Machine Learning Project Report: Protein Subcellular Localization

Group Members: JAWAAD ZAKI AHMED JARIWALA, ANKUSH MAKHIJANI, JAYVIRSINH CHUDASAMA **Date:** 2025-04-29

1. Introduction

This report details the process and results of a machine learning project aimed at predicting the subcellular localization of Gram-positive bacterial proteins. The goal is to classify proteins into one of four possible functional locations within the cell based on sequence-derived features. This is a multi-class classification problem. The project follows the steps outlined in the course description, utilizing Python and the scikit-learn library.

2. Data and Feature Usage

2.1. Data Source

The data used for this project was provided via Canvas (`Subcellular Data`), specifically the `comp_occur.csv` file. This file contains pre-computed features representing amino acid composition and occurrence for various protein sequences.

2.2. Feature Selection

As permitted by the project description, we utilized the existing features available in `comp_occur.csv` rather than performing primary sequence-based feature extraction. The dataset contained 40 feature columns. The first column represented the class label (protein location: 'Fold1', 'Fold2', 'Fold3', 'Fold4'), and the remaining 40 columns were used as input features (X) for the models.

3. Methodology

3.1. Data Preparation

The following steps were performed to prepare the data for modeling using the mlproject_test_2.py script:

1. **Loading:** The `comp_occur.csv` dataset was loaded using the pandas library.
2. **Label Encoding:** The categorical class labels ('Fold1', 'Fold2', 'Fold3', 'Fold4') were converted into numerical representations (0, 1, 2, 3 respectively) using `sklearn.preprocessing.LabelEncoder` as required by the machine learning models. The mapping was: 'Fold1': 0, 'Fold2': 1, 'Fold3': 2, 'Fold4': 3.
3. **Train-Test Split:** The dataset was split into a training set (75%) and an independent test set (25%) using `sklearn.model_selection.train_test_split` . Stratification based on the class labels (`stratify=y`) was used to ensure similar class distributions in both sets. The `random_state` was set for reproducibility.
4. **Feature Scaling:** Feature scaling is crucial for distance-based algorithms like KNN and SVM. `sklearn.preprocessing.StandardScaler` was used to standardize features by removing the mean and scaling to unit variance. To prevent data leakage, scaling was incorporated into a `sklearn.pipeline.Pipeline` for each model, ensuring the scaler was fit only on the training data during both cross-validation and final model training.

3.2. Classification Models

Three different classification algorithms from scikit-learn were implemented and evaluated:

1. **K-Nearest Neighbors (KNN):** A non-parametric, instance-based learning algorithm. Parameters: `n_neighbors=18` , `metric='minkowski'` , `p=2.5` .
2. **Support Vector Machine (SVM):** A powerful classifier effective in high-dimensional spaces. Parameters: `kernel='rbf'` , `decision_function_shape='ovo'` , `class_weight='balanced'` . The `class_weight='balanced'` parameter was used to address the observed class imbalance by giving more weight to minority classes.
3. **Random Forest:** An ensemble method based on decision trees. Parameters: `n_estimators=10000` , `criterion='entropy'` , `class_weight='balanced'` , `n_jobs=-1` . `class_weight='balanced'` was also used here to counteract class imbalance.

4. Evaluation and Results

The performance of each model was evaluated using two methods as required:

4.1. k-Fold Cross-Validation

5-fold cross-validation was performed on the *training data* for each model pipeline. This provides a more robust estimate of model performance by training and validating on different subsets of the training data. The average accuracy and standard deviation across the 5 folds were calculated.

