AI4ST 2024-2025: Signal and Imaging Acquisition and Modelling in Healthcare

Alessandro Dubini 885957
Elia Leonardo Martin 886366
Alice Menna 888364

Neuro Spike

AI-POWERED EEG EPILEPSY CLASSIFICATION SYSTEM

CLINICAL CONTEXT

IMPORTANCE OF CLINICAL CLASSIFICATION AND CONSEQUENCES

Epilepsy classification & Importance of NeuroSpike

Global Prevalence

Approximately 50 million people worldwide have epilepsy.
One of the most common neurological diseases globally.

SOURCE from World Health Organization

Epilepsy classification & Importance of NeuroSpike

Global Prevalence

Approximately 50 million people worldwide have epilepsy. One of the most common neurological diseases globally.

Treatment Potential

Up to 70% of people could live seizure-free with proper diagnosis and treatment.

SOURCE from World Health Organization

Epilepsy classification & Importance of NeuroSpike

Global Prevalence

Approximately 50 million people worldwide have epilepsy.
One of the most common neurological diseases globally.

Treatment Potential

Up to 70% of people could live seizure-free with proper diagnosis and treatment.

Diagnostic Complexity

Subjective interpretation of medical imaging.
Difficulty in distinguishing between epilepsy grades.

SOURCE from World Health Organization

About Our Product

NeuroSpike AI tool aims to classify patients epilepsy grade, helping experts to distinguish between surgical and pharmacological treatment.

About Our Product

NeuroSpike AI tool aims to classify patients epilepsy grade, helping experts to distinguish between surgical and pharmacological treatment.

Improving Patient Life Quality

Surgery is invasive, risky, and requires long recovery.
Avoiding unnecessary operations prevents stress and complications.

About Our Product

NeuroSpike AI tool aims to classify patients epilepsy grade, helping experts to distinguish between surgical and pharmacological treatment.

Improving Patient Life Quality

Surgery is invasive, risky, and requires long recovery.
Avoiding unnecessary operations prevents stress and complications.

Reducing Healthcare Costs

Surgery is expensive;
misclassification leads to financial strain

About Our Product

NeuroSpike AI tool aims to classify patients epilepsy grade, helping experts to distinguish between surgical and pharmacological treatment.

Improving Patient Life Quality

Surgery is invasive, risky, and requires long recovery.
Avoiding unnecessary operations prevents stress and complications.

Reducing Healthcare Costs

Surgery is expensive;
misclassification leads to financial strain

Optimizing Medical Resources

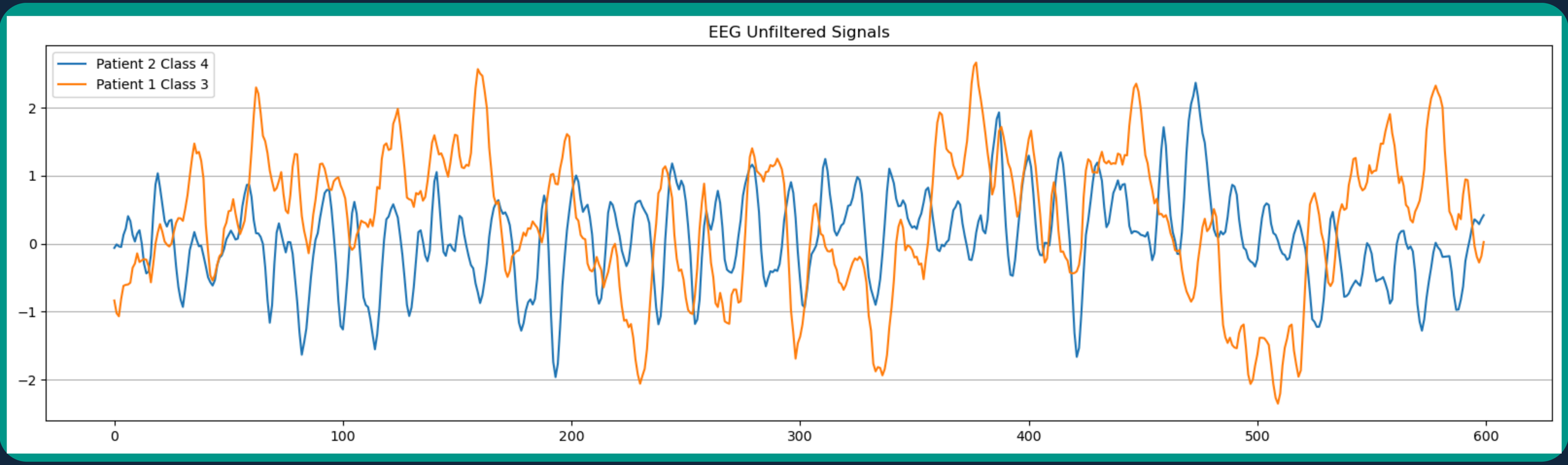
Neurosurgical units have limited capacity; proper classification ensures efficient resource use.

TECHNICAL CONCEPT

SIGNAL ANALYSIS AND PROCESSING

KEY CONCEPT

Patients with severe epilepsy exhibit a much higher zero-crossing rate, which can serve as a key feature in calssification model

