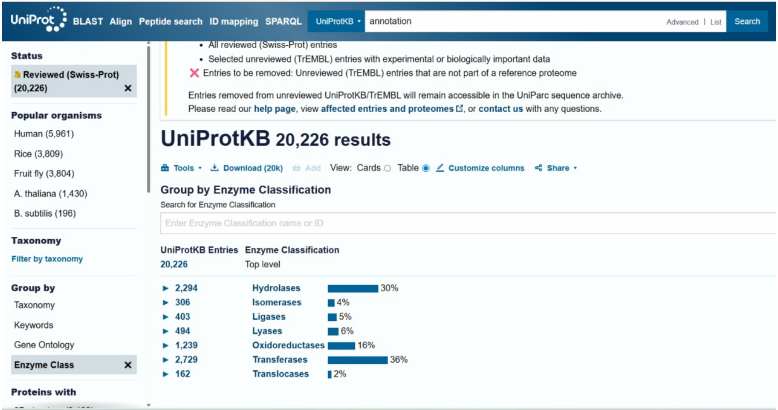
Enzyme Classification Using ProtBERT

December 17, 2025

Dataset Collection

- I gathered the dataset from **UniProtKB**
- I selected only proteins annotated with EC numbers



Environment Setup and Data Loading

- I installed all required libraries for the project
- I used `transformers` and `datasets` for model training
- I used `scikit-learn` for data processing and evaluation
- I loaded the UniProtKB reviewed enzyme dataset using Pandas
- The dataset was read directly from a compressed TSV file

```
[ ]  !pip install -q transformers datasets evaluate scikit-learn accelerate sentencepiece

     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 84.1/84.1 kB 3.4 MB/s eta 0:00:00
```

```python
[ ]  import pandas as pd
     df = pd.read_csv(
         "uniprotkb_reviewed_ec.tsv.gz",
         sep="\t",
         compression="gzip"
     )
```

Dataset Exploration

- I inspected the dataset structure and column names
- I displayed the first few protein entries to understand the data format
- The dataset contains protein identifiers, sequence names, amino acid sequences, and EC numbers
- I verified the total number of sequences in the dataset
- Initial dataset size: **20,226 protein sequences**

```
[ ]
print(df.head())
print(df.columns)
print("Total sequences:", len(df))
```

```
         Entry   Entry Name                                           Sequence  \
0  A0A0B4K692   NEP2_DROME  MQTVIQNPNWWRRRNKLEKSLLVSLGIMFVVLATGFGLWIGKVLRT...
1  A0A0B4K7J2   RBP2_DROME  MFTTRKEVDAHVHKMLGKLQPGRERDIKGLAVARLYMKVQEYPKAI...
2  A0A0B4KEE4    KOI_DROME  MSENTYQIETRRRSRSKTPFLRSSCDHENCEHAGEEGHVHHLKRKS...
3  A0A0B4KGY6   NOVA_DROME  MESIMKVAMDKAAEQLIQQFGFDYLQQQLQLQHQNQHNSSPQQPQH...
4  A0A0B4LFY9  STING_DROME  MAIASNVVEAGNAVRAEKGRKYFYFRKMIGDYIDTSIRIVATVFLA...

   EC number
0  3.4.24.11
1    2.3.2.-
2        NaN
3        NaN
4        NaN
Index(['Entry', 'Entry Name', 'Sequence', 'EC number'], dtype='object')
Total sequences: 20226
```

Removing Non-Enzyme Proteins

- I removed protein entries that do not have an EC number
- Proteins without EC annotations are not enzymes and cannot be used for classification
- This step ensures that all remaining samples belong to valid enzyme classes
- Dataset size after filtering: **7,441 enzyme sequences**

```
[ ]
# Remove proteins without EC numbers
df = df.dropna(subset=["EC number"])

print("After removing non-enzymes:", len(df))
```

```
... After removing non-enzymes: 7441
```

EC Class Extraction and Label Definition

- I handled proteins annotated with multiple EC numbers by keeping the first EC entry
- I extracted the first EC digit to represent the top-level enzyme class (EC level 1)
- I kept only valid enzyme classes ranging from EC1 to EC7
- This step defines a 7-class multi-class classification problem

```
# If multiple EC numbers exist, keep the first
df["EC_number"] = df["EC number"].apply(lambda x: x.split(";")[0])

# Extract first EC digit
df["ec_class"] = df["EC_number"].apply(lambda x: x.split(".")[0])

# Keep only valid enzyme classes 1-7
df = df[df["ec_class"].isin([str(i) for i in range(1, 8)])]

print(df["ec_class"].value_counts())
```

```
ec_class
2    2705
3    2225
1    1239
4     439
6     383
5     289
7     161
Name: count, dtype: int64
```

Label Mapping and Final Dataset Format

- I mapped each EC class to its corresponding enzyme family name
- I converted EC class identifiers into numeric labels (0–6) for model training
- I kept only the sequence, numeric label, and class name for the final dataset
- This format is directly compatible with transformer-based classification models

```
ec_map = {
    "1": "Oxidoreductases",
    "2": "Transferases",
    "3": "Hydrolases",
    "4": "Lyases",
    "5": "Isomerases",
    "6": "Ligases",
    "7": "Translocases"
}

df["label_name"] = df["ec_class"].map(ec_map)
df["label"] = df["ec_class"].astype(int) - 1  # 0-6

df = df[["Sequence", "label", "label_name"]]
df.head()
```