- **KNN:** CV Accuracy (5-fold): 64.02 % (+/- 3.90 %)
- **SVC (balanced):** CV Accuracy (5-fold): 69.64 % (+/- 2.68 %)
- **Random Forest (balanced):** CV Accuracy (5-fold): 69.36 % (+/- 5.66 %)

4.2. Independent Test Set Evaluation

Each model pipeline, once trained on the *entire* training set, was used to predict labels for the unseen independent test set. The following metrics were calculated:

- Overall Accuracy
- Confusion Matrix
- Classification Report (Precision, Recall, F1-score per class)

Test Set Results Summary:

Model	Accuracy	Macro Avg Precision	Macro Avg Recall	Macro Avg F1-Score	Weighted Avg F1-Score
KNN	67.94%	0.54	0.50	0.50	0.66
SVC (balanced)	77.10%	0.61	0.59	0.59	0.76
RandomForest (balanced)	73.28%	0.58	0.55	0.55	0.71

(Note: Macro averages give equal weight to each class, weighted averages account for class support.)

Detailed Test Set Performance (from mlproject_test_2.py run):

- **KNN:**
  - Confusion Matrix: `[[25 0 16 3], [0 0 1 3], [3 0 48 1], [1 0 14 16]]`
  - Struggled significantly with 'Fold2' and 'Fold4'. Poor recall for 'Fold1'. Excellent recall for 'Fold3'.
- **SVC (balanced):**
  - Confusion Matrix: `[[27 1 11 5], [0 0 1 3], [1 1 48 2], [0 0 5 26]]`
  - Highest overall accuracy. Good performance on 'Fold1', 'Fold3', 'Fold4'. Still failed to correctly predict any 'Fold2' samples despite class weighting. Misclassified some samples *as* 'Fold2'.
- **Random Forest (balanced):**
  - Confusion Matrix: `[[24 0 15 5], [0 0 1 3], [1 0 50 1], [0 0 9 22]]`
  - Good accuracy, slightly below SVC. Excellent recall for 'Fold3', reasonable for 'Fold4'. Still failed completely on 'Fold2'. Poor recall for 'Fold1'.

5. Discussion and Analysis

The results show that the Support Vector Classifier (SVC) achieved the best overall performance on the independent test set (77.10% accuracy), slightly outperforming Random Forest (73.28%) and KNN (67.94%). Cross-validation results also indicated SVC and Random Forest were potentially stronger models than KNN for this dataset.

A major challenge identified was significant **class imbalance**. The 'Fold2' class was severely under-represented in the dataset (only 4 samples in the test set). This led to all models performing extremely poorly on this class, failing to correctly identify any 'Fold2' samples in the test set, resulting in 0 recall and F1-score for this class.

Attempting to mitigate this using the `class_weight='balanced'` parameter in SVC and Random Forest did not yield significant improvements for 'Fold2' prediction in this specific case, although SVC's overall performance remained the highest. The models still struggled to learn the patterns for this rare class.

The confusion matrices reveal common misclassification patterns. For instance, 'Fold1' samples were often misclassified as 'Fold3' or 'Fold4' by all models. 'Fold2' samples were exclusively misclassified as 'Fold3' or 'Fold4'.

6. Conclusion and Future Work

This project successfully implemented a machine learning pipeline for protein subcellular localization prediction using pre-computed features. Three models (KNN, SVC, RandomForest) were trained and evaluated using cross-validation and an independent test set. SVC demonstrated the highest overall accuracy.

However, the severe class imbalance, particularly for 'Fold2', significantly impacted model performance for that class. Future work could explore more advanced techniques to handle this imbalance, such as:

- **Oversampling Techniques:** Implementing SMOTE (Synthetic Minority Over-sampling Technique) using libraries like `imbalanced-learn` to generate synthetic data for minority classes.
- **Different Algorithms:** Testing algorithms known to be less sensitive to imbalance or alternative ensemble methods.
- **Feature Engineering/Selection:** Exploring different feature sets or performing feature selection to identify the most discriminative features.
- **Hyperparameter Tuning:** Optimizing hyperparameters for each model (e.g., using GridSearchCV or RandomizedSearchCV) could potentially improve performance.

The implemented Python code (mlproject_test_2.py) accompanies this report, including comments explaining the steps.