Laboratory of Bioinformatics II Project Manual

Extended Technical Guide for Setup, Data Collection, Analysis and Evaluation

Group 5

A.Y. 2025–2026

Contents

Chapter 1

Project Overview

Aim

Provide an overview of the biological motivation, project objectives and modular workflow structure. This section serves as a guide before diving into technical details.

1.1 Context and Motivation

The correct identification of **signal peptides (SPs)** is a fundamental step in bioinformatics and molecular biology, as these short N-terminal motifs determine whether a protein will enter the secretory pathway. Understanding the presence and location of SPs provides valuable insight into the **functional role of proteins** and their subcellular localization. Experimental validation through wet-lab techniques is precise but time-consuming and resource-intensive. For this reason, computational methods are increasingly adopted, offering scalability and reproducibility.

1.2 Objectives

The project aims to compare two computational strategies for signal peptide prediction:

1. A **motif-based approach**, derived from the cleavage site consensus rules proposed by von Heijne (1986).
2. A **machine learning classifier** based on Support Vector Machines (SVM), exploiting sequence-derived features.

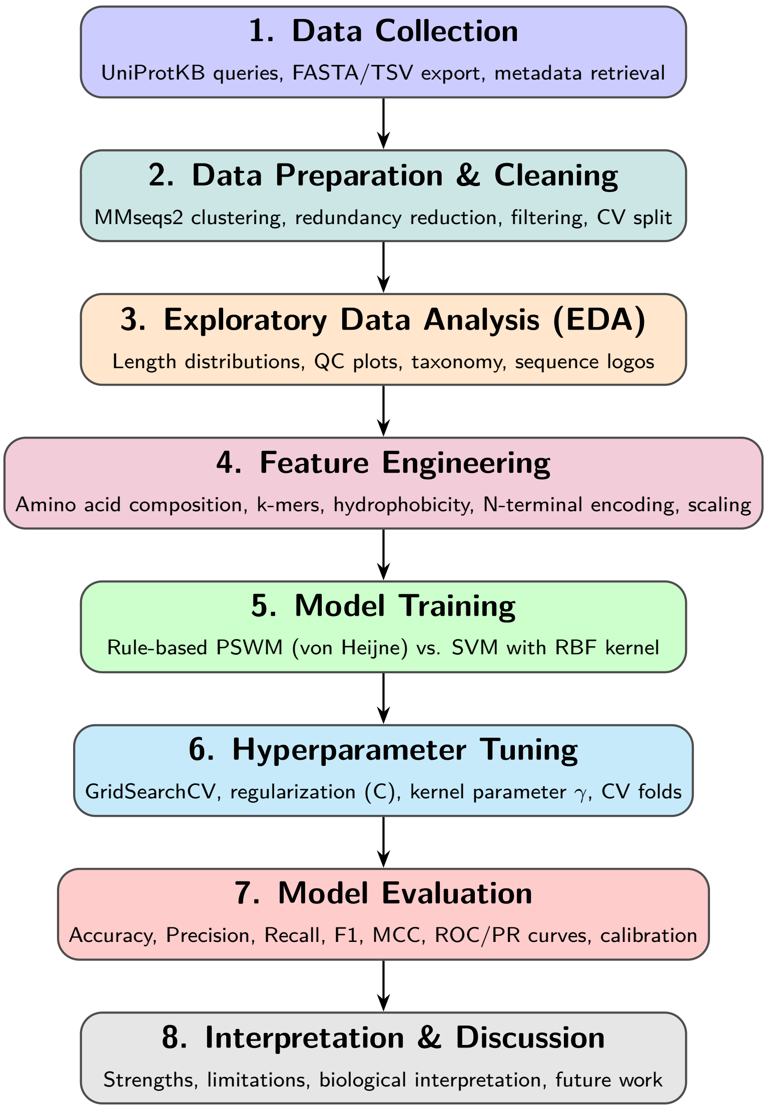
The comparative analysis allows assessment of the trade-offs between rule-based methods, which are interpretable but limited, and data-driven models, which can capture more complex sequence patterns.

1.3 Workflow Structure

The workflow is organized in modular phases, each corresponding to a reproducible step in the repository:

1. **Data Collection**: retrieval of curated datasets from UniProtKB in FASTA and TSV format, with strict filtering criteria.

2. **Data Preparation & Cleaning**: redundancy reduction by clustering with MMseqs2, representative selection, and generation of training and benchmarking sets with cross-validation splits.
3. **Exploratory Data Analysis (EDA)**: exploratory statistics, dataset quality control, sequence motif visualization.
4. **Feature Engineering**: derivation of numerical descriptors from raw sequences (e.g., amino acid composition, k-mer profiles, hydrophobicity indices).
5. **Model Training**: comparison between von Heijne's cleavage rule and SVM classifiers trained on extracted features.
6. **Hyperparameter Tuning**: optimization of model parameters through systematic search (e.g., grid search, randomized search, or Bayesian optimization), including regularization terms (e.g., C, γ for SVMs), number of CV folds, and other algorithm-specific parameters.
7. **Model Evaluation**: quantitative assessment using cross-validation, blind test benchmarking, and diagnostic plots (ROC, PR curves, confusion matrices).
8. **Interpretation & Discussion**: interpretation of results, identification of limitations, and proposal of methodological improvements.


1. Data Collection
UniProtKB queries, FASTA/TSV export, metadata retrieval
2. Data Preparation & Cleaning
MMseqs2 clustering, redundancy reduction, filtering, CV split
3. Exploratory Data Analysis (EDA)
Length distributions, QC plots, taxonomy, sequence logos
4. Feature Engineering
Amino acid composition, k-mers, hydrophobicity, N-terminal encoding, scaling
5. Model Training
Rule-based PSWM (von Heijne) vs. SVM with RBF kernel
6. Hyperparameter Tuning
GridSearchCV, regularization (C), kernel parameter γ, CV folds
7. Model Evaluation
Accuracy, Precision, Recall, F1, MCC, ROC/PR curves, calibration
8. Interpretation & Discussion
Strengths, limitations, biological interpretation, future work

1.4 Expected Outcomes

The project is designed to deliver:

- Curated, transparent datasets in FASTA and TSV format, available in the repository.
- Reproducible scripts and notebooks documenting every stage of the workflow.
- Diagnostic analyses and plots enabling objective performance comparison.
- A formal discussion of strengths and weaknesses of statistical versus machine learning approaches.

Chapter 2

Libraries

2.1 Overview

Aim

Describe and document the Python libraries used in this project, highlighting their roles in data handling, visualization, feature extraction, and machine learning. Each library is introduced with practical examples, a short glossary, and a mini cheat sheet for quick reference.

NumPy

Aim: Efficient numerical computation and linear algebra for feature extraction.
Overview. NumPy provides optimized array structures and mathematical functions essential for handling protein feature matrices.
Main roles in this project:

- Store and manipulate protein feature matrices.
- Vectorized operations for amino acid composition and k-mers.
- Linear algebra (normalization, projections).
- Random number generation for reproducibility.

