Protein Structure Prediction
A Multiclass Classification Problem

Math 156: Machine Learning Final Project Group 9

Andrew Zhang, Aryan Dalal
Steven Villanueva, Jiwoo Hyun

University of California, Los Angeles (UCLA)

Introduction: The Problem, Dataset and Goal

Problem: Investigate how increasing the size of the ESM-2 Protein Inference model improves its accuracy in secondary structure prediction.

Dataset: Raw dataset consists of 1,440,640 rows of Amino acids taken from 6148 unique protein sequences. Each row contains an amino acid's identity, the primary structure, its secondary structure (our target variable). 650M, 150M and 35M produce 1280, 640 and 480 dimensional embeddings respectively.

Questions: Does a five-fold increase in embedding dimension improve model performance? are certain models inherently better suited for secondary structure prediction?

Approaches

For predicting protein structure, we select three models: *Logistic Regression*, *Linear Support-Vector Machine*, and *Multi-layer Perceptron*.

We intend to evaluate these three models across three ESM embedding sizes (35M, 150M, and 650M).

Our goal is to understand how the choice of model and the dimension of the embeddings jointly influence classification accuracy and class-balanced performance for protein secondary structure prediction.

The data is split in 70/30 Train/Test respectively. Hyperparameters chosen using 3-fold cross-validation on 150M embeddings (applied to 35M, 150M and 650M).

Evaluation

We use **accuracy**, **macro F1 score** and **confusion matrices** to evaluate all three proposed models.

We note that the dataset exhibits **strong class imabalance** so macro F1 is included to equally weight performance across all nine structural classes.

Confusion matrices are used to visualize class-specific prediction patterns and illustrate the effects of imbalance on each model.

Model 1: Logistic Regression

Definition: A probabilistic linear classifier.
Optimization: Maximum likelihood (cross-entropy) with L2:

$$\widehat{\mathbf{W}} = \arg\min_{\mathbf{W}} \underbrace{\sum_{n=1}^{N} \log\left(\sum_{q=1}^{k} e^{\phi(x_n)^\top w_q}\right) - \mathrm{tr}(\mathbf{Y}^\top \Phi^\top \mathbf{W})}_{\text{negative log-likelihood}} + \frac{\lambda}{2}\|\mathbf{W}\|_F^2, \quad \lambda = \frac{1}{C}.$$

Solution: No closed form, but we can use iterative algorithms NR or GD. Here we used GPU `cuML` quasi-Newton (QN) solver or Newton-Ralphson.

Model 2: Linear Support Vector Machine (SVM)

Definition: A linear classifier that seeks a decision boundary maximizing the margin between classes. The Linear SVM optimizes a margin-based hinge-loss objective rather than relying on misclassification errors alone.

Optimization: We solve the convex problem

$$\widehat{\mathbf{w}} = \arg\min_{\mathbf{w}\in\mathbb{R}^d} \left(\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N} \max(0,\, 1 - y_i\, w^\top x_i) \right)$$

Solution: We used GPU cuML implementation of `LinearSVC`, which employs a scalable quasi-Newton–based optimizer suitable for high-dimensional embeddings.

Hyperparameter Selection: $C = 0.5$ and `max_iter` $= 2000$ were chosen from the search ranges $C \in \{0.5, 1, 2, 5, 10, 15\}$ and `max_iter` $\in \{2000, 3000, 5000, 8000\}$.

Model 3: 3-Layer Multi-Layer Perceptron (MLP)

Definition: A fully-connected feedforward neural network with 2 hidden layers consisting of a $(n, 512, 256, 9)$ architecture for $n \in \{1280, 640, 480\}$.

Optimization: We seek to solve the optimization problem

$$\widehat{\mathbf{w}} = \underset{\mathbf{w} \in \mathbb{R}^{512}}{\arg\min} \left[\ell_{\text{MLP}} = \sum_{s=1}^{n} \ell\left(\mathbf{y}_s, \widehat{\mathbf{y}}\left(\mathbf{x}_s; \mathbf{w}\right)\right) \right] = \underset{\mathbf{w} \in \mathbb{R}^{512}}{\arg\min} \left[-\sum_{i=1}^{n} \mathbf{1}\left(\mathbf{y}_s = \text{class } i\right) \log\left(\hat{y}_i\left(\mathbf{x}_s; \mathbf{w}\right)\right) \right]$$

Solution: Non-Convex Optimization Problem–hard to solve for global minimum. We find local minimum via Adaptive Moment Estimation (Adam)–*an extension of Stochastic Gradient Descent*.