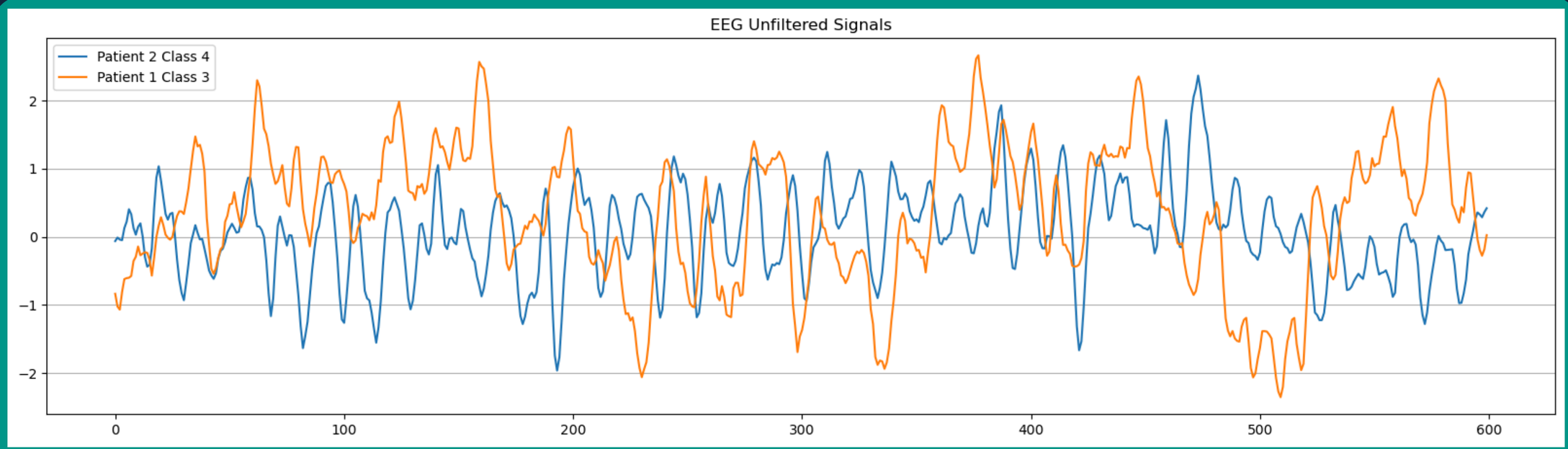


However, an unfiltered signal can lead to model overfitting

FILTERING PROCESS

1

Finit Impulse Response Filter



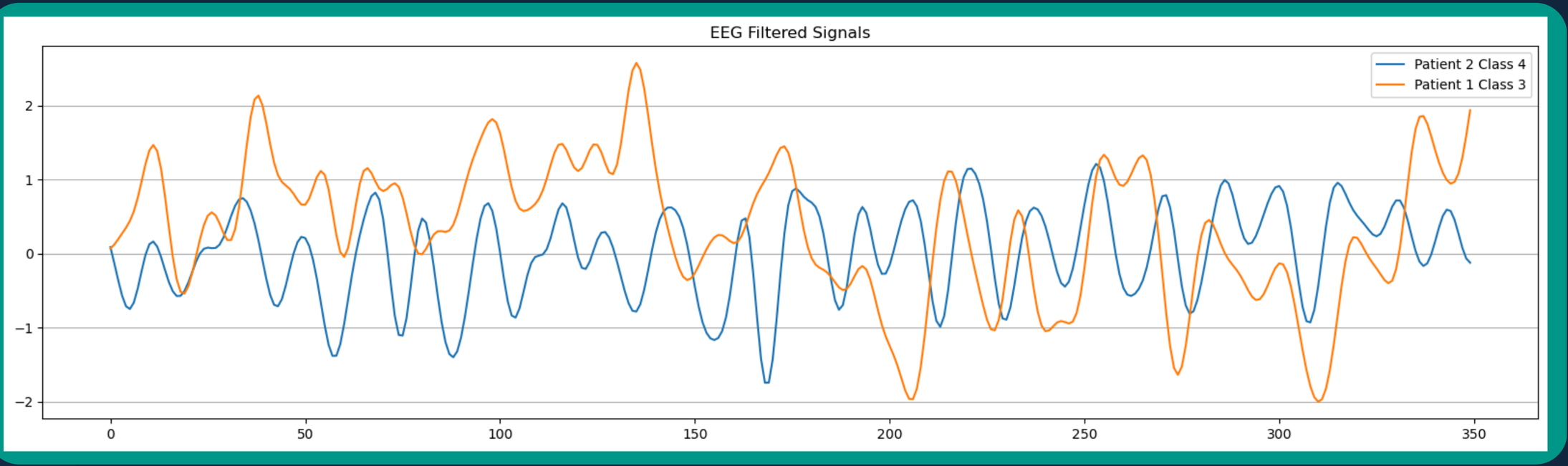
FILTERING PROCESS

1

Finit Impulse Response Filter

number of featuers:

400 —convolution→ 350



FILTERING PROCESS

1

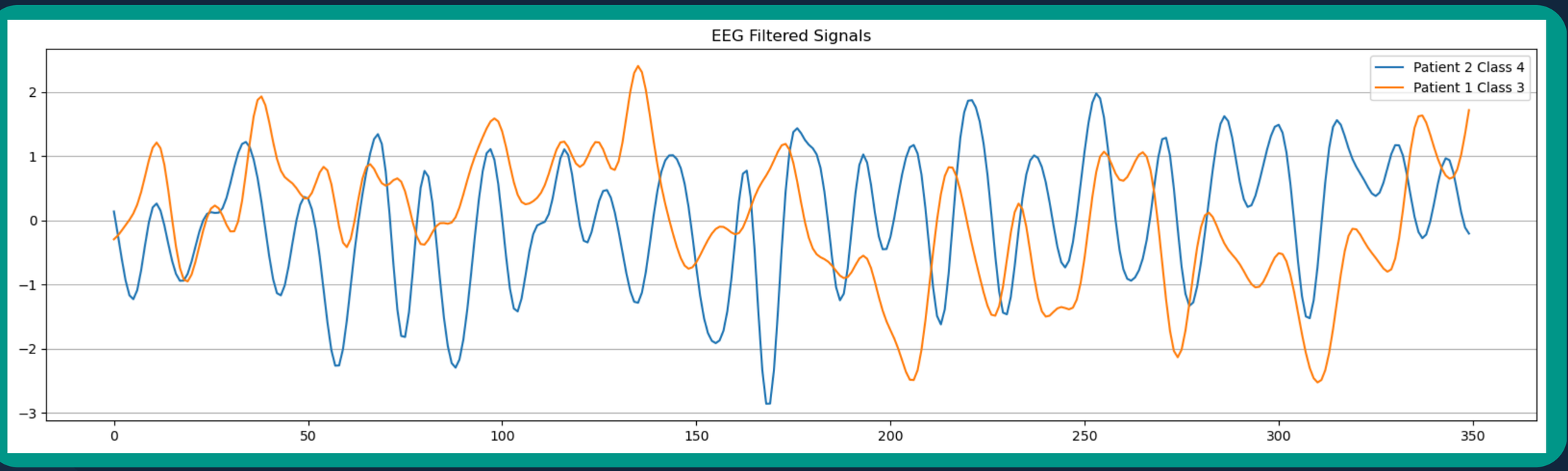
Finit Impulse Response Filter

number of featuers:

400 —convolution→ 350

2

Signal Scaling



FILTERING PROCESS

1

Finit Impulse Response Filter

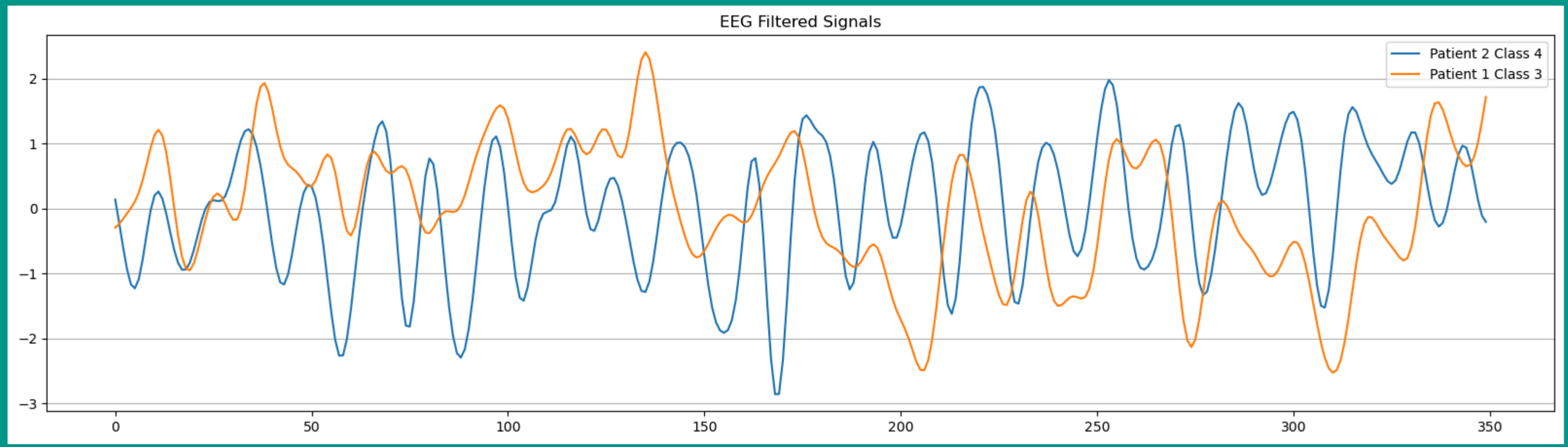
number of featuers:

400 —convolution→ 350

2

Signal Scaling

for enhancement of the z_crossing



FILTERING PROCESS

1

Finit Impulse Response Filter

number of featuers:

400 —convolution→ 350

2

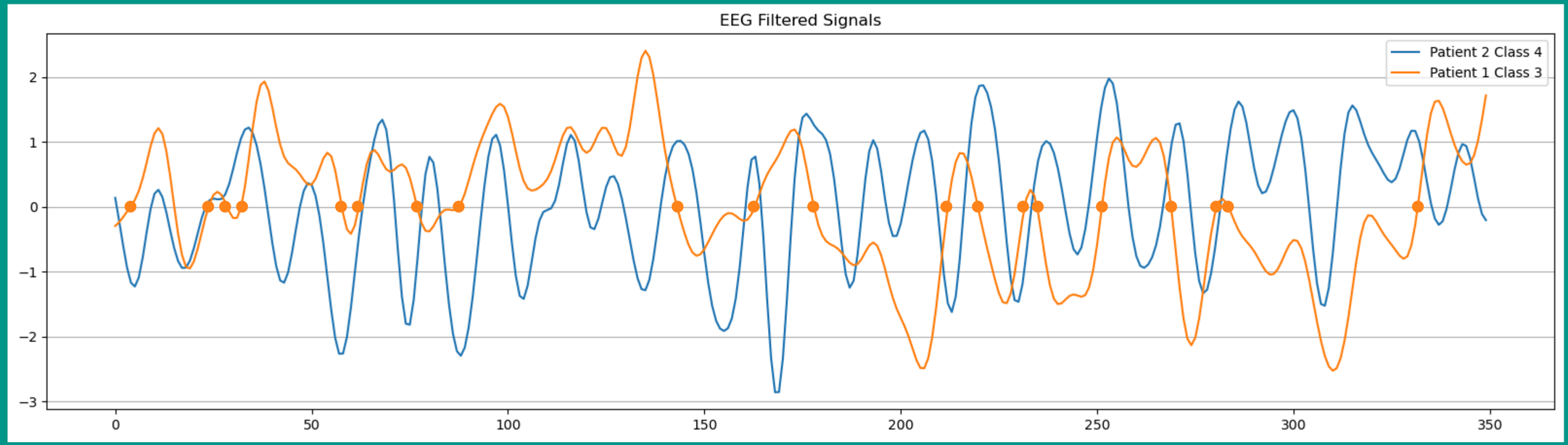
Signal Scaling

for enhancement of the z_crossing

3

Zero-Crossing Extraction

mean of zero_crossing for class 3 = **18.053**



FILTERING PROCESS

1

Finit Impulse Response Filter

number of featuers:

400 —convolution→ 350

2

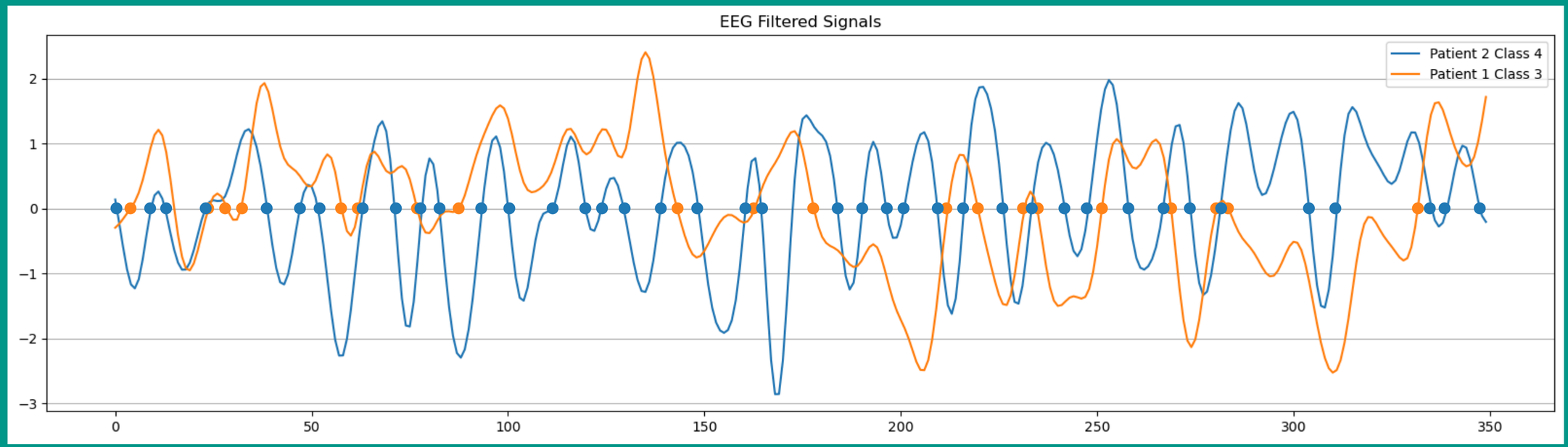
Signal Scaling

for enhancement of the z_crossing

3

Zero-Crossing Extraction

mean of zero_crossing for class 3 = **18.053**
mean of zero_crossing for class 4 = **40.641**



CLASSIFIERS EVALUATION

DECISION TREE, SVM, RANDOM FOREST

Decision Tree

```
DecisionTreeClassifier() : {
    'max_depth': [3, 5, 10],
    'min_samples_split': [2, 5, 10],
    'criterion': ['gini', 'entropy']},
```

Parameters

Decision Tree

Parameters

```
DecisionTreeClassifier() : {
    'max_depth': [3, 5, 10],
    'min_samples_split': [2, 5, 10],
    'criterion': ['gini', 'entropy']},
```

max_depth:

Low values help prevent overfitting

Decision Tree

Parameters

```
DecisionTreeClassifier() : {
    'max_depth': [3, 5, 10],
    'min_samples_split': [2, 5, 10],
    'criterion': ['gini', 'entropy']},
```

max_depth:
Low values help prevent overfitting

min_samples_split:
Low values allow the model to better detect the relationships of a clean dataset

Decision Tree

Parameters

```
DecisionTreeClassifier() : {
    'max_depth': [3, 5, 10],
    'min_samples_split': [2, 5, 10],
    'criterion': ['gini', 'entropy']},
```

max_depth:
Low values help prevent overfitting

min_samples_split:
Low values allow the model to better detect the relationships of a clean dataset

criterion:
Trade off between precision and efficiency

Decision Tree

Parameters

```
DecisionTreeClassifier() : {
    'max_depth': [3, 5, 10],
    'min_samples_split': [2, 5, 10],
    'criterion': ['gini', 'entropy']},
```

→ max_depth:
Low values help prevent overfitting

→ min_samples_split:
Low values allow the model to better detect the relationships of a clean dataset

→ criterion:
Trade off between precision and efficiency

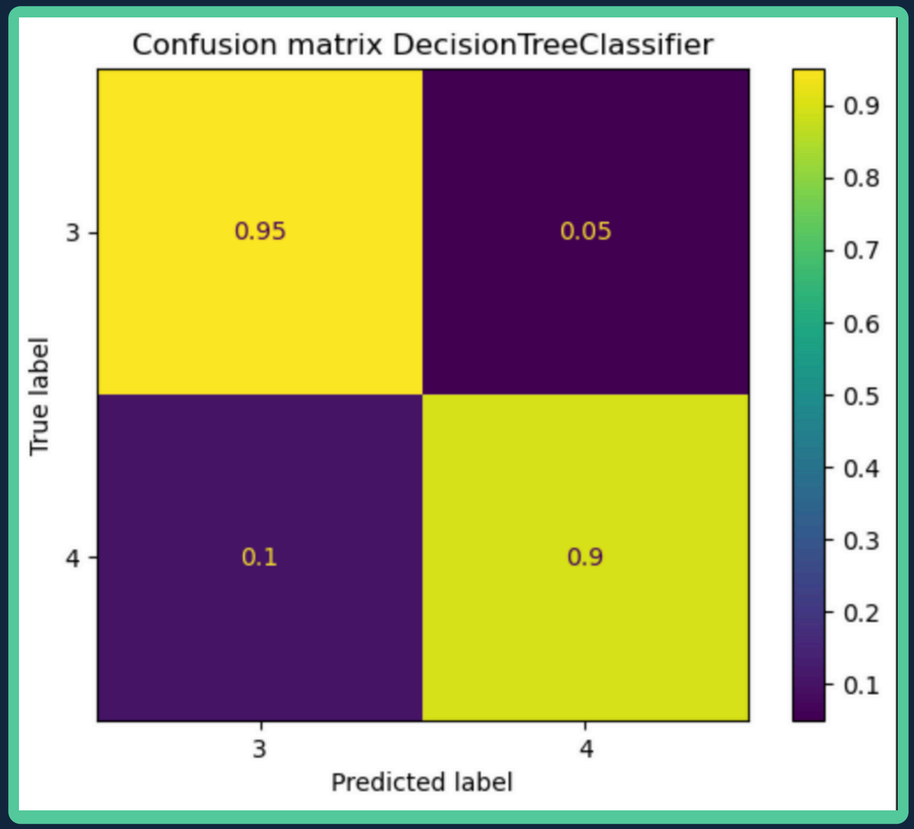
BEST MODEL

DecisionTreeClassifier(max_depth=3, min_samples_split=10, criterion= ‘entropy’)