Example:

```
import numpy as np
features = np.random.rand(100, 20)        # 100 proteins x 20 features
normalized = features / np.linalg.norm(features, axis=1)[:,None]
```

Cheat Sheet:

Command	Purpose
`np.array(list)`	Create array from list
`np.zeros((m,n))`	Matrix of zeros
`np.random.rand(m,n)`	Uniform random numbers
`np.linalg.norm(v)`	Vector norm
`np.mean(arr,axis=0)`	Column-wise mean

Glossary:

- **Array**: Core NumPy data structure, similar to a matrix.
- **Vectorization**: Operations on arrays without explicit loops.
- **Broadcasting**: Automatic expansion of arrays for element-wise ops.

Pandas

Aim: Structured data manipulation and annotation handling.

Overview. Pandas provides DataFrames for tabular data, crucial for integrating FASTA/TSV metadata.

Main roles:

- Load and merge datasets (FASTA/TSV/CSV).
- Organize extracted protein features into DataFrames.
- Clean and filter metadata.
- Perform statistics and exploratory analysis.

Example:

```
import pandas as pd
df = pd.read_csv("positive.tsv", sep="\t")
df["length"].describe()
```

Cheat Sheet:

Command	Purpose
`pd.read_csv("f.tsv",sep="")`	Load TSV file
`df.head()`	Show first rows
`df.merge(df2,on="id")`	Merge datasets
`df.groupby("class").size()`	Count per class
`df.to_csv("out.csv")`	Save DataFrame

Glossary:

- **DataFrame**: 2D labeled data structure (rows $\times$ columns).
- **Index**: Row identifiers for quick access.
- **Merge**: Combine datasets by shared keys.

Matplotlib

Aim: Create detailed and customizable plots.
Overview. Matplotlib is the core plotting library in Python.
Roles:

- Generate protein statistics plots.
- Customize axes, labels, legends.
- Export figures for the report.

Example:

```
import matplotlib.pyplot as plt
plt.hist(df["length"], bins=50)
plt.xlabel("Protein length"); plt.ylabel("Frequency")
plt.show()
```

Cheat Sheet:

Command	Purpose
`plt.plot(x,y)`	Line plot
`plt.hist(x)`	Histogram
`plt.savefig("fig.png")`	Export figure

Glossary:

- **Figure**: Container for plots.
- **Axes**: Coordinate system inside a figure.

Seaborn

Aim: High-level statistical plots for EDA.
Overview. Seaborn extends Matplotlib with integrated style and DataFrame support.
Roles:

- Plot protein length and SP distributions.
- Categorical plots (barplots, violin plots).
- Heatmaps for confusion matrices.

Example:

```
import seaborn as sns
sns.histplot(df, x="length", hue="class", kde=True)
```

Cheat Sheet:

Command	Purpose
`sns.histplot(df["len"])`	Histogram
`sns.violinplot(x,y)`	Violin plot
`sns.heatmap(cm)`	Heatmap

Glossary:

- **Palette**: Color scheme for plots.
- **Facet**: Subplots conditioned on categorical variables.

Scikit-learn

Aim: Provide machine learning algorithms and evaluation tools.
Overview. Scikit-learn is the standard ML library in Python, with SVM and model selection utilities.
Roles:

- Implement SVM classifier for SP prediction.
- Perform cross-validation and hyperparameter tuning.
- Compute evaluation metrics and ROC/PR curves.

Example:

```
from sklearn.svm import SVC
clf = SVC(kernel="rbf", probability=True)
clf.fit(X_train, y_train)
```

Cheat Sheet:

Command	Purpose
`SVC(kernel="rbf")`	Define SVM
`GridSearchCV(...)`	Hyperparameter tuning
`cross_val_score(...)`	Cross-validation
`confusion_matrix(y,p)`	Evaluation

Glossary:

- **Classifier**: Algorithm that assigns class labels.
- **Cross-validation**: Resampling to estimate performance.

Biopython

Aim: Handle biological sequences and annotations.
Overview. Biopython offers parsers and utilities tailored for bioinformatics.
Roles:

- Parse FASTA files into sequences.
- Extract metadata and annotations.
- Support feature extraction.

Example:

```
from Bio import SeqIO
for record in SeqIO.parse("positive.fasta","fasta"):
    print(record.id, len(record.seq))
```

Cheat Sheet:

Command	Purpose
`SeqIO.parse(f,"fasta")`	Parse FASTA
`record.id`	Sequence ID
`record.seq`	Sequence string

Glossary:

- **SeqRecord**: Object storing a biological sequence + metadata.

OS & Pathlib

Aim: Manage files and directories in a cross-platform way.
Overview. OS and **Pathlib** are standard Python libraries for file system operations.
Roles:

- Navigate project directories.
- Automate dataset saving and loading.
- Ensure compatibility across Linux/Windows.

Example:

```
import os
from pathlib import Path
Path("Data_Collection").mkdir(exist_ok=True)
print(os.listdir("Data_Collection"))
```

Cheat Sheet:

Command	Purpose
`os.listdir(path)`	List files
`Path(p).mkdir()`	Create folder
`os.path.join(a,b)`	Join paths

Glossary:

- **Path object**: Abstract representation of file path.

PyTorch

Aim: Deep learning framework for tensor computation and neural network modeling.
Overview. PyTorch provides dynamic computation graphs, GPU acceleration, and an intuitive Pythonic API for building and training deep learning models.
Roles:

- Define and train neural networks (`torch.nn`, `torch.optim`).
- Perform tensor operations with automatic differentiation (`autograd`).
- Manage datasets and loaders for model training (`torch.utils.data`).

Example:

```
import torch, torch.nn as nn

x = torch.randn(5, 10)
model = nn.Linear(10, 1)
y = model(x)
print(y.shape)
```

Cheat Sheet:

Command	Purpose
`torch.tensor(data)`	Create tensor
`tensor.cuda()`	Move tensor to GPU
`tensor.backward()`	Compute gradients
`torch.save(model, ’m.pt’)`	Save model
`torch.load(’m.pt’)`	Load model

Glossary:

- **Tensor:** Multidimensional array with GPU support.
- **Autograd:** Automatic differentiation engine.
- **Optimizer:** Updates model weights during training.

Chapter 3

Glossary of Technical Terms

3.1 Overview

Aim

Provide clear definitions of the technical terminology used throughout the project, covering biological, computational, and system administration concepts. The glossary ensures consistency in interpretation and serves as a reference during both implementation and evaluation.

General Terminology

Signal Peptide (SP). Short amino-terminal sequence that directs proteins to the secretory pathway.
Context: SPs are the positive class to be predicted.
Note: Typically 15–30 aa, composed of N-, H-, and C-regions.

Cleavage site. Position in the protein sequence where the signal peptide is cut off during secretion.
Role: Defines the SP length and motif context used for classification.

Label. The ground truth assigned to each example in supervised learning.
In project: `Positive` (with SP) or `Negative` (no SP).

