έχουμε χρησιμοποιήσει 2 μεθόδους.

Στην 1^η και πιο απλή περίπτωση, χρησιμοποιούμε **όλο το σύνολο δεδομένων**, το ανακατεύουμε και το χωρίζουμε σε δεδομένα εκπαίδευσης και δεδομένα αξιολόγησης (X, y train και X, y test).

Στη 2^η περίπτωση, έχουμε εφαρμόσει τη μέθοδο **Leave-One-Out Cross-Validation**, κατά την οποία χρησιμοποιούμε τα 21 από τα 22 άτομα κατά την εκπαίδευση και το τελευταίο για την αξιολόγηση. Αυτό γίνεται 22 φορές και στο τέλος προσθέτουμε τις προβλέψεις του δικτύου με τις αντίστοιχες πραγματικές και παρουσιάζουμε τη **συνολική** αναφορά κατηγοριοποίησης. Αυτό το έχουμε κάνει επειδή λόγω μεγάλου overlap στις χρονοσειρές εισόδου, είχαμε υποψίες ότι το νευρωνικό δίκτυο έκανε overfit. Επιλέξαμε μεγάλο overlap γιατί αν χρησιμοποιούσαμε μικρότερο θα χάναμε πολλά δείγματα από τις σπάνιες κλάσεις.

Εκπαίδευση σε όλο το σύνολο δεδομένων

Για την εκπαίδευση σε όλο το σύνολο δεδομένων χρησιμοποιούμε τη συνάρτηση **train_and_evaluate_models_no_cv**:

```

def train_and_evaluate_models_no_cv(
    X: list[np.array],
    y: list[np.array],
    epochs=20,

```

```

batch_size=1_000,
n_trees=100,
undersample=True,
us_factor=4,
individuals=22,
):
    """Train and evaluate classifiers without using Leave-One-Out Cross-Validation"""

    # Concatenate the data lists into 2 numpy arrays
    X = np.concatenate([X_ind for X_ind in X[:individuals]], axis=0)
    y = np.concatenate([y_ind for y_ind in y[:individuals]], axis=0)

    # Split the dataset to train and test data
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.3, random_state=7, shuffle=True
    )

```

Αρχικά, μετατρέπουμε τις λίστες σε ενιαίους πίνακες και χρησιμοποιώντας τη συνάρτηση `train_test_split` του module scikit-learn χωρίζουμε τα δεδομένα μας σε σετ εκπαίδευσης και αξιολόγησης. Στη συνέχεια, ανάλογα και με τις παραμέτρους της συνάρτησης, εφαρμόζουμε υποδειγματοληψία.

```

# Perform undersampling on training data
if undersample:
    # Get original label distribution
    y_train_original_distr = get_label_distribution(y_train)

    # Calculate the number of labels
    values, value_counts = np.unique(y_train, return_counts=True)

    # Calculate the desired number of labels depending on us_factor
    scaled_value_counts = scale_array(value_counts, us_factor)

    # Select calculated desired samples per label
    X_train, y_train = custom_undersample(
        X, y, dict(zip(values, scaled_value_counts))
    )
    # Get label distribution after undersampling
    y_train_us_distr = get_label_distribution(y_train)

    # Print before and after undersampling distributions
    plot_distribution_histograms(
        [y_train_original_distr, y_train_us_distr],
        set_names=["Before undersampling", "After undersampling"],
    )

```

Στη συνέχεια, υπολογίζουμε την παράμετρο `alpha` ανάλογα με την κατανομή των ετικετών στο σετ εκπαίδευσης και εκπαιδεύουμε το μοντέλο. Επειδή η συνάρτηση `kostouς` απαιτεί one-hot αναπαράσταση των ετικετών χρησιμοποιούμε τη συνάρτηση `np.eye` για να τις μετασχηματίσουμε. Έπειτα, εμφανίζουμε τις γραφικές παραστάσεις με όλες τις μετρικές κατά την εκπαίδευση ως προς το χρόνο και τέλος αξιολογούμε το μοντέλο αξιοποιώντας τη συνάρτηση `evaluateClassifier`.

```

##### Neural Network (tf Sequential model)
print("Training Neural Network classifier...")

```

```

num_classes = np.max(y_train) + 1
y_train_distr = get_label_distribution(y_train)
alpha = calculate_alpha(y_train_distr)
nn_model = get_sequential_model((X_train.shape[1:]), num_classes, alpha)
nn_history = train_nn_classifier(
    nn_model,
    X_train,
    np.eye(num_classes)[y_train],
    epochs,
    batch_size,
)
plot_history(nn_history)
evaluateClassifier(nn_model, X_test, np.eye(num_classes)[y_test])

```

Στη συνάρτηση `evaluateClassifier` αρχικά κάνουμε τις κατάλληλες προβλέψεις ανάλογα τον τύπο του μοντέλου και εμφανίζουμε τις **μετρικές κατηγοριοποίησης** και το **μητρώο σύγχυσης** (confusion matrix).

```

def evaluateClassifier(
    cl: RandomForestClassifier | Sequential | GaussianNB,
    X_test: np.array,
    y_test: np.array,
) -> None:
    """Evaluates a model and prints important classification metrics."""

    print(f"\nEvaluating {type(cl)} classifier...")

    if type(cl) == RandomForestClassifier or type(cl) == GaussianNB:
        y_pred_proba = cl.predict_proba(X_test)
        y_true = y_test
    elif type(cl) == Sequential:
        y_pred_proba = cl.predict(X_test)
        y_true = np.argmax(y_test, axis=1)

    y_pred = np.argmax(y_pred_proba, axis=1)
    print_important_classification_metrics(y_true, y_pred, y_pred_proba)

```

Η συνάρτηση `print_important_classification_metrics` φαίνεται παρακάτω:

```

def print_important_classification_metrics(
    y_true, y_pred, y_pred_proba=None, filename=""
):
    # Calculate evaluation metrics
    if y_pred_proba is not None:
        auc = roc_auc_score(y_true, y_pred_proba, multi_class="ovr")
        print(f"AUC: {auc:.2f}")
    accuracy = accuracy_score(y_true, y_pred)
    class_rep = classification_report(
        y_true,
        y_pred,
        target_names=LABEL_LIST,
        zero_division=0,
    )

```

```

print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:\n", class_rep)
display_confusion_matrix(y_true, y_pred, None, LABEL_LIST, filename)

```

Αντίστοιχη διαδικασία γίνεται και για τους άλλους 2 ταξινομητές:

```

##### Random Forest (sklearn)
print("Training Random Forest classifier...")
# Flatten the input data
X_train = flatten_array(X_train)
X_test = flatten_array(X_test)

rf_classifier = RandomForestClassifier(
    n_estimators=n_trees,
    random_state=7,
    n_jobs=-1,
    verbose=2,
    # class_weight="balanced",
    max_depth=5,
)
rf_classifier.fit(X_train, y_train)
evaluateClassifier(
    rf_classifier,
    X_test,
    y_test,
)

##### Gaussian Naive Bayes
print("Training Gaussian Naive Bayes classifier...")
gnb_classifier = GaussianNB().fit(X_train, y_train)
evaluateClassifier(gnb_classifier, X_test, y_test)

```

Επειδή οι ταξινομητές *RandomForestClassifier* και *Gaussian Naïve Bayes* δε δέχονται ως είσοδο τρισδιάστατους πίνακα, τροποποιούμε τις εισόδους αντίστοιχα πριν την εκπαίδευση.

