Peptide Classification Competition

Group 48

1. Introduction

In this competition, we set out to build a predictive model to classify peptide sequences into positive (+1) or negative (-1) classes. Our goal was to leverage our machine learning expertise to accurately distinguish between these classes using sequence data. Throughout the project, we ensured that all ethical practices were followed, including strict adherence to the team naming conventions and secure handling of provided data.

---

2. Problem Statement

We were provided with a training dataset consisting of peptide sequences along with their corresponding class labels. The test dataset, on the other hand, contained peptide sequences without any labels. Our task was to develop a classification model that predicts whether each peptide is positive (+1) or negative (-1). We evaluated the performance of our model using the AUC (ROC) metric, which allowed us to assess how well our model could rank positive and negative examples.

---

3. Data Description

Training Data (train.csv)

Content:
Our training dataset included peptide sequences along with their corresponding class labels (+1 or -1).

Key Variables:

- **Sequence:** The amino acid sequence of the peptide.
- **Label:** The class of the peptide, where +1 indicates a positive peptide and -1 indicates a negative peptide.

Test Data (test.csv and submission_test_final.csv)

Content:
The test dataset provided peptide sequences without labels. Our task was to predict the labels for these sequences.

Additional Files

- **Notebook (classificatioin_peptide.ipynb):**
  Our notebook contains the code, outputs, and detailed comments that explain every step of the model building, evaluation, and submission process.
- **Pretrained Model (best_model.pkl):**
  This file holds the best model we developed during the competition.
- **Query Data (query.fasta):**
  We used this additional FASTA file containing peptide sequences for further analysis and testing purposes.

---

4. Methodology

4.1 Data Preprocessing

Our first step was to preprocess the data to ensure its quality and consistency. Here's what we did:

- **Handling Missing Data:**
  We performed checks for missing or invalid sequences and handled any missing values appropriately.

- **Sequence Cleaning:**
  We verified that each sequence was consistent with standard amino acid codes. Any non-standard symbols were corrected or removed.

- **Label Encoding:**
  We encoded the labels as binary values, with +1 representing positive peptides and -1 representing negative peptides.

4.2 Feature Extraction

Feature extraction was a crucial component of our pipeline, and we implemented two primary methods to extract meaningful representations from peptide sequences:

a. PSI-BLAST Based PSSM Features

Function: `run_psiblast`
We used this function to run PSI-BLAST on each peptide sequence. It writes the sequence to a FASTA file, runs PSI-BLAST against the specified database (defaulting to 'nr'), and retrieves a Position Specific Scoring Matrix (PSSM). We then average the scores across sequence positions to produce a fixed-length feature vector.

Function: `extract_pssm_features`
This function iterates over every sequence in our DataFrame and applies `run_psiblast`. If PSI-BLAST fails for any sequence, we handle the error by substituting a zero vector. This ensures that our feature extraction process is robust.

b. ESM-2 Embeddings

Function: `extract_esm2_embeddings`
We leveraged the ESM-2 model from Facebook to generate embeddings for each peptide sequence. In this process:

- We load the pretrained tokenizer and model (facebook/esm2_t6_8M_UR50D).
- Each sequence is tokenized (with proper padding and truncation) and passed through the model in evaluation mode.
- We then average the hidden state outputs over the sequence length to produce a fixed-size embedding vector.

After extracting both the PSSM features and ESM-2 embeddings for our training and test datasets, we concatenated these features to create the final input matrices:

- **Training Features:** Combined from `X_train_pssm` and `X_train_esm`
- **Test Features:** Combined from `X_test_pssm` and `X_test_esm`

Finally, we scaled the features using a `StandardScaler` to normalize the data before feeding it into our classification models.

4.3 Model Development

In our model development phase, we built and evaluated several classification models using both base estimators and ensemble methods. Below is a detailed explanation of our process:

Base Estimators

We started by defining a variety of base estimators with carefully optimized parameters. Our models included:

- **Random Forest:** Configured with 500 trees, using a square root strategy for selecting maximum features, and balanced class weights.