Performance. General measure of how well a model makes predictions compared to true labels.
Usage: Evaluated through metrics such as accuracy, MCC, or AUC.

Dataset Terminology

Training set. Subset of data used to fit model parameters.
Rule: The model learns from this portion only.

Validation set. Subset of data used to tune hyperparameters and prevent overfitting.
Note: Distinct from both training and final test set.

Testing set (Benchmark set). Held-out data used only once at the end for unbiased evaluation.
In project: 20% of representative sequences after clustering.

Cross-validation (CV). Statistical technique splitting the training set into folds.
In project: 5-fold CV ensures stable performance estimation.

File Formats

FASTA. Standard text format for biological sequences.
Header: begins with `>` and contains identifier + metadata.
Body: one or more lines with amino acid sequence.

TSV. Tab-Separated Values table.
Usage: Store protein annotations, accessions, and cleavage site info.

YAML. Human-readable markup format.
Usage: Environment specification (`environment.yml` for conda).

Machine Learning Concepts

Support Vector Machine (SVM). Supervised classifier that identifies an optimal hyperplane separating classes.
In project: Implemented with scikit-learn using RBF kernel.

Kernel. Mathematical function that implicitly maps input data into a higher-dimensional feature space.
Purpose: Allows separation of non-linearly separable data.
Common types: Linear, polynomial, and RBF (Radial Basis Function).

RBF (Radial Basis Function) kernel. Measures similarity between samples based on their Euclidean distance.
Role: Introduces non-linearity, enabling complex decision boundaries.

Random Forest (RF). Ensemble learning method based on multiple decision trees trained on random subsets of data and features.
Usage in project: Applied for feature importance ranking and selection.
Strength: Reduces overfitting and provides interpretable variable importance scores.

Encoding. Process of converting categorical or symbolic data (e.g., amino acid letters) into numerical representations.
Examples: One-hot encoding, physicochemical property encoding (AAindex), or embedding vectors.
Purpose: Enables machine learning models to process biological sequences.

Feature extraction. Transformation of raw sequences into numerical descriptors. Examples: amino acid composition, k-mers, hydrophobicity.

Hyperparameter. Parameter set before training (e.g., C, γ in SVM).
Tuning: Done via grid search with cross-validation.

ROC curve. Receiver Operating Characteristic plot.
Axes: Sensitivity vs 1–Specificity.
Usage: Evaluate discriminative ability across thresholds.

PR curve (Precision–Recall curve). Diagnostic plot showing trade-off between precision and recall.
Context: Particularly useful for imbalanced datasets like SP vs non-SP.

MCC (Matthews Correlation Coefficient) Correlation between predicted and true labels.
Range: -1 (total disagreement) to 1 (perfect prediction).
Strength: Robust against imbalance.

Overfitting Phenomenon where a model performs well on training data but poorly on new data.
Prevention: Use CV, regularization, early stopping.

System Administration

Alias. Shortcut for shell commands, typically in `.bashrc`. Example: `alias lb2vm='ssh -i key umXX@mXX.lsb.biocomp.unibo.it'`

SSH (Secure Shell). Protocol for remote login to the VM with key-based authentication.

Screen. Terminal multiplexer.
Usage: Run long jobs persistently, survive VPN/SSH disconnections.

scp (Secure Copy). File transfer utility over SSH.
Example: `scp local.txt user@vm: /remote/dir/`

Chapter 4

Formulas and Key Equations

4.1 Overview

Aim

This chapter collects the mathematical definitions used in the project: evaluation metrics, the von Heijne Position-Specific Weight Matrix (PSWM), and the Support Vector Machine (SVM) classifier. Each equation is explained with its meaning, context, and usage guidelines.

4.2 Evaluation Metrics

Accuracy

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Aim: Overall fraction of correct predictions. **Note:** Misleading on imbalanced datasets (many negatives).

Precision

$$Precision = \frac{TP}{TP + FP}$$

Aim: Reliability of positive predictions. **Context:** High precision means few false positives.

Recall (Sensitivity)

$$Recall = \frac{TP}{TP + FN}$$

Aim: Ability to recover all true positives. **Context:** High recall avoids missing proteins with SP.

F1 Score

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

Aim: Balance between precision and recall. **Tip:** Best metric when both errors matter.

Matthews Correlation Coefficient (MCC)

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

Aim: Robust single-number metric, even with imbalance. **Context:** Main benchmark metric in this project.

4.3 von Heijne Method (PSWM)

PSWM Scoring Function

$$S(seq) = \max_p \sum_{k=-13}^{+2} \log \frac{P(a_{p+k} \mid k)}{q(a_{p+k})}$$

Where:

- a_{p+k} = residue at position k around cleavage site.
- $P(a|k)$ = probability of residue a at position k (positive SPs).
- $q(a)$ = background frequency (SwissProt).

Aim: Rule-based method to identify canonical cleavage motifs. **Tip:** Captures A–X–A pattern but misses non-canonical signals.

4.4 Support Vector Machine (SVM)

SVM Optimization Problem

$$\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i$$

subject to

$$y_i(w \cdot \phi(x_i) + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

Aim: Maximize separation margin between classes. **Note:** Controlled by C (regularization).

RBF Kernel

$$K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)$$

Aim: Map features into higher dimensions for nonlinear separation. **Tip:** γ controls kernel sensitivity (large γ = narrower influence).

4.5 Neural Network (NN)

Artificial Neuron

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

Aim: Compute weighted sum of inputs and apply activation function. **Note:** Basic computational unit of MLPs.

ReLU Activation

$$f(x) = \max(0, x)$$

Aim: Introduce non-linearity and avoid vanishing gradients. **Tip:** Promotes sparsity in hidden layers.

Gradient Descent Update

$$w_{t+1} = w_t - \eta \frac{\partial L}{\partial w_t}$$

Aim: Iteratively minimize loss function L. **Parameter:** η = learning rate controls step size.

Stochastic Gradient Descent (SGD)

$$w_{t+1} = w_t - \eta \frac{1}{m} \sum_{i=1}^{m} \nabla_w L(x_i, y_i)$$

Aim: Update weights after each mini-batch of size m. **Note:** Reduces training time with good convergence properties.

Dropout Regularization

$$h_i' = r_i \cdot h_i, \quad r_i \sim \text{Bernoulli}(p)$$

Aim: Prevent overfitting by randomly dropping neurons. **Tip:** Effective approximation of model averaging.

Batch Normalization

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}, \qquad y_i = \gamma \hat{x}_i + \beta$$

Aim: Normalize activations to stabilize training. **Note:** Allows higher learning rates and faster convergence.

Residual Connection

$$y = F(x, W) + x$$

Aim: Add skip connections to ease gradient flow in deep networks. **Tip:** Enables effective training of very deep MLPs.