Leave-One-Out Cross-Validation

Έχουμε δημιουργήσει μία συνάρτηση για την εκπαίδευση και την αξιολόγηση ταξινομητών με loo cv. Η συνάρτηση είναι κοινή και για τα 3 είδη ταξινομητών και ανάλογα με το όρισμα *cl_type* εκπαιδεύεται και αξιολογείται ο αντίστοιχος ταξινομητής.

```

def custom_loo_cv(
    cl_type,
    X,
    y,
    epochs=None,
    batch_size=None,
    n_trees=None,
    undersample=True,
    us_factor=4,
):
    """Given a model type (Sequential/RandomForest/BayesianNetworks), train and evaluate the
    model using Leave-One-Out Cross-Validation (all but one individuals are used during training

```

and the remaining one is used for testing). Finally, the total classification report and the confusion matrix are displayed.””

```
# Initialize storage for true and predicted labels
y_true_total = np.array([], dtype=int)
y_pred_total = np.array([], dtype=int)
y_pred_proba_total = np.empty((0, 12), dtype=float)
```

Στη βασική επανάληψη της συνάρτησης, διατρέχονται όλοι οι συμμετέχοντες που θα χρησιμοποιηθούν για την αξιολόγηση του μοντέλου και οι υπόλοιποι χρησιμοποιούνται για την εκπαίδευση.

```
# Leave-One-Out Cross-Validation
for i in range(len(X)):
    print(f"Training model {i}...")
    # Prepare the training and test sets
    X_train = np.concatenate([X[j] for j in range(len(X)) if j != i], axis=0)
    y_train = np.concatenate([y[j] for j in range(len(y)) if j != i], axis=0)
    if undersample:
        values, value_counts = np.unique(y_train, return_counts=True)
        scaled_value_counts = scale_array(value_counts, us_factor)
        X_train, y_train = custom_undersample(
            X_train, y_train, dict(zip(values, scaled_value_counts)))
    )
    X_test, y_test = shuffle(X[i], y[i], random_state=7)
```

Στη συνέχεια, γίνεται η αντίστοιχη διαδικασία προετοιμασίας εισόδων/εξόδων που γίνεται και με την 1^η, απλή περίπτωση εκπαίδευσης/αξιολόγησης, ανάλογα με τον τύπο μοντέλου. Έπειτα, δημιουργείται το αντίστοιχο μοντέλο, εκπαιδεύεται και αξιολογείται στο σύνολο αξιολόγησης. Τέλος, οι προβλέψεις μαζί με τις πραγματικές τιμές των ετικετών αποθηκεύονται και το μοντέλο «ξεσκαρτάρεται» ουσιαστικά, αφού δεν αποθηκεύεται κάπου.

```
# Save true and predicted labels
y_true_total = np.concatenate([y_true_total, y_true])
y_pred_total = np.concatenate([y_pred_total, y_pred])
y_pred_proba_total = np.concatenate([y_pred_proba_total, y_pred_proba])
```

Μετά το πέρας της βασικής επανάληψης, γίνεται η συνολική αξιολόγηση των μοντέλων.

```
# Collect and print total classification report
print("Total classification report:")
print_important_classification_metrics(
    y_true_total, y_pred_total, y_pred_proba_total, filename=cl_type
)
```

Ο χρόνος που χρειάζεται για την παραπάνω διαδικασία είναι αρκετά μεγάλος, αλλά έτσι γίνεται και πιο αντικειμενική αξιολόγηση των ταξινομητών.

Αρχιτεκτονική Νευρωνικού Δικτύου

Για την αρχιτεκτονική του νευρωνικού δικτύου επιλέξαμε να χρησιμοποιήσουμε **Long Short Term Memory (LSTM)** επίπεδα, τα οποία ανήκουν στην οικογένεια των **Recursive Neural Networks (RNNs)** επειδή έχουν καλή απόδοση στα δεδομένων χρονοσειρών. Η ακριβής αρχιτεκτονική του δικτύου φαίνεται στη συνάρτηση **get_sequential_model** παρακάτω:

```
def get_sequential_model(input_shape, output_shape, alpha) -> Sequential:
    """Returns a compiled sequential model for classification."""
```

```

model = Sequential()
model.add(Input(shape=(input_shape)))
model.add(LSTM(20))
model.add(Dropout(0.1))
model.add(Dense(output_shape, activation="softmax"))
model.summary()
model.compile(
    optimizer=Adam(),
    loss=CategoricalFocalCrossentropy(alpha=alpha, gamma=4),
    metrics=["accuracy", Precision(), Recall(), AUC(), F1Score()],
)
return model

```

Επιλέξαμε μία σχετικά απλή αρχιτεκτονική με 20 επίπεδα LSTM, ένα Dropout για να μειώσουμε το overfit και τέλος το επίπεδο εξόδου. Το επίπεδο εξόδου έχει αριθμό κόμβο ίσο με τις συνολικές κλάσεις των δεδομένων μας (12). Χρησιμοποιείται η συνάρτηση ενεργοποίησης softmax, ώστε για κάθε δείγμα να προβλέπεται η πιθανότητα του να ανήκει σε κάθε κλάση (το άθροισμα και των 12 κόμβων είναι πάντα 1). Επιλέγοντας τον κόμβο με τη μεγαλύτερη πιθανότητα γίνεται ουσιαστικά η πρόβλεψη της δραστηριότητας. Τέλος, κάνουμε compile το μοντέλο επιλέγοντας τις κατάλληλες παραμέτρους που έχουν αναλυθεί παραπάνω και τις μετρικές που θα φαίνονται κατά την εκπαίδευση.

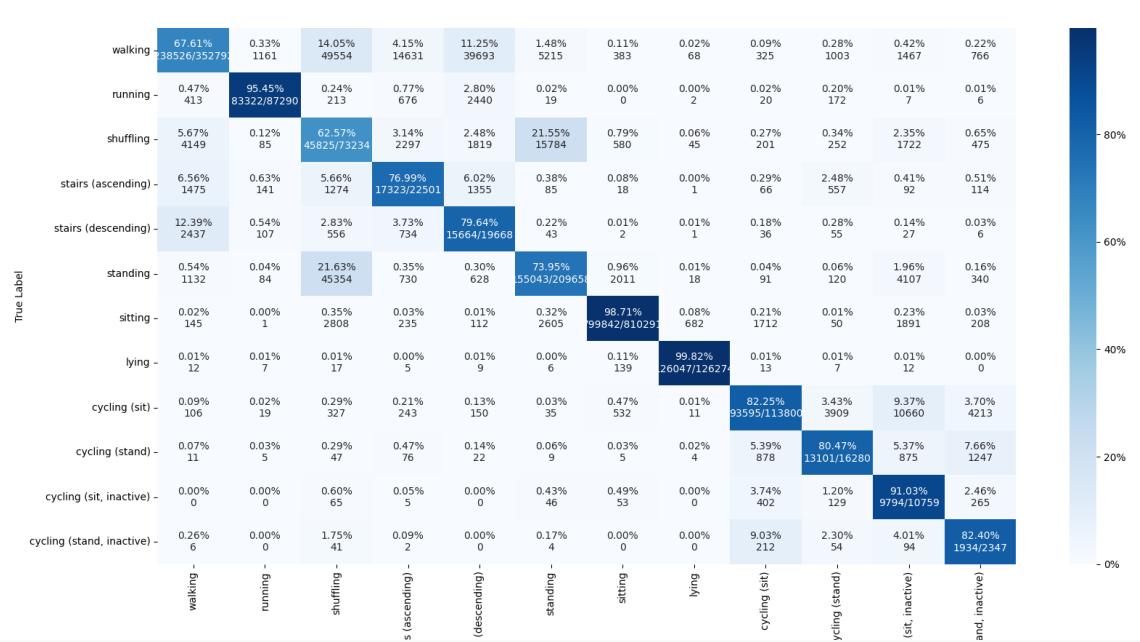
Αποτελέσματα

1^η Μέθοδος (Όλο το σύνολο δεδομένων)

Με υποδειγματοληψία (Undersampling)