Train, validation, and test Split

```
from sklearn.model_selection import train_test_split
# 70% train, 15% val, 15% test
train_df, temp_df = train_test_split(
    df,
    test_size=0.3,
    stratify=df["label"],
    random_state=42
)

val_df, test_df = train_test_split(
    temp_df,
    test_size=0.5,
    stratify=temp_df["label"],
    random_state=42
)

print("Train size:", len(train_df))
print("Validation size:", len(val_df))
print("Test size:", len(test_df))
```

```
Train size: 5208
Validation size: 1116
Test size: 1117
```

Sequence Preprocessing for ProtBERT

- I preprocessed protein sequences to match ProtBERT input requirements
- I separated each amino acid with a space
- This allows ProtBERT to tokenize amino acids individually
- The same preprocessing was applied to training, validation, and test sets

```
def space_separate(seq):
    return " ".join(list(seq))

train_df["Sequence"] = train_df["Sequence"].apply(space_separate)
val_df["Sequence"]   = val_df["Sequence"].apply(space_separate)
test_df["Sequence"]  = test_df["Sequence"].apply(space_separate)
```

```python
import torch
from transformers import AutoTokenizer, AutoModelForSequenceClassification
MODEL_NAME = "Rostlab/prot_bert"
tokenizer = AutoTokenizer.from_pretrained(
    MODEL_NAME,
    do_lower_case=False
)
model = AutoModelForSequenceClassification.from_pretrained(
    MODEL_NAME,
    num_labels=7
)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
print("on", device)
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as s
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
tokenizer_config.json: 100% 86.0/86.0 [00:00<00:00, 4.01kB/s]
config.json: 100% 361/361 [00:00<00:00, 14.2kB/s]
vocab.txt: 100% 81.0/81.0 [00:00<00:00, 2.32kB/s]
special_tokens_map.json: 100% 112/112 [00:00<00:00, 4.62kB/s]
pytorch_model.bin: 100% 1.68G/1.68G [00:27<00:00, 36.7MB/s]
model.safetensors: 20% 343M/1.68G [00:10<00:24, 54.0MB/s]
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at Rostlab/prot_bert and are newly init
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
on cuda
```

Sequence Tokenization

- I defined a custom tokenization function for protein sequences.
- Each amino acid sequence is converted into ProtBERT tokens.
- Sequences are padded to a fixed length of 512 tokens.
- Longer sequences are truncated to fit the model input size.
- This ensures uniform input format for efficient batch training.

```python
def tokenize(batch):
    return tokenizer(
        batch["Sequence"],
        padding="max_length",
        truncation=True,
        max_length=512
    )
```

Dataset Preparation

- I converted the training and validation dataframes into Hugging Face `Dataset` objects.
- The tokenization function was applied in batch mode for efficiency.
- Non-essential columns (`Sequence` and `label_name`) were removed after tokenization.
- The datasets were formatted as PyTorch tensors.
- This step prepares the data for direct use with the Trainer API.

```python
from datasets import Dataset
train_dataset = Dataset.from_pandas(train_df)
val_dataset = Dataset.from_pandas(val_df)
train_dataset = train_dataset.map(tokenize, batched=True)
val_dataset = val_dataset.map(tokenize, batched=True)
train_dataset = train_dataset.remove_columns(["Sequence", "label_name"])
val_dataset = val_dataset.remove_columns(["Sequence", "label_name"])
train_dataset.set_format("torch")
val_dataset.set_format("torch")
```

```
Map: 100%  5208/5208 [00:06<00:00, 908.12 examples/s]
Map: 100%  1116/1116 [00:01<00:00, 1056.19 examples/s]
```

Test Dataset Preparation

- I created a separate test dataset from the test split.
- The same tokenization process was applied to ensure consistency with training data.
- Original text columns were removed after tokenization.
- The test dataset was formatted as PyTorch tensors.
- This dataset is used only for final, unbiased model evaluation.

[15]
3s

```
#test dataset
test_dataset = Dataset.from_pandas(test_df)
test_dataset = test_dataset.map(tokenize, batched=True)
test_dataset = test_dataset.remove_columns(["Sequence", "label_name"])
test_dataset.set_format("torch")
```

```
Map: 100% 1117/1117 [00:03<00:00, 355.43 examples/s]
```

Training Configuration

[16]
0s

```python
from transformers import TrainingArguments

training_args = TrainingArguments(
    output_dir="./protbert_ec",
    eval_strategy="epoch",
    save_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=4,
    per_device_eval_batch_size=4,
    num_train_epochs=3,
    weight_decay=0.01,
    logging_steps=50,
    load_best_model_at_end=True,
    metric_for_best_model="accuracy",
    fp16=torch.cuda.is_available(),
    report_to="none"
)
```

Evaluation Metrics Definition

[17]
0s

```python
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
import numpy as np
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    preds = np.argmax(logits, axis=1)

    acc = accuracy_score(labels, preds)
    precision, recall, f1, _ = precision_recall_fscore_support(
        labels, preds, average="weighted"
    )
    return {
        "accuracy": acc,
        "precision": precision,
        "recall": recall,
        "f1": f1
    }
```

Here, I define the evaluation metrics used during model training and validation. I compute accuracy, precision, recall, and F1-score using a weighted average to handle class imbalance across enzyme classes.