Mathematical Glossary

- **TP (True Positive)**: SP correctly predicted.
- **TN (True Negative)**: Non-SP correctly predicted.
- **FP (False Positive)**: Non-SP predicted as SP.
- **FN (False Negative)**: SP missed by the model.
- **Hyperplane**: Decision boundary defined by SVM.
- **Margin**: Distance between hyperplane and nearest data points.
- **Kernel**: Function that maps inputs to higher-dimensional space.
- **Regularization (C)**: Balances margin size vs misclassifications.
- **Gamma (γ)**: Controls influence radius in RBF kernel.
- **ReLU (Rectified Linear Unit)**: Activation function $f(x) = \max(0, x)$ preventing vanishing gradients.
- **SGD (Stochastic Gradient Descent)**: Optimization algorithm updating model weights using mini-batches.
- **Adam (Adaptive Moment Estimation)**: Optimizer combining RMSProp and momentum for adaptive learning rates.
- **RMSProp (Root Mean Square Propagation)**: Adaptive optimizer adjusting learning rate per parameter.
- **Dropout**: Regularization technique randomly disabling neurons during training to prevent overfitting.
- **Batch Normalization (BN)**: Normalization layer stabilizing internal activations

and speeding up convergence.

- **Residual Connection (RC)**: Shortcut link adding a layer's input to its output to improve gradient flow.
- **Epoch**: One complete pass through the entire training dataset during learning.
- **Mini-batch**: Small subset of training data used in each iteration of SGD.
- **Loss Function (L)**: Quantifies prediction error used for gradient-based optimization.
- **Learning Rate (η)**: Controls the step size in weight updates during optimization.
- **Weights (w) and Bias (b)**: Trainable parameters of each neuron.
- **Activation Function ($f(x)$)**: Non-linear transformation applied to neuron outputs.

Chapter 5

GitHub Setup

5.1 Aim

Aim

Ensure reproducible, collaborative and traceable project development through version control and GitHub workflow. This chapter explains repository organization, branching, collaborative practices, and common commands.

5.2 Rationale

Efficient management of the project requires the use of a **version control system**. GitHub ensures:

- **Traceability**: every change is logged with author, timestamp, and commit.
- **Collaboration**: multiple contributors work in parallel through branches.
- **Reproducibility**: scripts, datasets, and environments remain accessible.

5.3 Initial Setup

Before starting collaborative work, configure your local Git environment:

```
git config --global user.name "Your Name"
git config --global user.email "your@email.com"
git config --list
```

Listing 5.1: Basic Git configuration

SSH Key Configuration

```
ssh-keygen -t rsa -b 4096 -C "your@email.com"
eval "$(ssh-agent -s)"
ssh-add ~/.ssh/id_rsa
```

Listing 5.2: Generate and add SSH key

Copy the public key (/.ssh/id_rsa.pub) into GitHub settings → SSH keys.

.gitignore

To prevent large or irrelevant files from being tracked:

```
# raw FASTA exports
*.fasta
# temporary files
*.tmp
# environment files
*.env
__pycache__/
```

Listing 5.3: Example .gitignore

5.4 Repository Structure

```
LB2_project_Group_5/
 Data_Collection/          # Retrieval of raw datasets from UniProtKB

 Data_Preparation/         # Redundancy reduction (MMseqs2), set generation,
    CV folds

 Data_Analysis/            # Exploratory data analysis (AA composition, motifs
    )

 von_Heijne/               # Implementation of von Heijne (1986) method

 Features_extraction/      # Feature extraction (AA composition +
    physicochemical scales)

 SVM_method/               # SVM classifier with feature selection and
    optimization

 Evaluation_and_Comparison/ # Comparative analysis of SVM vs von Heijne models

 Deep_Learning/ # Implementation of MLP

 Supplementary_materials/  # Figures, intermediate files, and additional
    resources

 Report/                   # Final project report (PDF)

 README.md                 # General project overview and workflow description
 LICENSE                   # Open-source license (GPL-3.0)
```

Each subdirectory contains a dedicated `README.md`.

5.5 Branching Model

- **main**: stable, validated code only.

- **dev**: integration of new features before merging to `main`.
- **feature/***: short-lived branches (e.g., `feature/data-prep`).

Example:

```
git checkout -b feature/data-prep
git add get_sets.py
git commit -m "Implemented CV split function"
git push origin feature/data-prep
```

5.6 Pull Requests and Code Review

Pull Requests (PR) ensure collaborative review before merging.

1. Create PR from feature branch → `dev`.
2. Summarize changes and motivation.
3. At least one approval required.

Tip: Never merge to `main` without review.

5.7 Issues and Project Tracking

- **Bug reports**: unexpected script behavior.
- **Enhancements**: propose new analyses/features.
- **Documentation**: missing explanations.

Labels: `bug`, `enhancement`, `documentation`. **Milestones:** group issues into deliverables.

5.8 Best Practices

- Write atomic commits: one logical change per commit.
- Use descriptive commit messages: `"Add clustering script"` not `"fix"`.
- Follow naming conventions: branches like `feature/*`, `bugfix/*`.
- Always pull latest `dev` before starting new work.
- Never commit large raw data → use Zenodo or institutional storage.

5.9 Troubleshooting

- **Merge conflicts**: resolve manually, then `git add` and `git commit`.
- **Undo last commit (soft reset)**:

  ```
  git reset --soft HEAD~1
  ```

- **Discard local changes**:

  ```
  git checkout -- filename
  ```

- **Stash temporary changes**:

  ```
  git stash
  git stash apply
  ```

5.10 Continuous Integration (CI)

GitHub Actions can be configured to:

- Run unit tests at each commit.
- Verify TSV/FASTA format consistency.
- Auto-build LaTeX documentation.

Example workflow YAML:

```
name: Python Tests
on: [push, pull_request]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.10'
      - run: pip install -r requirements.txt
      - run: pytest
```

Local Glossary

- **Commit**: Snapshot of repository at one point in time.
- **Branch**: Parallel line of development.
- **Merge**: Integrate changes between branches.
- **Pull Request (PR)**: Proposal to merge reviewed changes.
- **Clone**: Local copy of a remote repository.
- **Fork**: Copy of a repository under another account.
- **Conflict**: Situation where changes overlap.
- **Tag/Release**: Snapshot marking a version.
- **Stash**: Temporary storage of uncommitted changes.
- **Rebase**: Reapply commits on top of a new base branch.
- **Origin**: Default name for the remote repository.
- **Upstream**: Reference to the original repository when working on a fork.

Quick Reference Table

Command	**Purpose**
`git init`	Initialize new local repository
`git clone URL`	Clone remote repository
`git status`	Show modified/untracked files
`git checkout -b branch`	Create/switch to branch
`git add file`	Stage file for commit
`git commit -m "msg"`	Save snapshot with message
`git log -oneline -graph`	Compact commit history
`git push origin branch`	Push branch to remote
`git pull origin branch`	Update local branch from remote
`git merge branch`	Merge branch into current
`git stash`	Save changes temporarily
`git reset -hard HEAD`	Reset to last commit, discard changes
`git tag -a v1.0 -m "msg"`	Create annotated tag
`git fetch -all`	Download all changes from remote
`git rebase main`	Reapply commits on top of main

Chapter 6

Virtual Machine & Environment Configuration

6.1 Aim

Aim

Provide secure and reproducible access to the course Virtual Machine (VM), ensuring correct environment setup, package management, and robust execution of long-running jobs.