Νευρωνικό Δίκτυο

Evaluating <class 'keras.src.models.sequential.Sequential'> classifier...				
	precision	recall	f1-score	support
walking	0.96	0.68	0.79	352792
running	0.98	0.95	0.97	87290
shuffling	0.31	0.63	0.42	73234
stairs (ascending)	0.47	0.77	0.58	22501
stairs (descending)	0.25	0.80	0.38	19668
standing	0.87	0.74	0.80	209658
sitting	1.00	0.99	0.99	810291
lying	0.99	1.00	1.00	126274
cycling (sit)	0.96	0.82	0.89	113800
cycling (stand)	0.67	0.80	0.73	16280
cycling (sit, inactive)	0.32	0.91	0.47	10759
cycling (stand, inactive)	0.20	0.82	0.32	2347
accuracy			0.87	1844894
macro avg	0.67	0.83	0.70	1844894
weighted avg	0.92	0.87	0.88	1844894



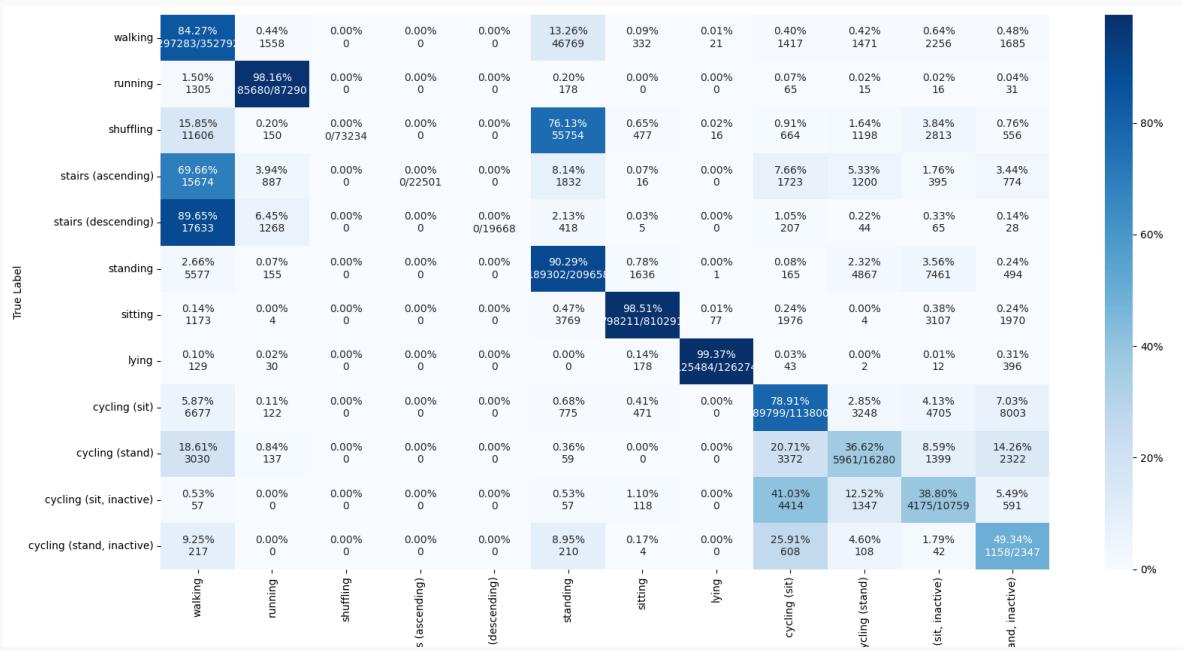
Random Forest

AUC: 0.96

Accuracy: 0.87

Classification Report:

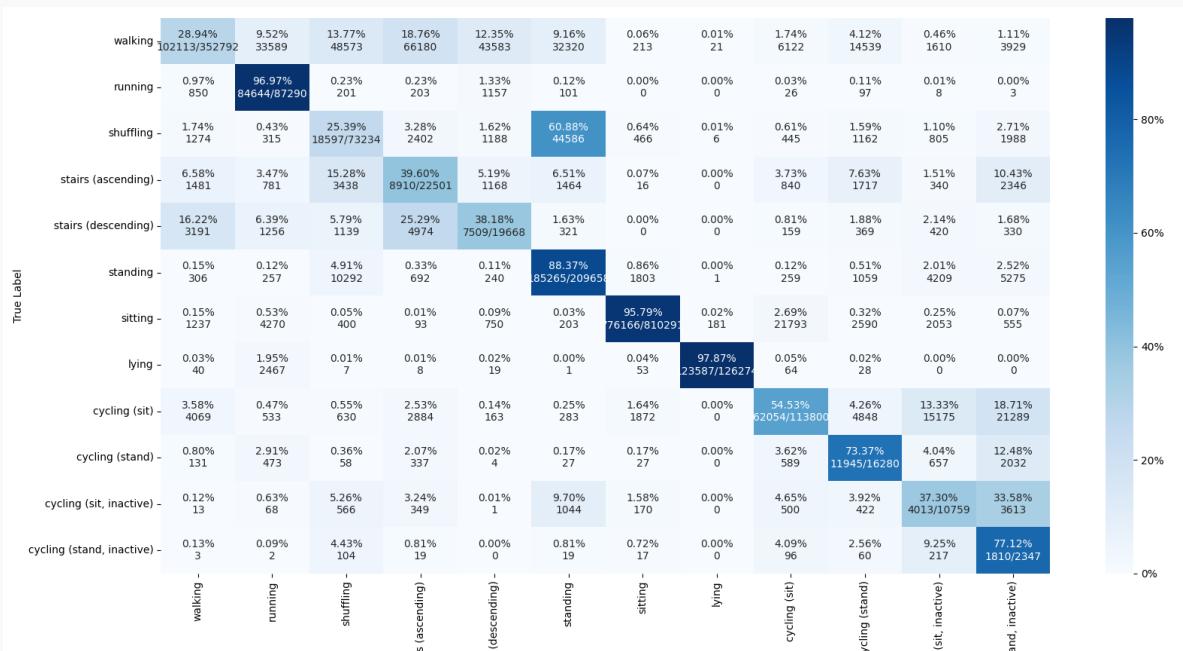
	precision	recall	f1-score	support
walking	0.82	0.84	0.83	352792
running	0.95	0.98	0.97	87290
shuffling	0.00	0.00	0.00	73234
stairs (ascending)	0.00	0.00	0.00	22501
stairs (descending)	0.00	0.00	0.00	19668
standing	0.63	0.90	0.74	209658
sitting	1.00	0.99	0.99	810291
lying	1.00	0.99	1.00	126274
cycling (sit)	0.86	0.79	0.82	113800
cycling (stand)	0.31	0.37	0.33	16280
cycling (sit, inactive)	0.16	0.39	0.22	10759
cycling (stand, inactive)	0.06	0.49	0.11	2347
accuracy			0.87	1844894
macro avg	0.48	0.56	0.50	1844894
weighted avg	0.84	0.87	0.85	1844894