- **XGBoost:** Set with 300 estimators, a learning rate of 0.05, maximum depth of 6, and other parameters such as colsample_bytree and subsample tuned for performance.
- **SVM:** Using an RBF kernel with probability estimates enabled, along with balanced class weights.
- **CatBoost:** Configured for 300 iterations and a depth of 6 with a learning rate of 0.05.
- **LightGBM (LGBM):** Set with 300 estimators and a learning rate of 0.05.
- **Gradient Boosting (GBM):** Configured with 300 estimators, a learning rate of 0.05, and a maximum depth of 6.
- **AdaBoost:** Configured with 300 estimators and a learning rate of 0.05.
- **MLP (Multilayer Perceptron):** Using two hidden layers (100, 50) and a maximum of 300 iterations.

Ensemble Methods

To boost performance and improve generalization, we implemented two ensemble strategies:

Stacking Ensemble:
We combined our base estimators using a stacking approach, where the predictions from the individual models served as input features to a final estimator (LightGBM). This ensemble was designed to capture complementary strengths across models.

Voting Ensemble:
We also created a voting ensemble that aggregates predictions using soft voting. Each base estimator's predicted probabilities were weighted to reflect their individual performance, and the final prediction was based on the weighted average.

Both ensembles were stored along with the individual base estimators in a dictionary for ease of evaluation.

Hyperparameter Tuning for XGBoost

Recognizing the importance of fine-tuning, we performed hyperparameter tuning specifically for the XGBoost model. Using `RandomizedSearchCV` with 15 iterations and a stratified 3-fold cross-validation, we explored different values for:

- Learning rate
- Maximum depth
- Column sample by tree
- Subsample ratio
- Minimum child weight

This tuning helped us identify the best set of parameters, and we updated our XGBoost model accordingly in our model dictionary.

Model Evaluation and Selection

We then evaluated all the models on our validation set:

- Each model was trained on the scaled training data.
- We computed the AUC (ROC) score on the validation set by using the predicted probabilities.
- We tracked the model with the highest AUC score to determine the best performing approach.

The best performing model was identified and saved as `best_model.pkl` for later use.

Final Predictions with the Stacking Ensemble

For our submission, we decided to use the stacking ensemble:

- We re-trained the stacking classifier on the full training set.
- After confirming its performance on the validation set (using AUC as the metric), we generated predictions on the test set.
- The predicted probabilities were then formatted into a submission file (`submission_test_final.csv`), which included the test IDs and the corresponding predicted labels.

This comprehensive model development strategy allowed us to leverage both individual model strengths and ensemble learning to achieve robust performance on the peptide classification task.

---

5. Experimental Results

Our experimental results, as documented in our notebook, highlight both the strengths of our approach and some compatibility warnings encountered during model persistence. Below are the key highlights:

Validation Metrics for the Best Model and Model Evaluation:

Table 1: Model Performance (AUC Scores)	
Model	AUC Score
Random Forest	0.7988

XGBoost	0.8578
SVM	0.8243
CatBoost	0.8467
GBM	0.8223
AdaBoost	0.8483
MLP	0.8242
Voting Ensemble	0.8341
Stacking Ensemble	0.8949

Table 2: Best Model (Stacking Ensemble) – Validation Metrics

Metric	Value
Accuracy	0.7718
Macro Average Precision	0.79
Macro Average Recall	0.76
Macro Average F1 Score	0.76
Weighted Average Precision	0.79
Weighted Average Recall	0.77
Weighted Average F1 Score	0.77
Final Validation AUC (Stacking)	0.901

The Stacking ensemble, which combined multiple base models with LightGBM as the final estimator, emerged as the best model.

Final Stacking Ensemble Performance:
After re-training the stacking ensemble on the entire training set, we obtained a validation AUC of 0.9010. We then generated the final predictions on the test set and saved them as `submission_test_final.csv`.

Overall, our experimental evaluation confirms that the stacking ensemble, with its robust performance and highest AUC score, is our best model for the peptide classification task.

---

6. Discussion

During our project, we encountered several challenges and learning opportunities, many of which were reflected in our code implementation:

Challenges

- **Feature Representation:**
  Capturing the nuances of peptide sequences required us to experiment with multiple feature extraction methods. We implemented two distinct approaches:
    - **PSI-BLAST PSSM Features:** Using the `run_psiblast` and `extract_pssm_features` functions, we generated position-specific scoring matrices (PSSMs) and averaged them over the sequence length.
    - **ESM-2 Embeddings:** The `extract_esm2_embeddings` function leverages a pretrained ESM-2 model to obtain deep sequence embeddings.
      Balancing and combining these heterogeneous features was critical to capture both evolutionary and contextual information inherent in the peptide sequences.
- **Model Generalization:**
  Balancing the complexity of our models was crucial to avoid overfitting, particularly given our relatively small dataset. We tackled this challenge by performing a stratified train-validation split and applying feature scaling using `StandardScaler` to normalize our combined feature set.