6.2 Access to the Virtual Machine

All computational tasks are executed on a **remote virtual machine (VM)** provided by the course infrastructure. Access requires two steps:

1. **VPN connection**: connect to the university network with OpenVPN.
2. **SSH authentication**: log into the VM using key-based authentication.

```
ssh -i ~/keys/umXX_id_rsa umXX@mXX.lsb.biocomp.unibo.it
```

Listing 6.1: SSH login command

Where: `umXX` = VM username, `mXX.lsb.biocomp.unibo.it` = hostname, `umXX_id_rsa` = private key.

6.3 Configuration of SSH Keys

- Generate key pair locally with `ssh-keygen`.
- Keep the private key (`.rsa`) secure with permission 600.
- Register the public key in the VM.

```
ssh-keygen -t rsa -b 4096 -C "your@email.com"
chmod 600 ~/.ssh/id_rsa
```

Listing 6.2: SSH key generation

```
Security tip:  Never share private keys.  If compromised, revoke
immediately.
```

6.4 Conda Environment Management

The VM is preconfigured with `conda`. A shared environment called `tools` contains required packages.

```
source /opt/conda/bin/activate
conda activate tools
```

Listing 6.3: Activation of course environment

Inspect packages:

```
conda list
```

Export environment for reproducibility:

```
conda env export > environment.yml
```

Recreate elsewhere:

```
conda env create -f environment.yml
```

6.5 Useful Aliases in `.bashrc`

```
# SSH login shortcut
alias lb2vm='ssh -i ~/keys/umXX_id_rsa umXX@mXX.lsb.biocomp.unibo.it'

# Activate conda environment
alias acttools='source /opt/conda/bin/activate && conda activate tools'

# Add SSH key to agent
alias addkey='eval "$(ssh-agent -s)" >/dev/null 2>&1 && \
              ssh-add ~/keys/umXX_id_rsa >/dev/null 2>&1'
```

Listing 6.4: Suggested aliases

6.6 File Transfer

Use `scp` for moving files between local and VM.

```
# Local  VM
scp -i ~/keys/umXX_id_rsa local.txt umXX@mXX.lsb.biocomp.unibo.it:~/remote/

# VM  Local
scp -i ~/keys/umXX_id_rsa umXX@mXX.lsb.biocomp.unibo.it:~/remote/out.txt ~/
   local/
```

Listing 6.5: Secure copy examples

6.7 Execution of Long Jobs

Use `screen` to avoid interruptions due to VPN/SSH disconnections.

```
# Start a screen session
screen -S jobname
# Detach (Ctrl+a, d)
# List sessions
screen -ls
# Reattach
screen -r jobname
```

Listing 6.6: Screen usage

```
Best practice:  Always run clustering or cross-validation inside screen.
```

6.8 Best Practices

- Store all project files inside home directory, not in `/tmp`.
- Always activate `tools` environment before running scripts.
- Use aliases to avoid typing long SSH commands.
- Keep backups of key scripts locally and in GitHub.

6.9 Troubleshooting

- **VPN disconnects**: reconnect and resume via `screen`.
- **Permission denied (publickey)**: check key path and file permissions.
- **Module not found**: ensure correct conda environment is active.
- **Conda conflicts**: rebuild environment from `environment.yml`.

Local Glossary

- **VPN**: Virtual Private Network, secures connection to university network.
- **SSH**: Secure Shell, protocol for remote access.
- **SSH agent**: Utility to manage private keys in memory.
- **Conda environment**: Isolated set of Python packages.
- **Alias**: Custom shell shortcut stored in `.bashrc`.
- **screen**: Terminal multiplexer for persistent sessions.
- **scp**: Secure copy for file transfer via SSH.

Quick Reference Table

Command	Purpose
`openvpn -config students.ovpn`	Connect to VPN
`ssh -i key user@host`	Login to VM
`chmod 600 id_rsa`	Set private key permissions
`conda activate tools`	Activate course environment
`conda list`	Show installed packages
`conda env export > env.yml`	Export environment
`conda env create -f env.yml`	Recreate environment
`alias name='cmd'`	Define alias in `.bashrc`
`scp local remote: /dir`	Copy local → VM
`scp remote: /file local/`	Copy VM → local
`screen -S job`	Start screen session
`screen -ls`	List active sessions
`screen -r job`	Resume screen session

Chapter 7

Data Collection

7.1 Overview

Aim

The objective of this stage is to construct two carefully curated and non-overlapping datasets that form the backbone of the entire predictive pipeline. In particular:

- A **positive set** composed exclusively of proteins with experimentally validated signal peptides, annotated in UniProtKB with strong evidence codes.
- A **negative set** containing proteins that do not carry signal peptides and are localized in non-secretory compartments, explicitly excluding potential confounders such as transmembrane proteins.

By enforcing strict filtering rules (SwissProt entries only, minimum sequence length, removal of fragments, exclusion of automated annotations), this step ensures that subsequent stages of feature extraction and model training are based on high-quality data. The rationale is that no machine learning model, regardless of its complexity, can compensate for noisy or poorly curated input data.

This stage also establishes the principles of **transparency** and **reproducibility**: each query string used in UniProtKB must be documented in the repository, each dataset must be exported both in FASTA format (for sequence-level processing) and TSV format (for annotation-level analysis), and all filtering operations must be explicitly reported.

Ultimately, the aim is to provide a solid, reliable foundation for downstream analysis, guaranteeing biological representativeness, balanced classes, and reproducible datasets that can be re-generated at any time by other researchers or evaluators.

7.2 Rationale

The quality of the input data determines the ceiling of model performance. False annotations or redundant entries directly bias classification. Only curated SwissProt entries are considered to avoid errors introduced by automated annotations (TrEMBL). Negatives must be carefully filtered to exclude transmembrane proteins, as these may mimic SPs.

7.3 Workflow

1. Define UniProtKB query constraints (reviewed, taxonomy, length, experimental evidence, exclusion of fragments).
2. Export results in `.fasta` and `.tsv`.
3. Apply additional curation (removal of TM proteins, fragments).
4. Archive final files into `Data_Collection/` with consistent naming.

7.4 Tools and Outputs

- **UniProtKB interface** (advanced query + batch export).
- **Positive set:** `positive.fasta`, `positive.tsv`.
- **Negative set:** `negative.fasta`, `negative.tsv`.
- Annotation columns preserved: Accession, Protein name, Length, Subcellular location, Evidence code.

7.5 Operational Notes

- Always download both FASTA and TSV to preserve link between sequence and metadata.
- Queries must be copied verbatim into README for reproducibility.
- Ensure no overlap between positive and negative entries.