Bayesian Network

```
Evaluating <class 'sklearn.naive_bayes.GaussianNB'> classifier...
AUC: 0.95
Accuracy: 0.75
Classification Report:
              precision    recall   f1-score  support
walking          0.89      0.29      0.44     352792
running          0.66      0.97      0.78     87290
shuffling         0.22      0.25      0.24     73234
stairs (ascending)  0.10      0.40      0.16     22501
stairs (descending) 0.13      0.38      0.20     19668
standing          0.70      0.88      0.78     209658
sitting           0.99      0.96      0.98     810291
lying             1.00      0.98      0.99     126274
cycling (sit)      0.67      0.55      0.60     113800
cycling (stand)     0.31      0.73      0.43     16280
cycling (sit, inactive) 0.14      0.37      0.20     10759
cycling (stand, inactive) 0.04      0.77      0.08     2347

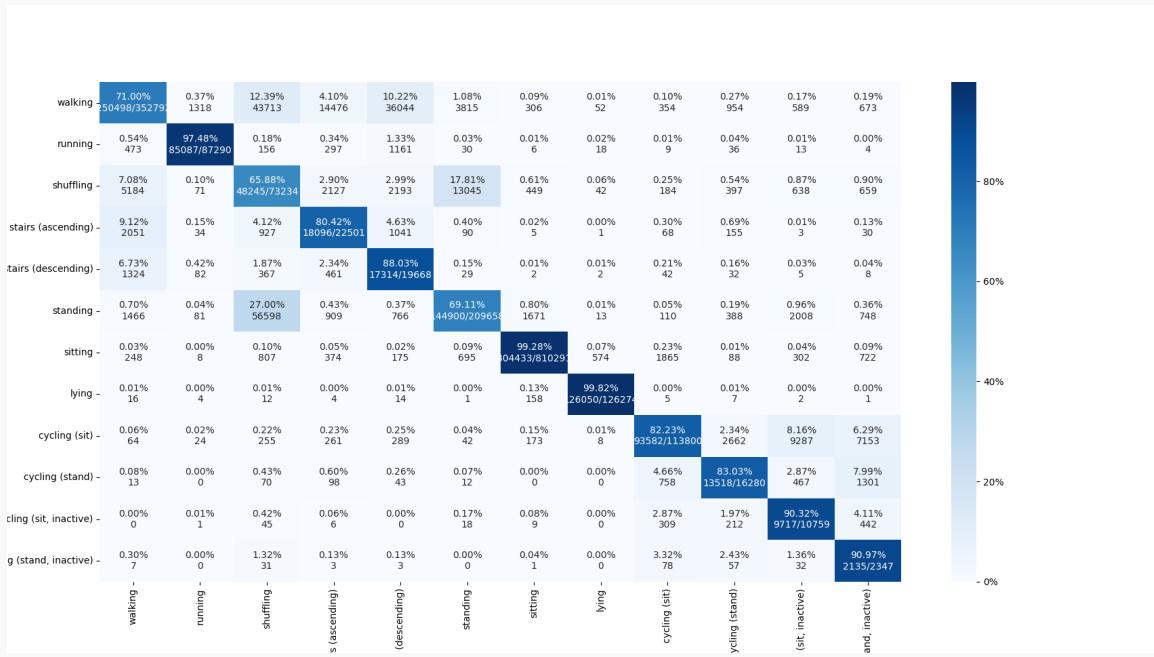
              accuracy                   0.75     1844894
macro avg        0.49      0.63      0.49     1844894
weighted avg      0.84      0.75      0.76     1844894
```



Χωρίς Υποδειγματοληψία

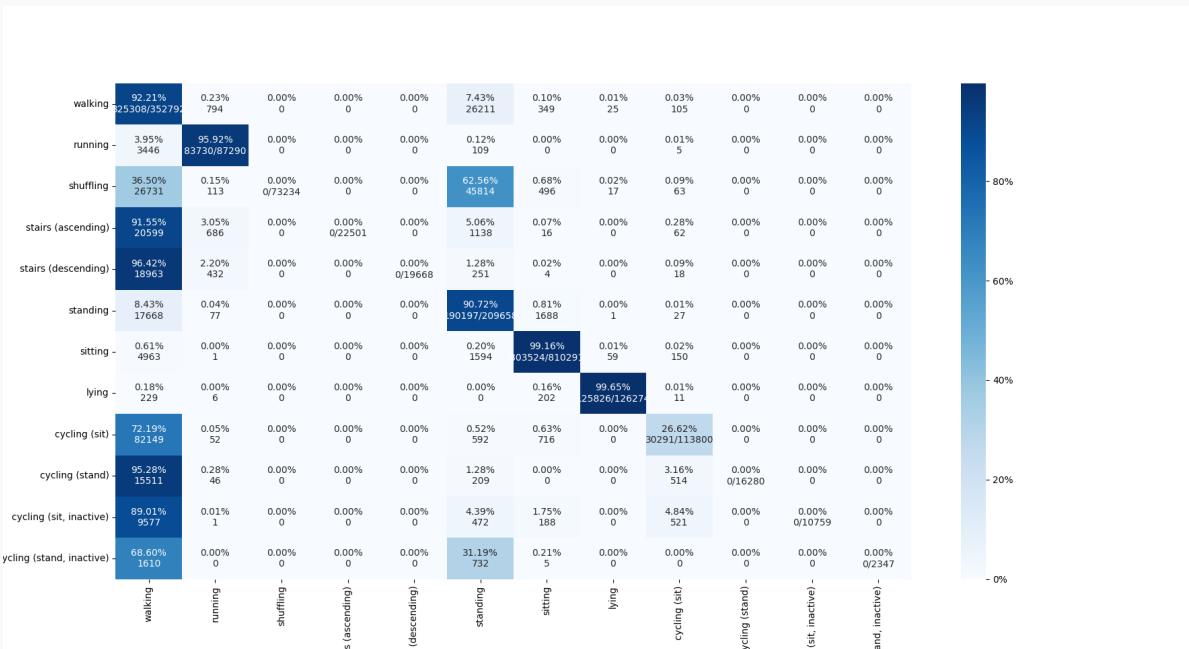
Νευρωνικό Δίκτυο

Evaluating <class 'keras.src.models.sequential.Sequential'> classifier...										
	precision	recall	f1-score	support						
walking	0.96	0.71	0.82	352792						
running	0.98	0.97	0.98	87290						
shuffling	0.32	0.66	0.43	73234						
stairs (ascending)	0.49	0.80	0.61	22501						
stairs (descending)	0.29	0.88	0.44	19668						
standing	0.89	0.69	0.78	209658						
sitting	1.00	0.99	0.99	810291						
lying	0.99	1.00	1.00	126274						
cycling (sit)	0.96	0.82	0.89	113800						
cycling (stand)	0.73	0.83	0.78	16280						
cycling (sit, inactive)	0.42	0.90	0.57	10759						
cycling (stand, inactive)	0.15	0.91	0.26	2347						
accuracy			0.87	1844894						
macro avg	0.68	0.85	0.71	1844894						
weighted avg	0.93	0.87	0.89	1844894						



Random Forest

AUC: 0.96										
Accuracy: 0.84										
Classification Report:										
	precision	recall	f1-score	support						
walking	0.62	0.92	0.74	352792						
running	0.97	0.96	0.97	87290						
shuffling	0.00	0.00	0.00	73234						
stairs (ascending)	0.00	0.00	0.00	22501						
stairs (descending)	0.00	0.00	0.00	19668						
standing	0.71	0.91	0.80	209658						
sitting	1.00	0.99	0.99	810291						
lying	1.00	1.00	1.00	126274						
cycling (sit)	0.95	0.27	0.42	113800						
cycling (stand)	0.00	0.00	0.00	16280						
cycling (sit, inactive)	0.00	0.00	0.00	10759						
cycling (stand, inactive)	0.00	0.00	0.00	2347						
accuracy			0.84	1844894						
macro avg	0.44	0.42	0.41	1844894						
weighted avg	0.81	0.84	0.81	1844894						



Bayesian Network

Classification Report:				
	precision	recall	f1-score	support
walking	0.89	0.36	0.51	352792
running	0.66	0.97	0.78	87290
shuffling	0.22	0.26	0.24	73234
stairs (ascending)	0.10	0.39	0.17	22501
stairs (descending)	0.18	0.31	0.23	19668
standing	0.70	0.89	0.78	209658
sitting	0.99	0.96	0.98	810291
lying	1.00	0.98	0.99	126274
cycling (sit)	0.66	0.58	0.62	113800
cycling (stand)	0.33	0.73	0.46	16280
cycling (sit, inactive)	0.15	0.38	0.22	10759
cycling (stand, inactive)	0.05	0.76	0.09	2347
accuracy			0.77	1844894
macro avg	0.49	0.63	0.50	1844894
weighted avg	0.84	0.77	0.78	1844894