- **Version and Compatibility Warnings:**
  While running our experiments, we encountered several `InconsistentVersionWarning` messages related to unpickling estimators (e.g., DecisionTreeClassifier, RandomForestClassifier, SVC) from previous scikit-learn versions. These warnings, though not directly affecting our immediate results, highlighted the importance of maintaining consistency in library versions when deploying or sharing models.

Strengths

- **Robust Preprocessing and Feature Extraction:**
  Our meticulous data cleaning and validation ensured high-quality input features. By integrating traditional bioinformatics methods (PSI-BLAST) with modern deep learning

embeddings (ESM-2), we created a rich and complementary feature representation for each peptide.

- **Diverse Modeling Approaches:**
  We explored a wide array of base estimators—including Random Forest, XGBoost, SVM, CatBoost, Gradient Boosting, AdaBoost, and MLP—and advanced ensemble methods (Stacking and Voting classifiers). Hyperparameter tuning using `RandomizedSearchCV` for XGBoost further improved our model's discriminative capability. This diverse approach allowed us to compare performance across models and ultimately select the best-performing stacking ensemble.

Limitations

- **Data Size:**
  The relatively small dataset might limit our ability to capture rare sequence patterns, potentially affecting the generalizability of our model.

- **Biological Interpretability:**
  While our model achieved strong statistical performance (with high AUC scores), further work is needed to extract and interpret the biological insights from the features, especially from the deep embeddings produced by ESM-2.

Overall, our code-driven approach—from feature extraction with PSI-BLAST and ESM-2, through rigorous model evaluation and hyperparameter tuning, to the final ensemble strategy—enabled us to effectively address these challenges. Our experience underscored the importance of integrating both traditional and modern techniques, as well as the need to manage dependencies and versioning issues when working with evolving machine learning libraries.

---

7. How To Run Models by Command Line

To run the classification_peptide.py script from the command line, follow these steps:

- Open a terminal or command prompt.
- Navigate to the directory containing the classification_peptide.py file. For example:

```
cd c:\Users\vmcsa\Downloads\S_Group_48_01\S_Group_48\S_Group_48
```

- If both input files train.csv and test.csv are in the same folder as the script and want output in same folder, you can run the script like this:

```
python classification_peptide.py train.csv test.csv submission_test_final.csv
```

- Run the script with the required arguments: the path to the training dataset, the path to the test dataset, and the path to save the output predictions. For example:

```
python classification_peptide.py path\to\train.csv path\to\test.csv submission_test_final.csv
```

The Output will be displayed on the command prompt panel.

---

8. Conclusion

We successfully developed a robust machine learning pipeline for peptide classification using the provided sequence data. Our integrated approach—combining rigorous data preprocessing, advanced feature extraction through PSI-BLAST PSSM and ESM-2 embeddings, and comprehensive model development with diverse base estimators and ensemble methods—resulted in a competitive performance with a final competition score of 0.9107. Throughout the project, we strictly adhered to competition guidelines and maintained high ethical standards, ensuring that our methods were both reproducible and transparent.

In summary, our work not only demonstrates the effectiveness of combining traditional bioinformatics tools with modern deep learning techniques but also highlights the importance of a well-structured pipeline in addressing complex classification tasks. By meticulously cleaning the data, extracting complementary features, and evaluating multiple models through rigorous cross-validation and hyperparameter tuning, we were able to achieve strong discriminative power and robust performance. The success of our approach is evident in the high competition score, and our results underscore the potential for further innovations in peptide classification using similar hybrid methodologies.

---

9. Appendix

Code and Outputs

Our notebook (`classificatioin_peptide.ipynb`) contains the complete code, detailed outputs, and comprehensive comments that document every step—from data preprocessing to

final model evaluation. We also included visualizations such as ROC curves, confusion matrices, and feature importance plots to illustrate our model's performance.

Supplementary Files

- **Pretrained Model:** `best_model.pkl` – This is our serialized final model.
- **Datasets:**
  - `train.csv` – Contains the training data with peptide sequences and labels.
  - `test.csv` – Contains the test data with peptide sequences.
  - `submission_test_final.csv` – The final submission file format.
- **Additional Data:**
  - `query.fasta` – Provides additional peptide sequences for extended analysis.