7.6 Common Pitfalls

- Including TrEMBL entries → noise and false positives.
- Forgetting to exclude TM proteins from negatives → false negatives.
- Exporting only FASTA → loss of metadata and experimental evidence.

Cheat Sheet

```
# Positive set query
(fragment:false) AND (taxonomy_id:2759) AND (length:[40 TO *])
AND (reviewed:true) AND (existence:1) AND (ft_signal_exp:)

# Negative set query
(fragment:false) AND (taxonomy_id:2759) AND (length:[40 TO *])
AND (reviewed:true) NOT (ft_signal:)
```

Chapter 8

Data Preparation

8.1 Overview

Aim

The goal of this phase is to transform raw protein datasets into non-redundant, balanced, and systematically partitioned collections suitable for robust machine learning.
Specifically, this stage ensures that:

- **Redundancy is reduced** by clustering highly similar proteins (MMseqs2) and selecting one representative per cluster, preventing overestimation of performance due to duplicate sequences.
- **Representative IDs are retained** so that each cluster is traceable and metadata integrity is preserved.
- **Datasets are split** into training and benchmark subsets (typically 80/20), ensuring unbiased evaluation.
- **Cross-validation folds are assigned** to the training set (e.g., 5-fold CV), supporting reliable estimation of generalization ability.

This stage establishes the foundation for reproducible and fair evaluation. It ensures that the model learns generalizable sequence features rather than memorizing highly similar proteins.

8.2 Rationale

Machine learning requires balanced, independent data. Redundancy inflates performance, while poorly defined splits introduce information leakage. Clustering ensures diversity, while cross-validation partitions prevent overfitting.

8.3 Workflow

1. Cluster positives and negatives separately with MMseqs2.
2. Extract representative sequences into `rep_*.ids`.
3. Filter cluster mapping tables using custom scripts (`get_tsv.py`).
4. Merge metadata from TSV exports to maintain annotation context.

5. Split into training and benchmark sets (80/20) with `get_sets.py`.
6. Assign CV folds (round-robin) to the training subset.

8.4 Tools and Outputs

- **Tool:** MMseqs2 (`easy-cluster` mode).
- **Scripts:** `get_tsv.py`, `get_sets.py`.
- **Outputs:**
  - `cluster-results_rep_seq.fasta` – representatives.
  - `pos/neg_cluster_results.tsv` – filtered mappings.
  - `pos/neg_train.tsv`, `pos/neg_bench.tsv`.

8.5 Operational Notes

- Always document clustering parameters (`-min-seq-id`, coverage).
- Keep the mapping TSVs: they allow reconstruction of cluster membership.
- Use deterministic seeds (`random_state`) for reproducibility.

8.6 Common Pitfalls

- Running MMseqs2 with insufficient disk space (large `tmp/` dir).
- Forgetting to regenerate `rep.ids` after new clustering.
- Not merging annotation metadata (results in ID-only TSVs).

Cheat Sheet

```
# MMseqs2 clustering
mmseqs easy-cluster positive.fasta pos_cluster tmp \
  --min-seq-id 0.3 -c 0.4 --cov-mode 0 --cluster-mode 1

# Extract representative IDs
grep ">" pos_cluster_rep_seq.fasta | tr -d ">" > rep_positive.ids

# Filter cluster TSV to representatives
python Scripts/get_tsv.py

# Train/benchmark split with CV folds
python Scripts/get_sets.py
```

Chapter 9

Data Analysis

9.1 Overview

Aim

The purpose of this stage is to validate the curated datasets through **exploratory data analysis (EDA)** and to characterize their biological and statistical properties before feature extraction and model training.
Main objectives include:

- Assessing **protein length distributions** in positives and negatives.
- Measuring **signal peptide lengths** from annotated cleavage sites.
- Computing **amino acid composition** of SPs vs SwissProt background.
- Inspecting **taxonomic composition** to avoid species bias.
- Visualizing **cleavage motifs** via sequence logos.

EDA not only ensures dataset quality and coherence, but also provides biological insights into the conserved properties of SPs.

9.2 Rationale

Before training, datasets must be checked for hidden biases (length artifacts, species overrepresentation). EDA ensures that the training and benchmark sets are comparable and that observed patterns reflect biological reality.

9.3 Workflow

1. Load TSV and FASTA files into pandas DataFrames.
2. Plot distributions: sequence length, SP length.
3. Compute amino acid frequencies and compare to SwissProt background.
4. Assess taxonomic composition (kingdom and species levels).
5. Extract windows around cleavage sites and build sequence logos.

9.4 Tools and Outputs

- **Libraries:** pandas, matplotlib, seaborn, logomaker.
- **Outputs:**
- `density_protein_lengths.png`, `SP_lengths.png`.
- `residues_composition.png`.
- Taxonomy plots (kingdom, species).
- Sequence logos (`seq_logo_bench.png`, `seq_logo_bench.png`).

9.5 Operational Notes

- Always normalize distributions (avoid bias from class imbalance).
- Positive sequences without cleavage annotation must be excluded from motif analysis.
- Apply identical analysis pipeline to both training and benchmark.

9.6 Common Pitfalls

- Mixing raw and curated datasets.
- Comparing train vs benchmark without stratification.
- Misaligned sequence logos due to wrong window extraction.

Cheat Sheet

```
# Protein length distribution
sns.histplot(train_df, x="Sequence length", hue="class", stat="density")

# SP length extraction
df_sp["SP_length"] = df_sp["SP cleavage"].astype(int)

# Amino acid composition
aa_freq = {aa: seq.count(aa)/len(seq) for aa in "ACDEFGHIKLMNPQRSTVWY"}

# Sequence logo (logomaker)
mat = logomaker.alignment_to_matrix(sequences=seqs, to_type="information")
logomaker.Logo(mat)
```

Chapter 10

von Heijne Method

10.1 Aim

Aim

Provide a complete description, implementation, and evaluation of the **von Heijne (1986)** statistical model for detecting signal peptides (SPs). The method forms the historical baseline of SP prediction and relies on position-specific amino acid preferences surrounding the cleavage site.

10.2 Biological Rationale

Signal peptides exhibit three canonical regions:

- **N-region**: positively charged (K, R)
- **H-region**: strongly hydrophobic (L, A, V, I)
- **C-region**: polar, containing the **[A/V]-X-A** motif around the cleavage site

Von Heijne observed that the 16 residues surrounding the cleavage site (-13 to $+2$) exhibit reproducible positional patterns of enrichment and depletion. These patterns can be learned as frequency matrices and converted to log-odds scores.