Decision Tree

Parameters

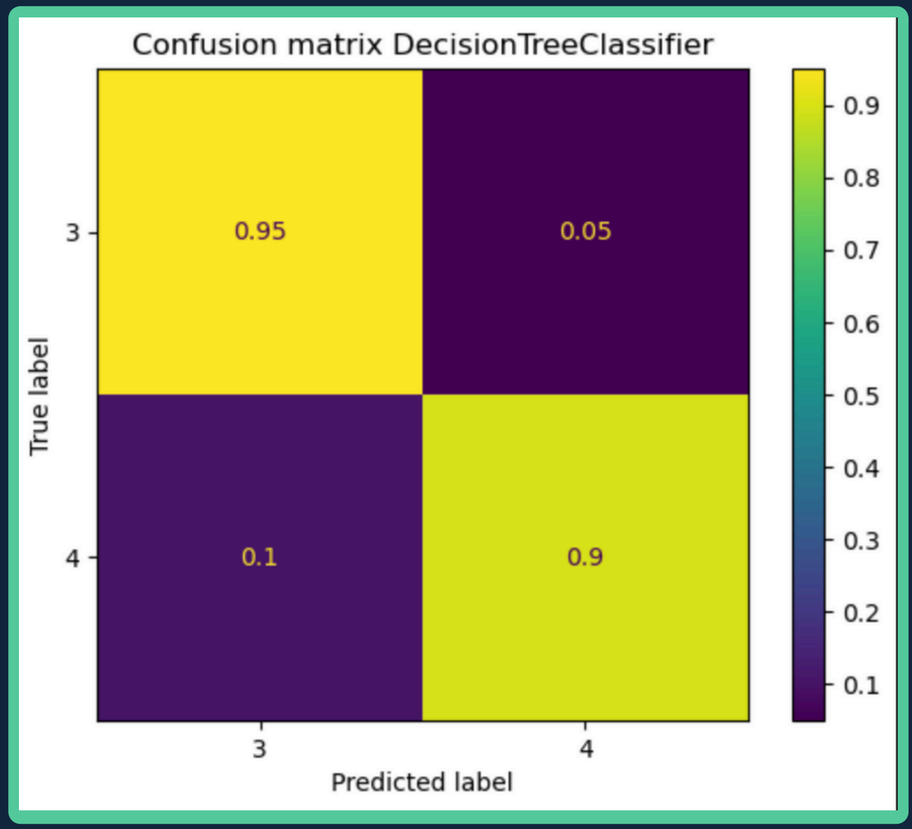
Statistics & Results



Decision Tree

Parameters

Statistics & Results

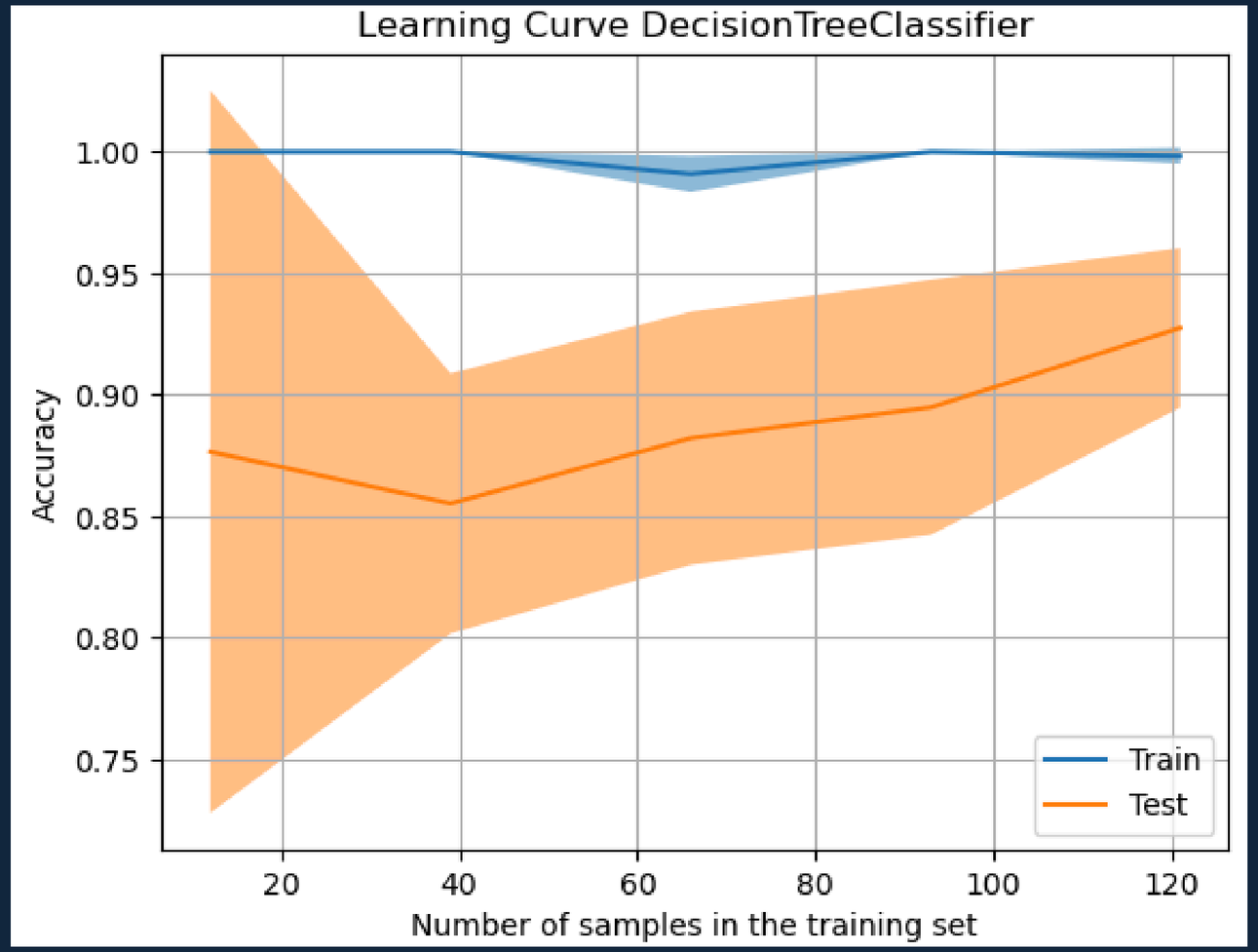


Mean statistics on 20 independent runs:

- Accuracy: 97.87%
- Sensitivity: 96.74%
- Specificity: 98.99%

Decision Tree

- Parameters
- Statistics & Results
- Learning Curve



SVM

Parameters

```
SVC() : {
    'kernel': ['rbf', 'poly', 'sigmoid'],
    'C': [1e-1, 1e-2, 1e-3, 1e-4],
    'gamma': [1e-1, 1e-2, 1e-3, 1e-4],
},
```

kernel:
Evaluation of different kernel shapes to better separate data

SVM

Parameters

```
SVC() : {
    'kernel': ['rbf', 'poly', 'sigmoid'],
    'C': [1e-1, 1e-2, 1e-3, 1e-4],
    'gamma': [1e-1, 1e-2, 1e-3, 1e-4],
},
```

kernel:
Evaluation of different kernel shapes to better separate data

C:
Lower values offer a trade off between accuracy and generalization to avoid overfitting

SVM

Parameters

```
SVC() : {
    'kernel': ['rbf', 'poly', 'sigmoid'],
    'C': [1e-1, 1e-2, 1e-3, 1e-4],
    'gamma': [1e-1, 1e-2, 1e-3, 1e-4],
},
```

kernel:
Evaluation of different kernel shapes to better separate data.

C:
Lower values offer a trade off between accuracy and generalization to avoid overfitting.

gamma:
Lower gamma values ensure simpler boundaries leading to a better generalization

SVM

Parameters

```
SVC() : {
    'kernel': ['rbf', 'poly', 'sigmoid'],
    'C': [1e-1, 1e-2, 1e-3, 1e-4],
    'gamma': [1e-1, 1e-2, 1e-3, 1e-4],
},
```

kernel:
Evaluation of different kernel shapes to better separate data.