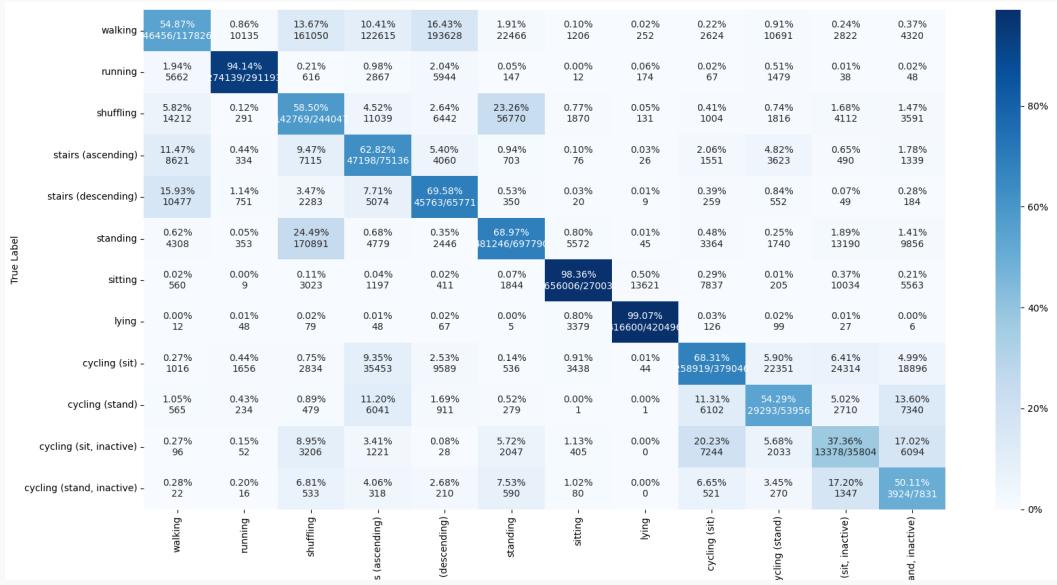


2^η Μέθοδος (Leave-One-Out Cross-Validation)

Με υποδειγματοληψία (Undersampling)

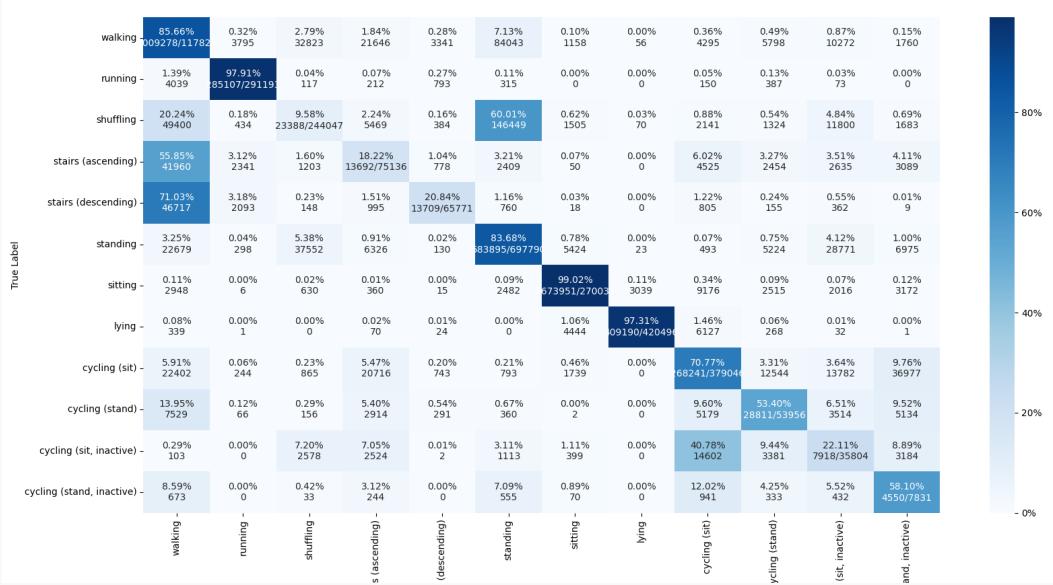
Νευρωνικό Δίκτυο

Total classification report:										
Classification Report:										
	precision	recall	f1-score	support						
walking	0.93	0.55	0.69	1178265						
running	0.95	0.94	0.95	291193						
shuffling	0.29	0.59	0.39	244047						
stairs (ascending)	0.20	0.63	0.30	75136						
stairs (descending)	0.17	0.70	0.27	65771						
standing	0.85	0.69	0.76	697790						
sitting	0.99	0.98	0.99	2700310						
lying	0.97	0.99	0.98	420496						
cycling (sit)	0.89	0.68	0.77	379046						
cycling (stand)	0.40	0.54	0.46	53956						
cycling (sit, inactive)	0.18	0.37	0.25	35804						
cycling (stand, inactive)	0.06	0.50	0.11	7831						
accuracy			0.82	6149645						
macro avg	0.57	0.68	0.58	6149645						
weighted avg	0.90	0.82	0.84	6149645						



Random Forest

Total classification report:										
Classification Report:										
	precision	recall	f1-score	support						
walking	0.84	0.86	0.85	1178265						
running	0.97	0.98	0.97	291193						
shuffling	0.24	0.10	0.14	244047						
stairs (ascending)	0.18	0.18	0.18	75136						
stairs (descending)	0.68	0.21	0.32	65771						
standing	0.71	0.84	0.77	697790						
sitting	0.99	0.99	0.99	2700310						
lying	0.99	0.97	0.98	420496						
cycling (sit)	0.85	0.71	0.77	379046						
cycling (stand)	0.46	0.53	0.49	53956						
cycling (sit, inactive)	0.10	0.22	0.13	35804						
cycling (stand, inactive)	0.07	0.58	0.12	7831						
accuracy			0.87	6149645						
macro avg	0.59	0.60	0.56	6149645						
weighted avg	0.87	0.87	0.86	6149645						



Bayesian Network

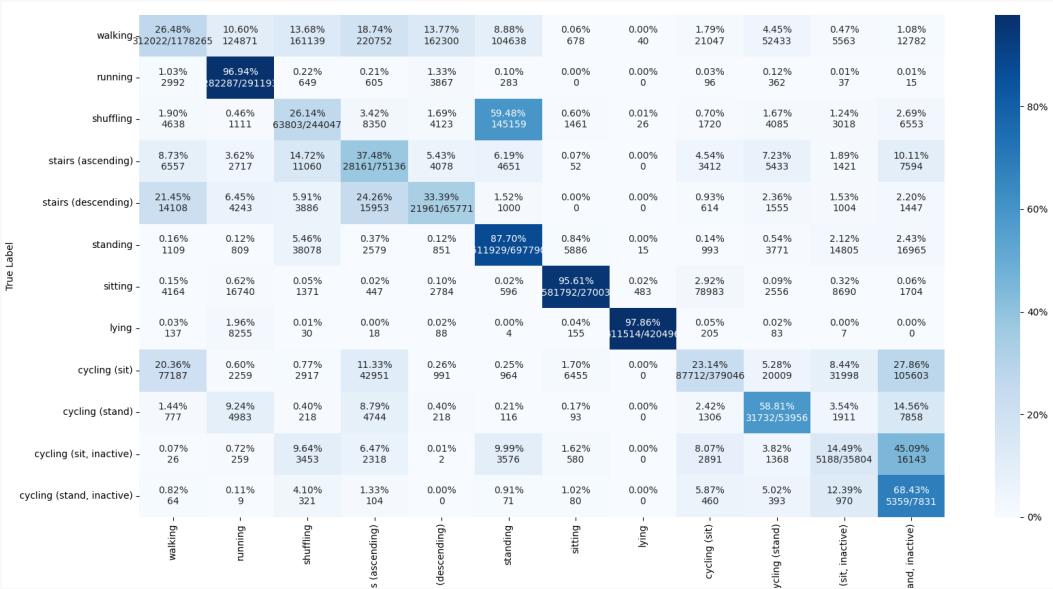
Total classification report:

AUC: 0.93

Accuracy: 0.72

Classification Report:

	precision	recall	f1-score	support
walking	0.74	0.26	0.39	1178265
running	0.63	0.97	0.76	291193
shuffling	0.22	0.26	0.24	244047
stairs (ascending)	0.09	0.37	0.14	75136
stairs (descending)	0.11	0.33	0.16	65771
standing	0.70	0.88	0.78	697790
sitting	0.99	0.96	0.97	2700310
lying	1.00	0.98	0.99	420496
cycling (sit)	0.44	0.23	0.30	379046
cycling (stand)	0.26	0.59	0.36	53956
cycling (sit, inactive)	0.07	0.14	0.09	35804
cycling (stand, inactive)	0.03	0.68	0.06	7831
				accuracy
				0.72
				6149645
				macro avg
				0.44
				6149645
				weighted avg
				0.73
				6149645



Χωρίς Υποδειγματοληψία

Νευρωνικό Δίκτυο

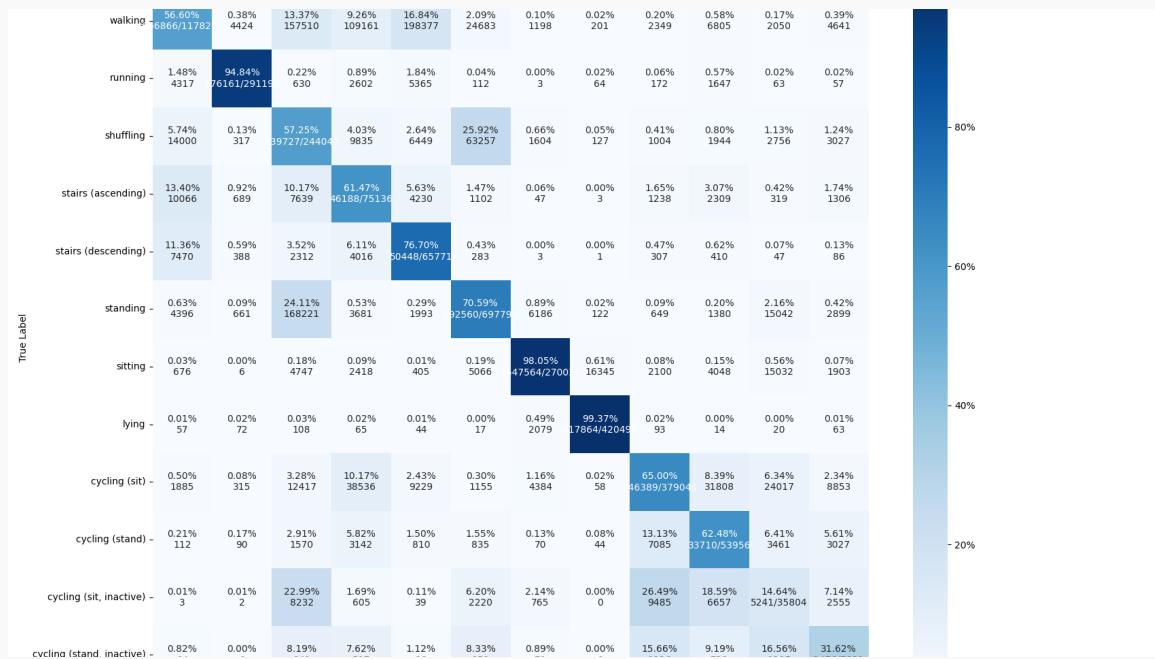
Total classification report:

AUC: 0.95

Accuracy: 0.77

Classification Report:

	precision	recall	f1-score	support
walking	0.87	0.45	0.60	1178265
running	0.97	0.93	0.95	291193
shuffling	0.27	0.55	0.36	244047
stairs (ascending)	0.16	0.54	0.25	75136
stairs (descending)	0.12	0.60	0.21	65771
standing	0.81	0.68	0.74	697790
sitting	0.99	0.94	0.97	2700310
lying	0.94	0.99	0.97	420496
cycling (sit)	0.88	0.59	0.71	379046
cycling (stand)	0.36	0.60	0.45	53956
cycling (sit, inactive)	0.15	0.47	0.23	35804
cycling (stand, inactive)	0.04	0.58	0.07	781
accuracy			0.77	6149645
macro avg	0.55	0.66	0.54	6149645
weighted avg	0.88	0.77	0.80	6149645



Random Forest

Total classification report:

AUC: 0.97

Accuracy: 0.79

Classification Report:

	precision	recall	f1-score	support
walking	0.96	0.48	0.64	1178265
running	0.97	0.98	0.98	291193
shuffling	0.25	0.28	0.26	244047
stairs (ascending)	0.15	0.52	0.23	75136
stairs (descending)	0.14	0.81	0.24	65771
standing	0.75	0.81	0.78	697790
sitting	1.00	0.98	0.99	2700310
lying	0.99	0.98	0.99	420496
cycling (sit)	0.89	0.49	0.63	379046
cycling (stand)	0.43	0.65	0.52	53956
cycling (sit, inactive)	0.13	0.33	0.18	35804
cycling (stand, inactive)	0.04	0.66	0.07	7831
accuracy			0.79	6149645
macro avg	0.56	0.66	0.54	6149645
weighted avg	0.89	0.79	0.82	6149645

Bayesian Network

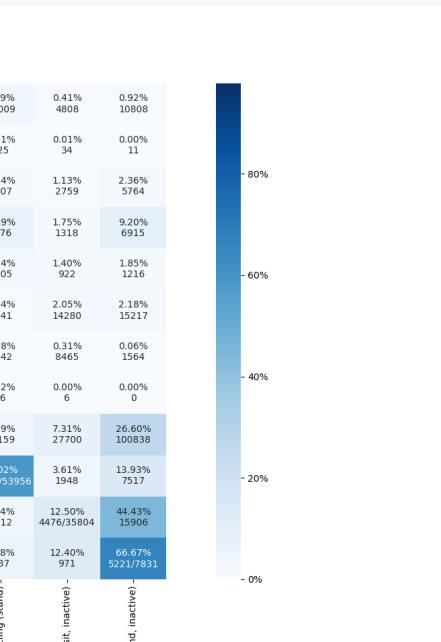
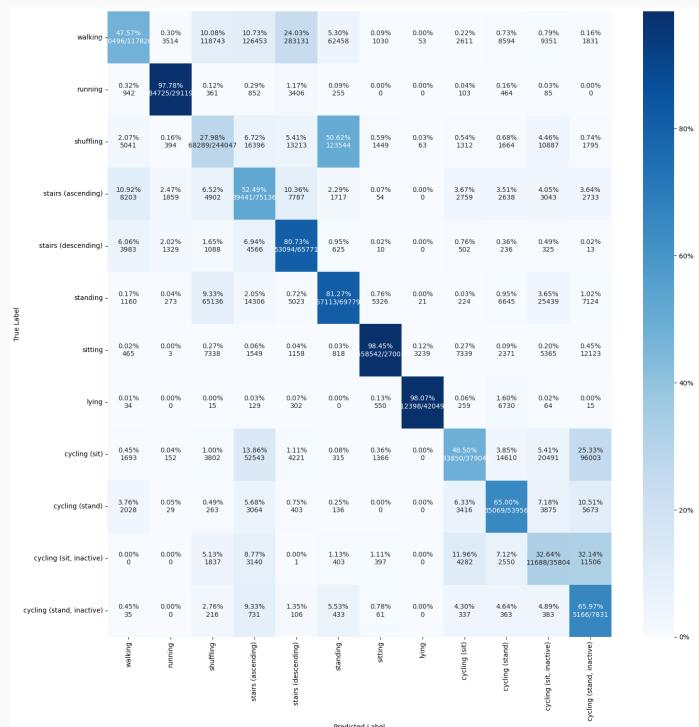
Total classification report:

AUC: 0.93

Accuracy: 0.74

Classification Report:

	precision	recall	f1-score	support
walking	0.75	0.33	0.45	1178265
running	0.63	0.97	0.76	291193
shuffling	0.22	0.26	0.23	244047
stairs (ascending)	0.09	0.36	0.14	75136
stairs (descending)	0.13	0.26	0.17	65771
standing	0.70	0.88	0.78	697790
sitting	0.99	0.96	0.97	2700310
lying	1.00	0.98	0.99	420496
cycling (sit)	0.46	0.26	0.33	379046
cycling (stand)	0.27	0.59	0.37	53956
cycling (sit, inactive)	0.07	0.13	0.09	35804
cycling (stand, inactive)	0.03	0.67	0.06	7831
accuracy			0.74	6149645
macro avg	0.44	0.55	0.45	6149645
weighted avg	0.80	0.74	0.74	6149645



Συμπεράσματα

Undersampling

Παρατηρούμε ότι γενικά η τεχνική της υποδειγματοληψίας βοηθάει στην καλύτερη αναγνώριση κλάσεων που εμφανίζονται λιγότερο στο σύνολο δεδομένων.

Στο νευρωνικό δίκτυο η επιρροή του δεν είναι τόσο μεγάλη επειδή ήδη έχουμε χρησιμοποιήσει κατάλληλη συνάρτηση κόστους (Categorical Focal Loss), ενώ για να βγάλει αντίστοιχα αποτελέσματα με την εκπαίδευση χωρίς υποδειγματοληψία έπρεπε να τροποποιηθεί αντίστοιχα και το Batch Size. Συγκεκριμένα, έπρεπε να το μειώσουμε σε 10 από 1000 για να έχουμε παρόμοια αποτελέσματα στον ίδιο αριθμό εποχών (20), γεγονός που σημαίνει ότι ο χρόνος εκπαίδευσης δε μειώθηκε.

Αντιθέτως, στον ταξινομητή Random Forest, είδαμε σημαντικότερη βελτίωση, μειώνοντας ταυτόχρονα και το χρόνο εκπαίδευσης σημαντικά.

Ο 3^{ος} ταξινομητής (Naïve Bayes) δε βλέπει σημαντικές διαφορές με ή χωρίς υποδειγματοληψία.

Leave-One-Out Cross-Validation

Το Leave-One-Out Cross-Validation αποτελεί πιο αντικειμενικό τρόπο αξιολόγησης των ταξινομητών, αλλά και αρκετά πιο χρονοβόρο.

Σύγκριση ταξινομητών

Σε όλα τα πειράματα, το Νευρωνικό Δίκτυο είχε τα καλύτερα αποτελέσματα στην ταξινόμηση των δειγμάτων, ο Random Forest ταξινομητής ήταν ο επόμενος καλύτερος και ο χειρότερος ταξινομητής ήταν αυτός βασισμένος σε Bayesian Networks. Παρόλα αυτά, ο τελευταίος είναι πολύ γρήγορος στην εκπαίδευση και δεν φαίνεται να επηρεάστηκε από το μη ισορροπημένο σύνολο δεδομένων, αφού η υποδειγματοληψία δε βοήθησε την επίδοσή του.

Ερώτημα 3: Συσταδοποίηση συμμετεχόντων

Για τη συσταδοποίηση των συμμετεχόντων βάσει της δραστηριότητάς τους στο 2ωρο που κατέγραφαν οι αισθητήρες, χρησιμοποιήσαμε τους αλγορίθμους KMeans και Agglomerative Hierarchical Clustering.

```
#### Clustering
# Create feature matrix
feature_matrix = fn.create_feature_matrix(y)

# Perform K-means clustering
kmeans = KMeans(n_clusters=3, random_state=17)
km_labels = kmeans.fit_predict(feature_matrix)
dv.plot_clusters(feature_matrix, km_labels)

# Perform hierarchical clustering
agg_clustering = AgglomerativeClustering(n_clusters=3)
agg_labels = agg_clustering.fit_predict(feature_matrix)
dv.plot_dendrogram(feature_matrix, agg_labels)
dv.plot_clusters(feature_matrix, agg_labels)
```

Με τη συνάρτηση `create_feature_matrix` δημιουργούμε τα διανύσματα ιδιοτήτων του κάθε ατόμου και τα προσθέτουμε στο μητρώο ιδιοτήτων. Τα διανύσματα ιδιοτήτων είναι 12 διαστάσεων (1 διάσταση για κάθε δραστηριότητα) και η τιμή κάθε διάστασης αναπαριστά το συνολικό χρόνο που ξόδεψε ο χρήστης κάνοντας την αντίστοιχη δραστηριότητα. Στην πραγματικότητα υπολογίζεται το πόσα δείγματα έχει ο κάθε χρήστης από κάθε δραστηριότητα, οπότε δεν αναπαριστά δευτερόλεπτα, αλλά τον αριθμό των παραθύρων που έχει ανά δραστηριότητα. Αυτό όμως δεν έχει σημασία γιατί όλες οι τιμές **κανονικοποιούνται**.

```

def create_feature_matrix(y: np.array):
    """Creates feature matrix to use for clustering."""

    num_subjects = len(y)
    # Initialize the feature matrix
    feature_matrix = np.zeros((num_subjects, 12))
    # Fill the feature matrix
    for i, subject_data in enumerate(y):
        labels, counts = np.unique(subject_data, return_counts=True)
        feature_matrix[i, labels] = counts

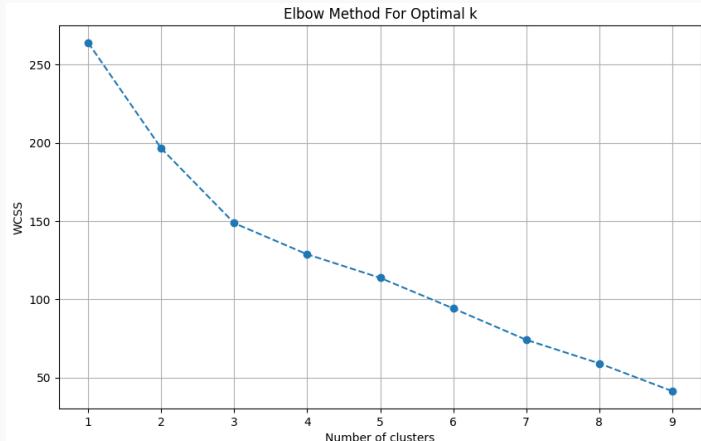
    # Normalize the feature vectors
    scaler = StandardScaler()
    feature_matrix = scaler.fit_transform(feature_matrix)
    return feature_matrix

```

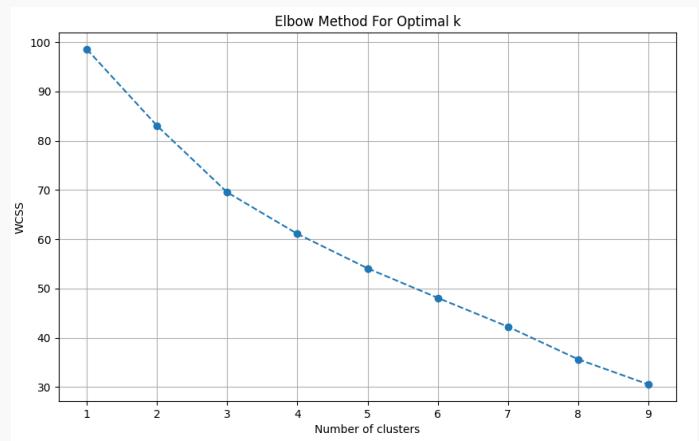
Τέλος, για την απεικόνιση των συστάδων χρησιμοποιούμε Principal Component Analysis (PCA) ώστε να μειώσουμε τις διαστάσεις των διανυσμάτων και να τις απεικονίσουμε σε 2 διαστάσεις. Οι 2 αλγόριθμοι παράγουν την ίδια έξοδο όταν ζητήσουμε 3 ομάδες (για το KMeans εξαρτάται και από την αρχική κατάσταση των κέντρων αλλά επειδή χρησιμοποιούμε συγκεκριμένο random_state είναι πάντα το ίδιο). Όπως φαίνεται και παρακάτω, τα άτομα χωρίζονται στις ίδιες συστάδες και με τους 2 αλγορίθμους.