The method is therefore:

- simple
- interpretable
- biologically grounded
- limited in flexibility (cannot model long-range interactions)

10.3 Mathematical Formulation

Given:

- $M_{k,j}$ = observed probability of residue k at position j
- b_k = background probability of residue k in SwissProt

The **Position-Specific Weight Matrix (PSWM)** is:

$$W_{k,j} = \log\left(\frac{M_{k,j}}{b_k}\right)$$

To avoid zeros, a pseudocount $\alpha = 1$ is added:

$$M_{k,j} = \frac{c_{k,j} + \alpha}{N + 20\alpha}$$

A 16-aa sliding window is scored as:

$$S(i) = \sum_{j=1}^{16} W_{x_{i+j-1},j}$$

The score of the sequence is the maximum over the first 90 residues:

$$S_{\max} = \max_{i \in [1,74]} S(i)$$

Classification rule:

$$\hat{y} = \{ 1 \, if S_{\max} > \tau 0 otherwise$$

Where τ is a threshold optimized to maximize the F1-score.

10.4 Implementation Workflow

1. Extract 16-aa windows centered on annotated cleavage sites.
2. Build position-specific frequency matrix.
3. Normalize using SwissProt background.
4. Apply log-odds transformation to obtain PSWM.
5. Slide window across N-terminal region of each protein.
6. Select the maximum PSWM score.
7. Learn optimal threshold τ through 5-fold CV.

10.5 Pseudocode

```
# Build PSWM
pswm = compute_pswm(positive_training_sites)

# Predict
def predict_pswm(seq, pswm, threshold):
    scores = []
    for pos in range(0, 90 - 16):
        window = seq[pos:pos+16]
        s = 0
        for j in range(16):
            aa = window[j]
            s += pswm[j][aa]
        scores.append(s)
    return 1 if max(scores) > threshold else 0
```

10.6 Cross-Validation Results

Table 10.1: 5-fold CV performance (mean ± SE).

Metric	CV	Benchmark
Accuracy	0.939 ± 0.002	0.930
Precision	0.708 ± 0.017	0.665
Recall	0.756 ± 0.032	0.726
F1-score	0.728 ± 0.011	0.694
MCC	0.697 ± 0.013	0.656

10.7 Interpretation

The method shows:

- **high recall**: captures canonical cleavage motifs
- **moderate precision**: many false positives (hydrophobic TM helices)
- **stable behavior across folds**

10.8 Strengths and Limitations

Strengths

- fully interpretable
- captures well-known SP motifs
- computationally cheap

Limitations

- fails on weak/non-canonical hydrophobic cores
- sensitive to SP length variation
- cannot model long-range signals or subtler patterns

Chapter 11

Feature Extraction

11.1 Aim

Aim

Transform raw protein sequences into fixed-length numerical vectors encoding physicochemical properties, amino acid composition, and structural tendencies of the N-terminal region where signal peptides reside.

11.2 Biological Motivation

Signal peptides share characteristic structural patterns:

- charged N-region
- hydrophobic H-core
- polar C-region containing the cleavage motif

Capturing these properties numerically enables machine learning models (SVM, MLP) to learn discriminative decision boundaries.

11.3 Feature Categories

Category	**Description**	**Biological Meaning**
Amino Acid Composition	Frequency of 20 amino acids in first 22 residues	Distinguishes hydrophobic enrichment (SP) vs mixed regions
Hydrophobicity	Kyte-Doolittle mean/max in first 40 residues	Determines the strength of H-core
Charge	Mean and net charge	Captures positively charged N-region
α-Helix Propensity	Mean and max Chou-Fasman values	SPs form stable helices while entering translocon
Size/Volume	Mean/max van der Waals volume	Differentiates bulky TM helices vs compact SPs
TM Propensity	AAindex-based TM index	Key to reducing FP due to TM helices

Total dimensionality: **29 features**.

11.4 Workflow

1. Load curated dataset.
2. Extract N-terminal region (22–40 aa).
3. Compute amino acid composition.
4. Compute physicochemical properties from scales.
5. Normalize to fixed-length numeric vector.
6. Export into `ML_features.tsv`.

11.5 Key Equations

Amino acid composition:

$$f_i = \frac{count(i)}{22}$$

Hydrophobicity:

$$H_{mean} = \frac{1}{n}\sum_j h_j$$

Charge:

$$Q = (K + R) - (D + E)$$

Helical propensity:

$$\alpha_{max} = \max_j p_j^\alpha$$

TM propensity:

$$TM_{mean} = \frac{1}{n}\sum_j t_j$$

11.6 Pseudocode

```
for record in dataset:
    seq = record.seq[:40]
    aa_comp = [seq.count(aa)/22 for aa in AA]
    hydro = kyte_doolittle_scale(seq)
    charge = charge_scale(seq)
    alpha  = alpha_propensity(seq)
    tm     = tm_index(seq)

    features = concatenate([
        aa_comp,
        mean(hydro), max(hydro),
        mean(charge), net(charge),
        mean(alpha), max(alpha),
        mean(tm)
    ])

    save(features)
```

11.7 Common Issues

- Must ensure positive/negative sets include no paralog contamination.
- Missing scale values $\rightarrow$ must clean AAindex.
- Avoid unscaled features $\rightarrow$ kernel distortions.

11.8 Summary

Feature extraction encodes the essential biochemical identity of SPs and supports the training of both classical ML and deep learning approaches.

Chapter 12

Support Vector Machine Classifier

12.1 Aim

Aim

Develop and optimize an SVM classifier capable of discriminating signal peptides from non-SP N-terminal regions using engineered physicochemical features.

12.2 Mathematical Foundation

SVM solves:

$$\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_i \xi_i$$

subject to:

$$y_i(w \cdot \phi(x_i) + b) \geq 1 - \xi_i$$

Where:

- C = regularization strength
- ϕ = mapping into high-dimensional feature space
- ξ_i = slack variables for margin violations

The **RBF kernel** models non-linear decision boundaries:

$$K(x_i, x_j) = \exp\left(-\gamma\|x_i - x_j\|^2\right)$$

12.3 Workflow Summary

1. Load 29-feature matrix.
2. MinMax normalization.
3. Feature ranking via Random Forest Gini.
4. Select top 15 features.
5. Grid search on $(C, \gamma, kernel)$.
6. 5-fold Stratified CV.
7. Train final model.
8. Test on benchmark set.

12.4 Feature Importance

Top contributors:

- Leucine composition (`comp_L`)
- α-helix mean propensity
- Mean charge
- Hydrophobicity max
- TM propensity

12.5 Pseudocode

```
rf = RandomForestClassifier().fit(X_train, y_train)
selected = select_top_k(rf.importances_, k=15)

param_grid = {
    "C": [0.1, 1, 10],
    "gamma": ["scale", "auto"],
    "kernel": ["rbf", "linear"]
}

grid = GridSearchCV(
    SVC(probability=True),
    param_grid,
    scoring="balanced_accuracy",
    cv=5
)

grid.fit(X_train[:, selected], y_train)
y_pred = grid.predict(X_test[:, selected])
```

12.6 Performance

Training/Validation (CV):

- Accuracy = 0.927
- Precision = 0.620
- Recall = 0.857
- MCC = 0.691

Benchmark:

- Accuracy = 0.922
- Precision = 0.594
- Recall = 0.895
- F1 = 0.714
- MCC = 0.690

12.7 Interpretation

The SVM:

- generalizes well
- reduces false positives vs. von Heijne
- still struggles with weak SPs or borderline TM helices

Chapter 13

Evaluation and Comparative Analysis

13.1 Aim

Aim

Provide a quantitative and biological comparison between the **von Heijne** rule-based model and the **Support Vector Machine (SVM)** classifier. The chapter summarises cross-validation behaviour, benchmark performance, confusion matrix structure, and the biological nature of misclassifications.