C:
Lower values offer a trade off between accuracy and generalization to avoid overfitting.

gamma:
Lower gamma values ensure simpler boundaries leading to a better generalization

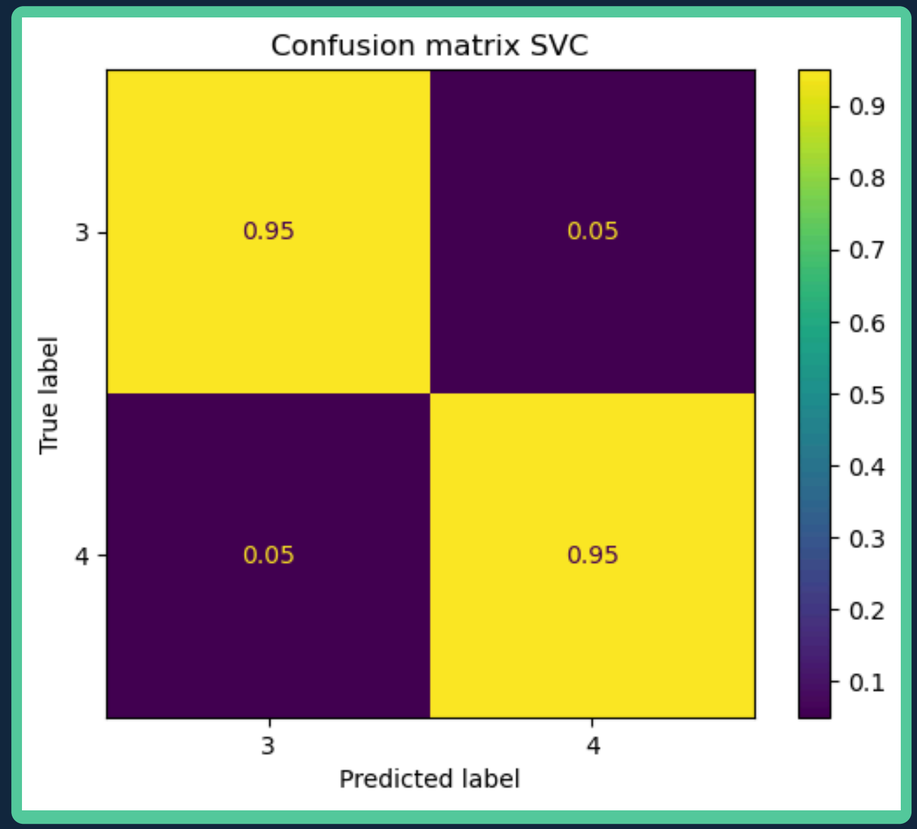
BEST MODEL

SVC(kernel='poly', C=0.1, gamma=0.1)

SVM

Parameters

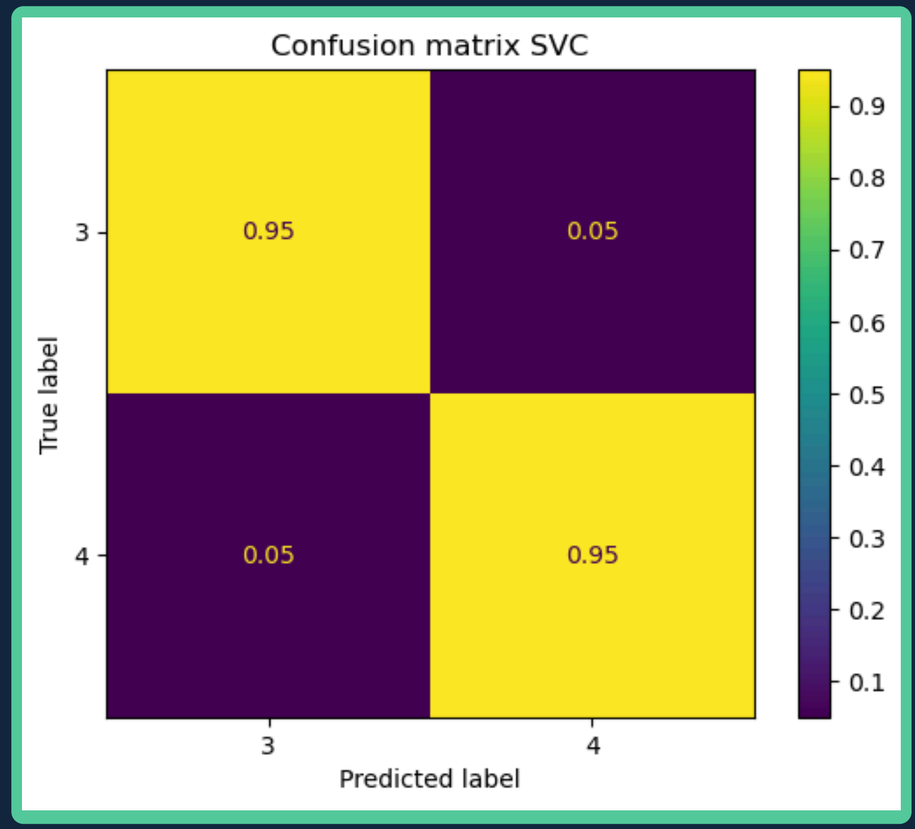
Statistics & Results



SVM

Parameters

Statistics & Results

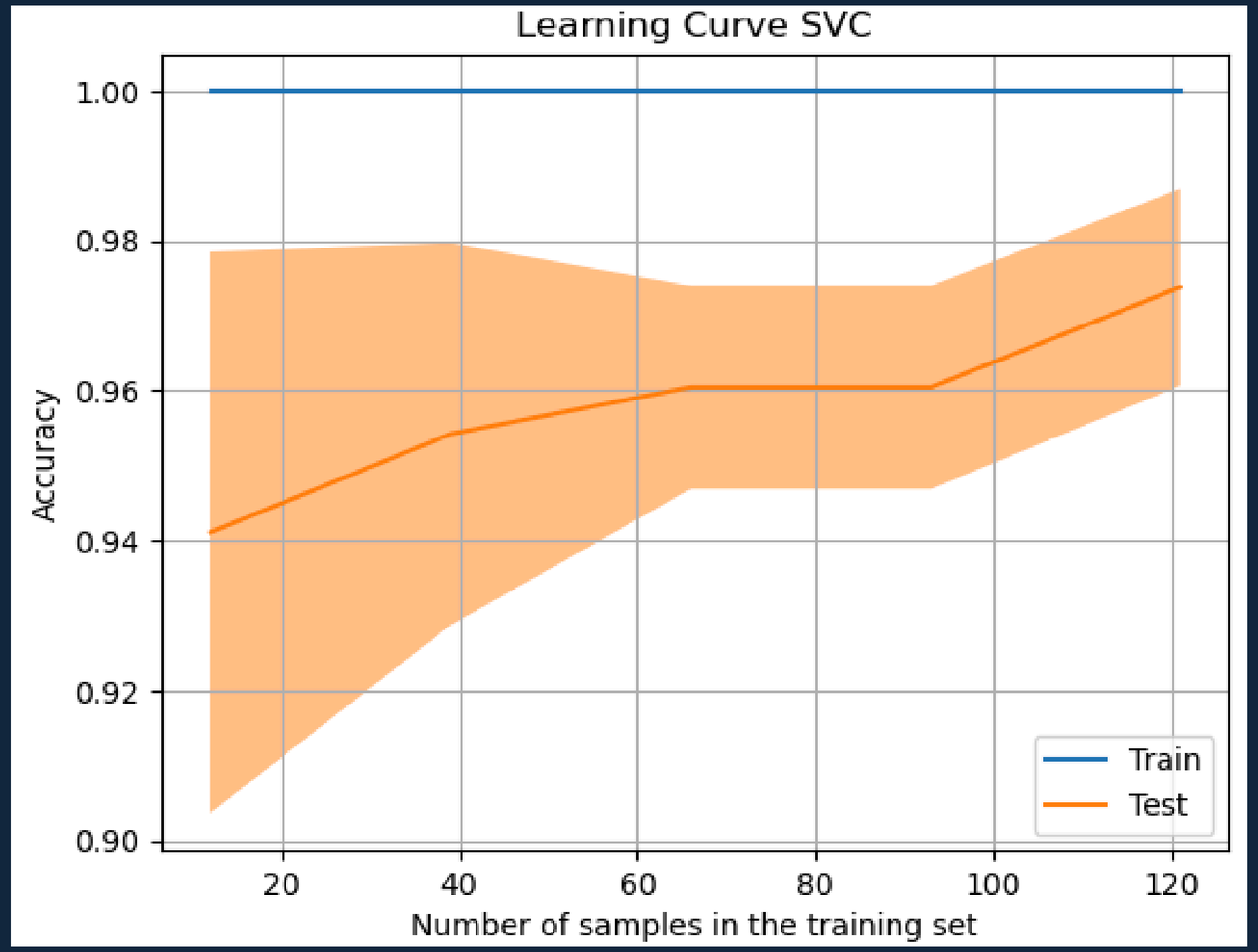


Mean statistics on 20 independent runs:

- Accuracy: 93.37%
- Sensitivity: 92.24%
- Specificity: 94.50%

SVM

- Parameters
- Statistics & Results
- Learning Curve



RANDOM FOREST

Parameters

```
RandomForestClassifier() : {
    'n_estimators': [10, 20, 30, 40, 50],
    'criterion': ['gini', 'entropy'],
    'max_depth': [3, 5, 10],
    'max_features': ['sqrt'],
    'min_samples_split': [2, 5, 10],
    'bootstrap': [True, False]
}}
```

→ max_depth, min_samples_split, criterion

RANDOM FOREST

Parameters

```
RandomForestClassifier() : {
    'n_estimators': [10, 20, 30, 40, 50],
    'criterion': ['gini', 'entropy'],
    'max_depth': [3, 5, 10],
    'max_features': ['sqrt'],
    'min_samples_split': [2, 5, 10],
    'bootstrap': [True, False]
}}
```

max_depth, min_samples_split, criterion:

n_estimators:
Trade off between computational cost and performances.