Activation: GELU (Gaussian Error Linear Unit). *Chosen over ReLU for its smoothness and non-monotonic properties.*

3-Layer MLP Visualization

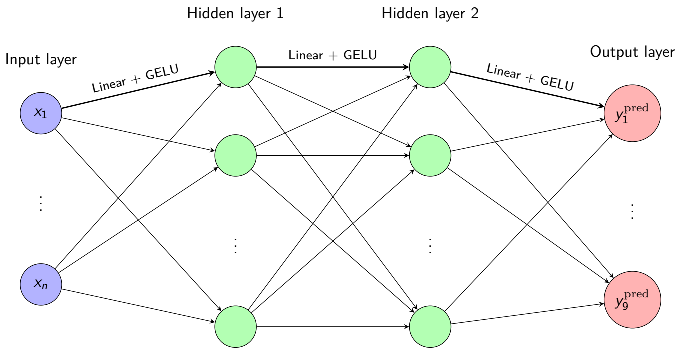
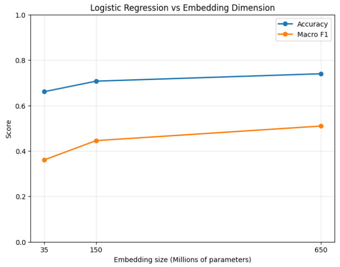
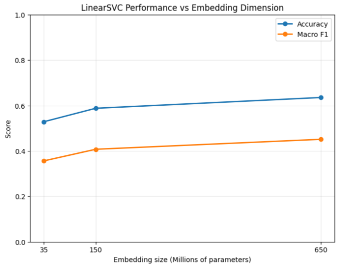


Figure: 3-Layer Multi-layer Perceptron Classifier with TikZ/pgfplots

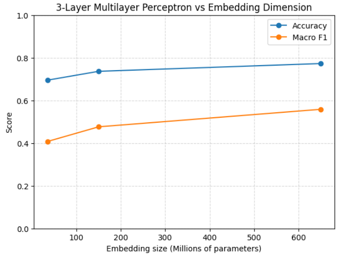
Results & Performance



(a) Logistic Regression



(b) LinearSVC



(c) MLP

Figure: Performance across embedding sizes for all models.

Summarized Results

Table: Performance comparison across embedding sizes for all models

Model	Embedding Size	Accuracy	Macro F1
Logistic Regression	35M	0.6608	0.3606
	150M	0.7071	0.4454
	650M	**0.7396**	**0.5094**
LinearSVC	35M	0.5286	0.3563
	150M	0.5877	0.4073
	650M	**0.6352**	**0.4511**
3-Layer MLP (512, 256)	35M	0.6951	0.4078
	150M	0.7363	0.4764
	650M	**0.7729**	**0.5584**

Conclusion/Improvements

Conclusion:

1) Embedding quality dominates model choice in this task.
2) Larger ESM-2 embeddings improves balanced multi-class performance.
3) 650M variant delivering the strongest outcomes
4) Logistic Regression performed best with cost/benefit consideration.

Improvements:

1) We fixed hyperparameters tuned on 150M when evaluating other sizes; per-size retuning could close small gaps.
2) More targeted imbalance handling (e.g., calibrated class weights, focal loss, or cost-sensitive thresholds) may increase Macro-F1 for rare classes.

Acknowledgements

The dataset used in this project is from UCLA CS C121: “Probabilistic Models in Computational Genomics” developed by Prof. Eleazar Eskin. Our raw data comes from this course’ dataset.

The ESM-2 Models were released by Meta alongside their “Evolutionary-scale Prediction of Atomic Level Protein Structure with a Language Model” paper. They were accessed through the HuggingFace transformers package.

The ESM-2 Embeddings and building the Feedforward Neural Network were done with PyTorch, using CUDA on Google Colab’s A100 GPU. We used cuML packages to run LinearSCV and LogisticRegression on the GPU.

Evaluation were done with Scikit-learn.metrics. Visualizations done with Matplotlib and Seaborn.

Thank you!