13.2 Evaluation Framework

The von Heijne and SVM models were evaluated using different data partitions reflecting their respective training procedures. After generating an initial 80/20 train–benchmark split, the von Heijne model used only the 80% training portion, which was internally divided into a 3:1:1 ratio to create dedicated train, validation, and test subsets. The SVM, instead, relied on scikit-learn's `train_test_split` applied to the same 80% training portion, while the held-out 20% benchmark set was reserved exclusively as its independent test set.

13.3 Evaluation Performance on Test Set

The table summarises the performance of the von Heijne and SVM classifiers tested on test set.

Table 13.1: Evaluation performance on Test Set.

Model	**Acc**	**Prec**	**Rec**	**F1**	**MCC**
von Heijne	0.939	0.708	0.756	0.728	0.697
SVM	0.927	0.620	0.857	0.719	0.691

13.4 Benchmark Results

On the fully independent benchmark set, the SVM outperforms the von Heijne model across all major metrics.

Table 13.2: Benchmark performance comparison.

Model	**Acc**	**Prec**	**Rec**	**F1**	**MCC**
von Heijne	0.930	0.665	0.726	0.694	0.656
SVM	0.922	0.594	0.895	0.714	0.690

13.5 Confusion Matrix Analysis

Confusion matrices further highlight systematic differences:

Table 13.3: Confusion matrix comparison on benchmark set.

Model	TN	FP	FN	TP
von Heijne	1707	80	60	159
SVM	1653	134	23	196

Interpretation

- **von Heijne** produces more false positives due to strong reliance on hydrophobicity, often confusing transmembrane helices with signal peptides.
- **SVM** reduces FP by incorporating transmembrane-propensity and physicochemical features, while also slightly lowering the false-negative count.

13.6 Error Profile and Biological Interpretation

13.6.1 False Positives Analysis by Transmembrane Content

- **von Heijne**: 53.8% of false positives correspond to transmembrane helices with strong hydrophobic cores.
- **SVM**: FP rate drops significantly (9.7%) due to features capturing TM propensity and global physicochemical patterns.

13.6.2 False Negatives

Both models struggle on:

- SPs with reduced hydrophobicity in the H-region.
- SPs enriched in S, T, D, E (polar/acidic residues).
- Atypical SP lengths (<18 aa or >25 aa).

13.6.3 Motif-Level Differences

Sequence logos reveal that:

- Correctly predicted SPs show a strong A–X–A motif near the cleavage site.
- False negatives display degradation of this motif and weaker hydrophobic–polar transitions.

13.7 Comparative Summary

- **von Heijne** remains an interpretable, motif-based baseline with good recall.
- **SVM** substantially improves precision and overall performance, reducing transmembrane-derived errors.
- The two models offer complementary perspectives: motif-driven scoring vs. physicochemical discrimination.

13.8 Conclusion

The SVM outperforms the von Heijne model in both cross-validation and benchmark scenarios, particularly in terms of precision and MCC. However, the biologically interpretable nature of the von Heijne method remains valuable for motif-level analysis and for understanding canonical SP architectures. Together, the models highlight both conserved and variable components of signal peptide recognition.

Chapter 14

Deep Learning: Multilayer Perceptron

14.1 Aim

Aim

Develop a high-performance signal peptide classifier by combining pre-trained ESM-2 representations with a tuned Multilayer Perceptron. The goal is to achieve strong generalization with minimal feature engineering, leveraging transfer learning and automated hyperparameter optimization.

14.2 Biological and Computational Rationale

Protein language models (pLMs) encode sequence regularities shaped by evolution. ESM-2 (T33_650M) captures:

- structural and physicochemical context
- long-range dependencies between residues
- evolutionary constraints learned from massive MSAs
- subtle N-terminal patterns relevant for SP recognition

Signal peptides are short, motif-rich signals confined to the N-terminus. Embedding only the first 90 residues reduces noise from the mature protein region and increases sensitivity to biologically relevant cues (as adopted in the optimized workflow). This approach follows the updated experimental design in the project repository and greatly improves the classifier's discriminative power.

14.3 Embedding Extraction

We used:

- **Model:** ESM-2 (T33_650M_UR50D)
- **Embedding dimension:** 1280 per residue
- **Sequence window:** first 90 amino acids
- **Pooling:** mean pooling across the selected residues

Final protein embedding:

$$\mathbf{z} = \frac{1}{90}\sum_{i=1}^{90} h_i, \qquad h_i \in \mathbf{R}^{1280}$$

14.4 MLP Architecture

The MLP architecture was optimized with Optuna. Trial 7 provided the best validation MCC.

Final Model

- Input: 1280D pooled embedding
- Hidden Layers: 35 – 35 – 34 neurons
- Activation: ReLU
- Dropout: $p = 0.115$
- Output: sigmoid (binary classifier)

Training Setup

- Loss: Binary cross-entropy
- Optimizer: Adam
- Learning rate: 1.552×10^{-4}
- Early stopping: patience = 10 epochs
- Hyperparameter optimization: Optuna (10 trials)
- Model selection criterion: maximize Validation MCC

14.5 Pseudocode

```
model = MLP([
    Linear(1280, 35), ReLU(), Dropout(0.115),
    Linear(35, 35),   ReLU(), Dropout(0.115),
    Linear(35, 34),   ReLU(), Dropout(0.115),
    Linear(34, 1),    Sigmoid()
])

train(model, embeddings_train, y_train,
      lr=1.552e-4, early_stopping=10)
evaluate(model, embeddings_test, y_test)
```

14.6 Training Dynamics

Training curves indicate:

- rapid convergence due to highly informative ESM-2 embeddings
- early stopping triggered at epoch 28
- minimal train–validation gap, confirming robust generalization

Best validation MCC: **0.984** at epoch 17.

14.7 Benchmark Performance

Evaluation on the independent blind test set (N=2006):

- Accuracy = **0.990**
- Precision = **0.971**
- Recall = **0.936**
- F1-score = **0.953**
- MCC = **0.948**

Confusion matrix:

	Pred.−	*Pred.*+
Actual−	1781	6
Actual+	14	205

14.8 Interpretation

The optimized MLP achieves near–state-of-the-art performance:

- extremely low false positive (6) and false negative (14) rates
- reliable detection of both canonical and weak/non-canonical SPs
- effective handling of class imbalance
- strong synergy between ESM-2 features and lightweight MLP classifier

This demonstrates that transfer learning from protein language models combined with minimal but well-targeted neural architectures yields powerful and efficient SP predictors, outperforming motif-based or handcrafted-feature models.