RANDOM FOREST

Parameters

```
RandomForestClassifier() : {
    'n_estimators': [10, 20, 30, 40, 50],
    'criterion': ['gini', 'entropy'],
    'max_depth': [3, 5, 10],
    'max_features': ['sqrt'],
    'min_samples_split': [2, 5, 10],
    'bootstrap': [True, False]
}}
```

max_depth, min_samples_split, criterion:

n_estimators:
Trade off between computational cost and performances.

max_featuers:
Common for classification tasks, a good balance between diversity and accuracy.

RANDOM FOREST

Parameters

```
RandomForestClassifier() : {
    'n_estimators': [10, 20, 30, 40, 50],
    'criterion': ['gini', 'entropy'],
    'max_depth': [3, 5, 10],
    'max_features': ['sqrt'],
    'min_samples_split': [2, 5, 10],
    'bootstrap': [True, False]
}}
```

max_depth, min_samples_split, criterion:

n_estimators:
Trade off between computational cost and performances.

max_featuers:
Common for classification tasks, a good balance between diversity and accuracy.

bootstrap:
If enabled, allows the model to randomly select a subset of samples with replacement.

RANDOM FOREST

Parameters

BEST MODEL

RandomForestClassifier(
max_depth=10,
min_samples_split=10,
criterion='gini',
n_estimators=50
max_features='sqrt'
bootstrap=False)

```
RandomForestClassifier() : {
    'n_estimators': [10, 20, 30, 40, 50],
    'criterion': ['gini', 'entropy'],
    'max_depth': [3, 5, 10],
    'max_features': ['sqrt'],
    'min_samples_split': [2, 5, 10],
    'bootstrap': [True, False]
}}
```

max_depth, min_samples_split, criterion:

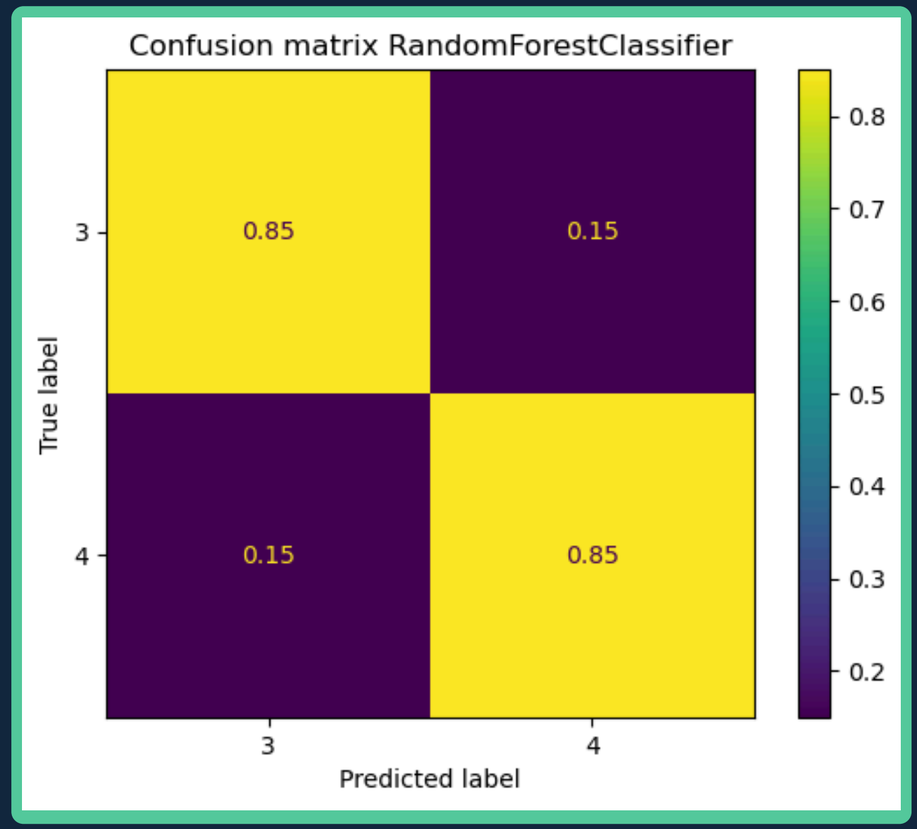
n_estimators:
Trade off between computational cost and performances.

max_featuers:
Common for classification tasks, a good balance between diversity and accuracy.

bootstrap:
If enabled, allows the model to randomly select a subset of samples with replacement.