Για την επιλογή του αριθμού των συστάδων χρησιμοποιήσαμε το **elbow_method**, σύμφωνα με το οποίο επιλέξαμε να έχουμε 3 ομάδες και για τους 2 αλγορίθμους.

KMeans



Hierarchical



Άτομα ανά συστάδα

```

root@bd267b4d646a:/home/data-mining-24# /bin/python3.11 /home/data-mining-24/main.py
Cluster 0 individuals: [1, 3, 6, 7, 8]
Cluster 1 individuals: [0, 2, 4, 5, 11, 12, 13, 14, 15, 16, 18, 19, 20, 21]
Cluster 2 individuals: [9, 10, 17]

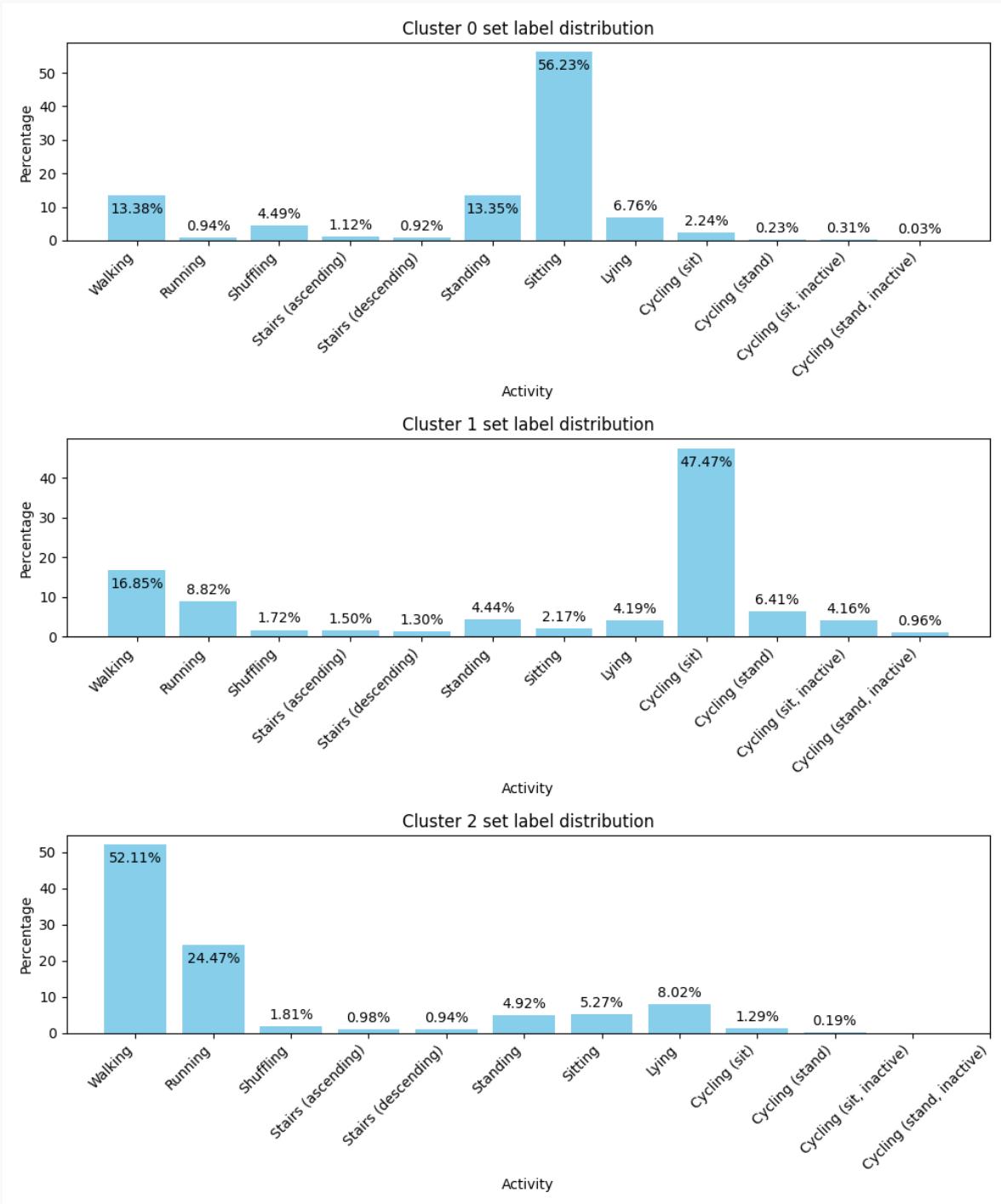
```

```

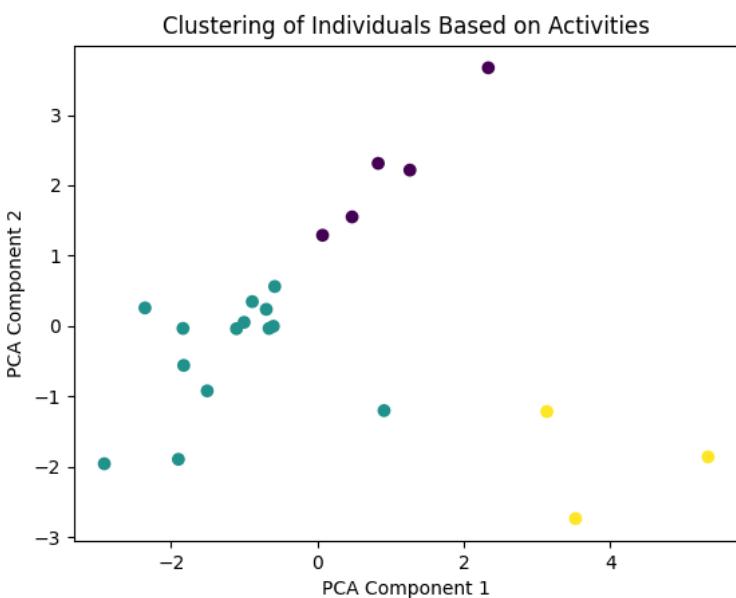
Cluster 0 individuals: [0, 2, 4, 5, 11, 12, 13, 14, 15, 16, 18, 19, 20, 21]
Cluster 1 individuals: [9, 10, 17]
Cluster 2 individuals: [1, 3, 6, 7, 8]

```

Κατανομή ετικετών ανά συστάδα



Οπτικοποίηση χρηστών χρησιμοποιώντας PCA για μείωση διαστάσεων



Συμπέρασμα

Η ομαδοποίηση των χρηστών σε 3 συστάδες βγάζει ικανοποιητικά αποτελέσματα και με τους 2 αλγορίθμους. Από την κατανομή των ετικετών φαίνεται ότι η 1^η συστάδα αποτελείται κυρίως από χρήστες χωρίς ιδιαίτερη δραστηριότητα, αφού περίπου το 70% της ώρας που καταγράφουν οι αισθητήρες είτε κάθονται (56.23%), είτε στέκονται (13.35%). Η 2^η συστάδα αποτελείται από «ποδηλάτες», οι οποίοι ξοδεύουν με διαφορά τον περισσότερο χρόνο κάνοντας ποδήλατο (59% του χρόνου καταγραφής). Τέλος, στην 3^η συστάδα παρατηρείται η τάση για πολύ περπάτημα (52.11%) και τρέξιμο (24.47%).