RANDOM FOREST

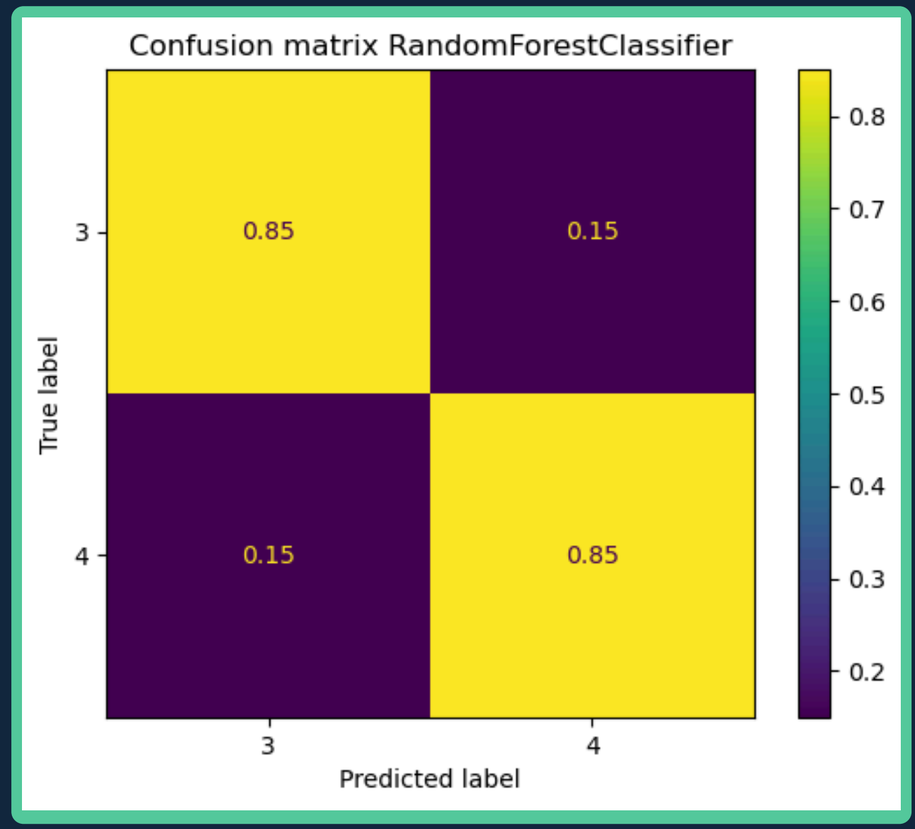
- Parameters
- Statistics & Results



RANDOM FOREST

Parameters

Statistics & Results

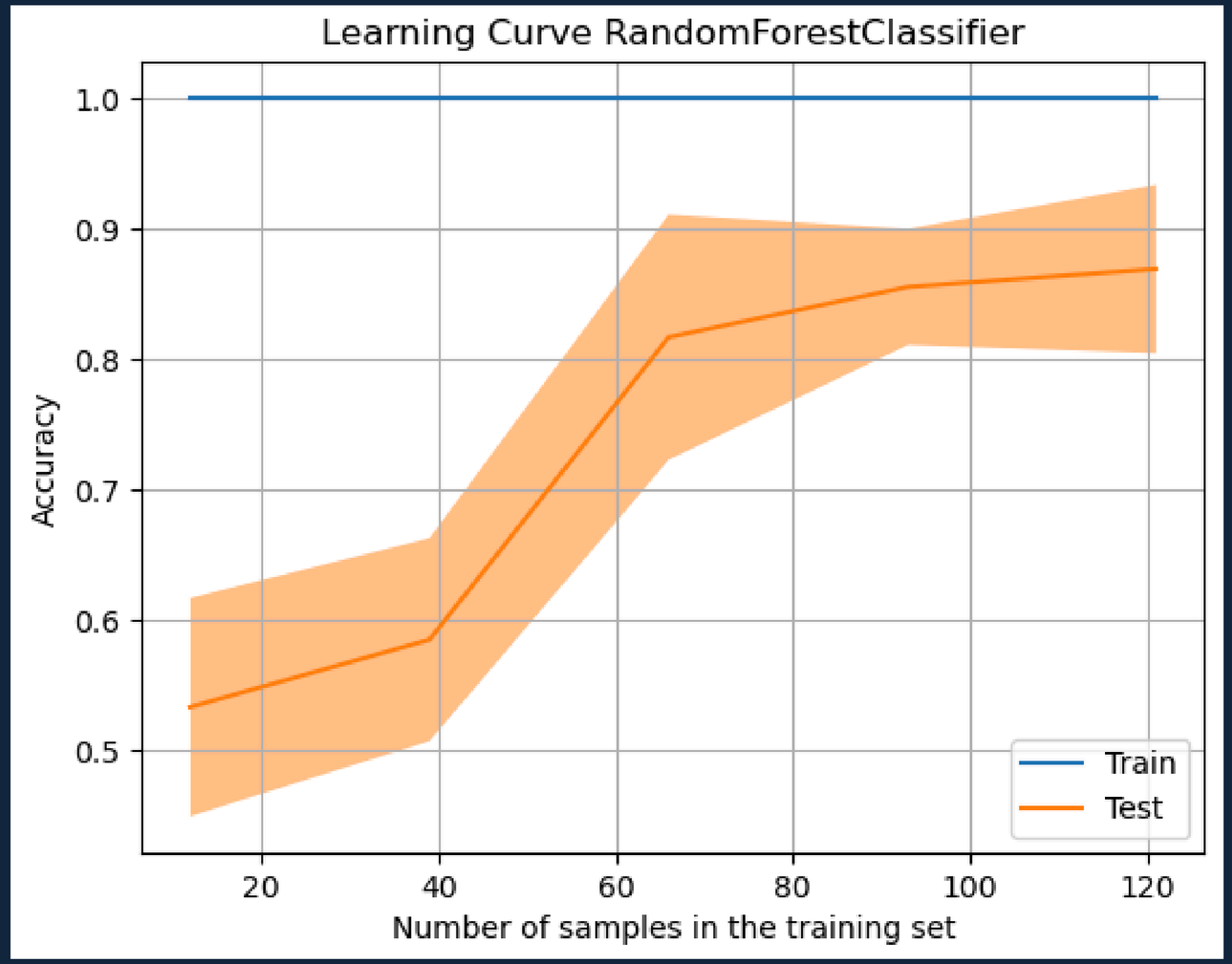


Mean statistics on 20 independent runs:

- Accuracy: 89.37%
- Sensitivity: 93.39%
- Specificity: 84.74%

RANDOM FOREST

- Parameters
- Statistics & Results
- Learning Curve

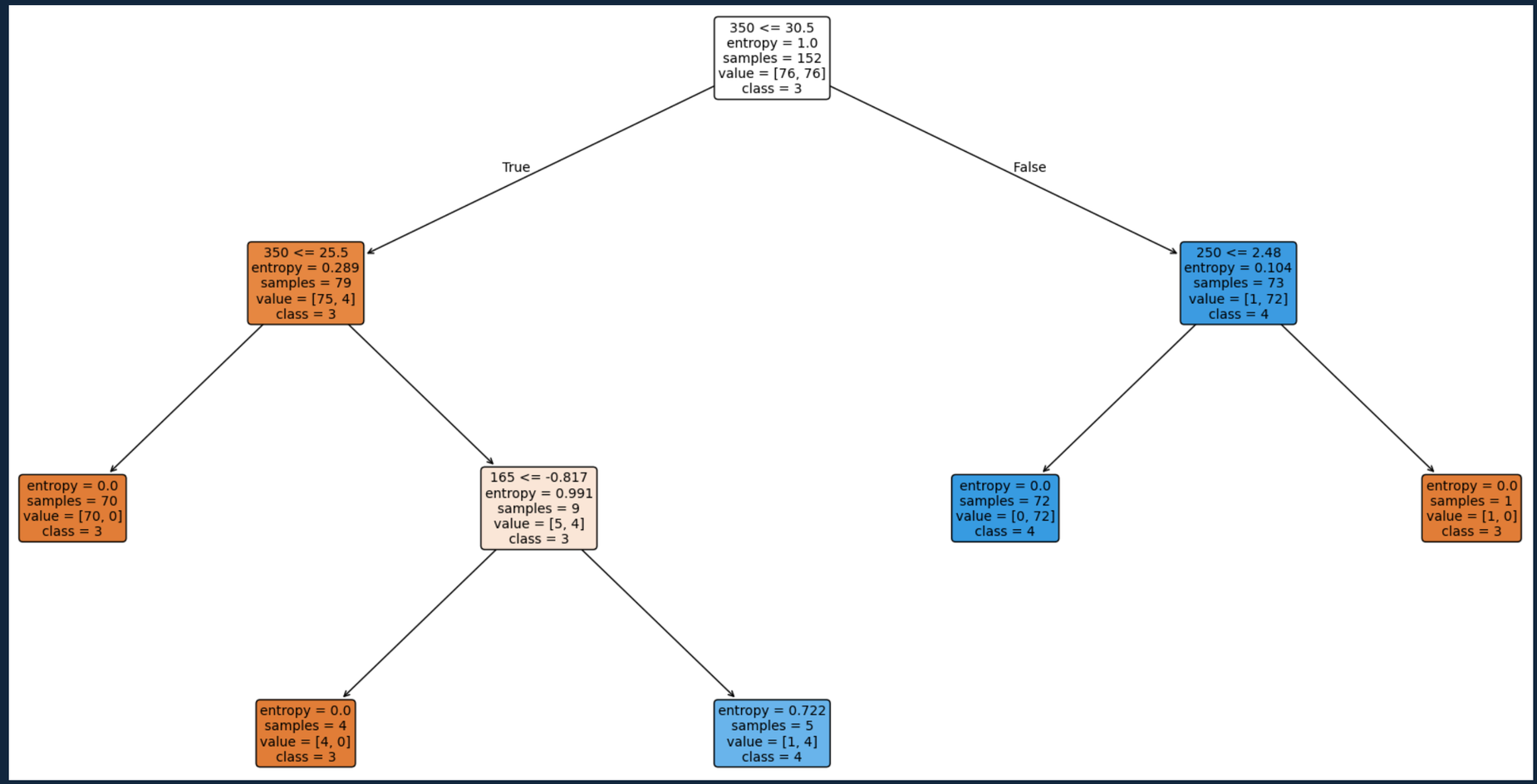


CONCLUSIONS

PERFORMANCES EVALUTIONS

BEST MODEL CLASSIFIER

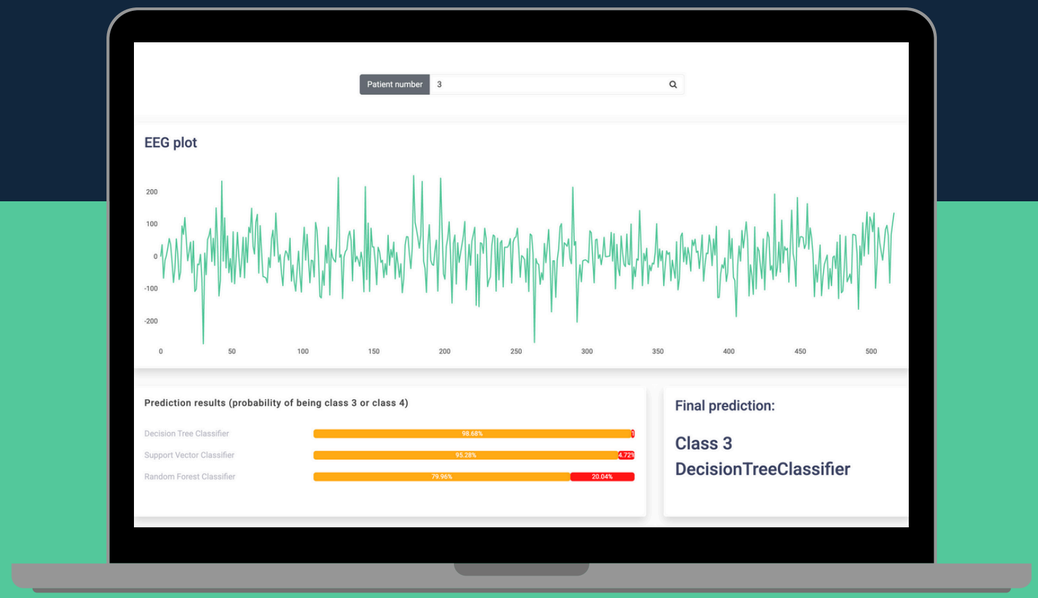
DecisionTreeClassifier(max_depth=3, min_samples_split=10, criterion= ‘entropy’)



FEATURE IMPORTANCES: 350 (zero_crossing)=0.8617 , 250=0.03195513, 165=0.05068369

Try yourself on our Web App

Users upload EEG data, and the app provides real time results, model comparison and basic visualizations to assist in clinical decision making.



Get Application Here
http://127.0.0.1/NeuroSpike

THANK FOR YOUR ATTENTION

BACKUP SLIDES

Variable settings

```
# Outlier detection settings
OUTLIERS_DETECTORS = ["IsolationForest", "ZScore"]
DETECTOR = "IsolationForest"
REMOVE_OUTLIERS = True

# Classifier settings
models = {
    DecisionTreeClassifier() : {
        'max_depth': [3, 5, 10],
        'min_samples_split': [2, 5, 10],
        'criterion': ['gini', 'entropy']},
    SVC() : {
        'kernel': ['rbf', 'poly', 'sigmoid'],
        'C': [1e-1, 1e-2, 1e-3, 1e-4],
        'gamma': [1e-1, 1e-2, 1e-3, 1e-4],
    },
    RandomForestClassifier() : {
        'n_estimators': [10, 20, 30, 40, 50],
        'criterion': ['gini', 'entropy'],
        'max_depth': [3, 5, 10],
        'max_features': ['sqrt'],
        'min_samples_split': [2, 5, 10],
        'bootstrap': [True, False]
    }}
```

[298]

Function to detect outliers using Isolation Forest

```
def outlier_detection_isolation_forest(X: np.ndarray, y:np.ndarray, outlier_class: int = -1) -> np.ndarray:
    isolation_forest = IsolationForest(contamination=0.05)
    res = isolation_forest.fit_predict(X=X)

    # Identify outlier indices
    if outlier_class == -1:
        isolation_forest_outlier_indices = np.where(np.array(res == -1))[0]
    else:
        isolation_forest_outlier_indices = np.where(np.array(res == -1) & (y == outlier_class))[0]

    return isolation_forest_outlier_indices
```

[299]

Function to detect outliers using Z-score method

```
# Function to detect outliers using Z-score method
def outlier_detection_Z_score(X: np.ndarray, y: np.ndarray, outlier_class: int = -1) -> np.ndarray:
    threshold = 3 # Z-score threshold for identifying outliers
    drop_threshold = 100 # Minimum count of outlier dimensions for a sample to be removed

    z_scores = np.abs(stats.zscore(X))
    z_score_count = np.sum(z_scores > threshold, axis=1)
    res = np.where(z_score_count > drop_threshold, [-1]*z_scores.shape[0], 1)

    if outlier_class == -1:
        z_score_outliers_indices = np.where(np.array(res == -1))[0]
    else:
        z_score_outliers_indices = np.where(np.array(res == -1) & (y == outlier_class))[0]

    return z_score_outliers_indices
```

[300]

Function to normalize data using StandardScaler

```
# Function to normalize data using StandardScaler
def data_scaling(df: pd.DataFrame) -> tuple[np.ndarray, np.ndarray]:
    scaler3 = StandardScaler()
    X_3 = scaler3.fit_transform(df[df.y == 3].iloc[:, 1:].to_numpy())
    scaler4 = StandardScaler()
    X_4 = scaler4.fit_transform(df[df.y == 4].iloc[:, 1:].to_numpy())

    X_3[:, -1] = 3
    X_4[:, -1] = 4

    normalized_df = np.concatenate([X_3, X_4], axis=0)
    np.random.shuffle(normalized_df)
    X = normalized_df[:, :-1]
    y = normalized_df[:, -1]

    return X, y

def data_scaling_ext(df: pd.DataFrame) -> tuple[np.ndarray, np.ndarray]:
    scaler = StandardScaler()
    X = scaler.fit_transform(df.iloc[:, 1:-1])
    y = df.iloc[:, -1]

    return X, y
```

[302]

Function to extract time-domain features from EEG data

```
[303]
# Function to extract time-domain features from EEG data
def time_domain_features(eeg_signal):
    features = {
        "mean": np.mean(eeg_signal),
        "variance": np.var(eeg_signal),
        "skewness": skew(eeg_signal),
        "kurtosis": kurtosis(eeg_signal),
        "rms": np.sqrt(np.mean(eeg_signal**2)),
        "zero_crossings": np.sum(np.diff(np.sign(eeg_signal)) != 0),
    }

    features["norm_zero_crossing"] = (features["zero_crossings"] - features["mean"]) / (features["rms"] + 1e-6)

    return features
```

Function for signal processing

```
# Function to preprocess EEG signals with filtering and feature extraction
def signal_processing(row):
    fs = 250  # Sampling frequency (Hz)

    # Low-pass filter (FIR filter design)
    kernel = signal.firwin(51, cutoff=30, fs=fs)  # 30 Hz cutoff
    filtered_signal = np.convolve(row[:400], kernel, mode='valid')
    features = time_domain_features(filtered_signal)
    filtered_signal = (filtered_signal - features["mean"]) / np.sqrt(features["variance"])
    features = time_domain_features(filtered_signal)

    row = np.append(filtered_signal, features["norm_zero_crossing"])

    return row
```

Training and external test

Genera | Codice | Markdown

```python
def train_models(targets, X, y):
    best_models = {}
    for model, params in targets.items():
        best_model, best_params = fit_classifier(model, X, y, param_grid=params)
        best_models[model.__class__.__name__] = best_model

    return best_models

def run_external_test(models, X, y_true):
    results = {}
    for model_name, model in models.items():
        y_pred = model.predict(X)
        accuracy = sklearn.metrics.accuracy_score(y_true, y_pred)
        sensitivity = sklearn.metrics.recall_score(y_true, y_pred, pos_label=4)
        specificity = sklearn.metrics.recall_score(y_true, y_pred, pos_label=3)
        matrix = confusion_matrix(y_true, y_pred, normalize='true')
        results[model_name] = {"accuracy": accuracy, "sensitivity": sensitivity, "specificity": specificity, "confusion_matrix": matrix}

    return results
```

Running test function

```python
scorers = {
    'accuracy': sklearn.metrics.make_scorer(sklearn.metrics.accuracy_score),
    'sensitivity': sklearn.metrics.make_scorer(sklearn.metrics.recall_score, pos_label=4),
    'specificity': sklearn.metrics.make_scorer(sklearn.metrics.recall_score, pos_label=3)
}

def fit_classifier(classifier, x, y, param_grid, cv=5, scoring='accuracy') -> tuple[Any, Any]:
    grid_search = GridSearchCV(estimator=classifier,
                               param_grid=param_grid,
                               cv=cv,
                               scoring=scoring)

    grid_search.fit(x, y)
    return grid_search.best_estimator_, grid_search.best_params_
```

```python
# Training loop
best_models = train_models(models, extended_x, y)
print(best_models)
```