Trainer Initialization

[18]
✓ 2s

```
from transformers import Trainer

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    tokenizer=tokenizer,
    compute_metrics=compute_metrics
)
```

```
/tmp/ipython-input-293716929.py:3: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Trainer.__init
  trainer = Trainer(
```

In this step, I initialize the Hugging Face `Trainer` by connecting the ProtBERT model, training arguments, training and validation datasets, tokenizer, and the evaluation metrics function. This configuration manages the full training and evaluation pipeline.

Training Results Analysis

[19]
44m

```
trainer.train()
```

[3906/3906 44:01, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
1	0.874300	0.712859	0.793907	0.779327	0.793907	0.771268
2	0.317900	0.581317	0.861111	0.867967	0.861111	0.858107
3	0.348100	0.494214	0.893369	0.896536	0.893369	0.893761

```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined an
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
TrainOutput(global_step=3906, training_loss=0.5777749984739258, metrics={'train_runtime': 2643.7921, 'train_samples_per_second':
5.91, 'train_steps_per_second': 1.477, 'total_flos': 1.8188221943291904e+16, 'train_loss': 0.5777749984739258, 'epoch': 3.0})
```

- I observe a consistent decrease in training and validation loss across epochs.
- Classification accuracy improves from **79.4%** in epoch 1 to **89.3%** in epoch 3.
- Precision, recall, and F1-score increase steadily, indicating better class discrimination.
- Validation performance closely follows training performance, suggesting good generalization.
- No strong signs of overfitting are observed within the 3 training epochs.

Validation Set Evaluation

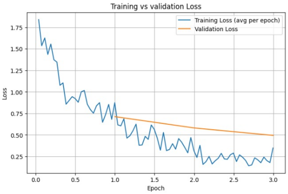
```
[20]
34s
trainer.evaluate()

[279/279 00:34]
{'eval_loss': 0.49421370029449463,
 'eval_accuracy': 0.8933691756272402,
 'eval_precision': 0.8965362131355358,
 'eval_recall': 0.8933691756272402,
 'eval_f1': 0.8937610405614278,
 'eval_runtime': 34.5459,
 'eval_samples_per_second': 32.305,
 'eval_steps_per_second': 8.076,
 'epoch': 3.0}
```

Validation Results

- I evaluated the trained model on the validation set after 3 epochs.
- The model achieves an accuracy of 89.34%.
- Precision, recall, and F1-score are all close to 0.89, indicating balanced performance.
- The low validation loss confirms stable learning and good generalization.
- These results guided the selection of the best model checkpoint.

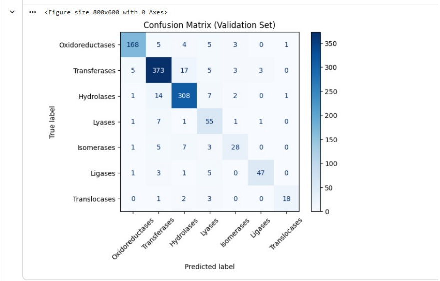
Training vs Validation Loss

```
import matplotlib.pyplot as plt
logs = trainer.state.log_history
df_logs = pd.DataFrame(logs)
train_epoch = df_logs[df_logs["loss"].notna() & df_logs["epoch"].notna()].groupby("epoch")["loss"].mean().reset_index()
eval_epoch  = df_logs[df_logs["eval_loss"].notna()][["epoch", "eval_loss"]]

plt.figure(figsize=(8,5))
plt.plot(train_epoch["epoch"], train_epoch["loss"], label="Training Loss (avg per epoch)")
plt.plot(eval_epoch["epoch"], eval_epoch["eval_loss"], label="Validation Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")

plt.title("Training vs validation Loss")
plt.legend()
plt.grid(True)
plt.show()
```

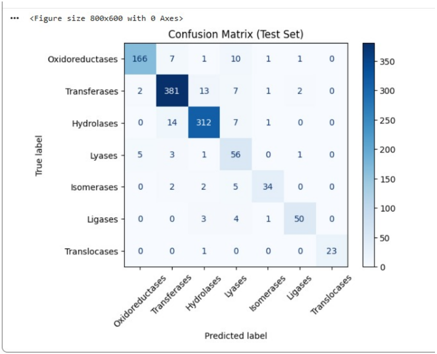


- I monitored both training and validation loss across epochs.
- Training loss decreases steadily.
- Validation loss also decreases and stabilizes over time.
- This indicates good generalization and no strong overfitting.

Confusion Matrices



Validation Set



Test Set

Both confusion matrices show strong diagonal dominance, indicating accurate classification across enzyme classes with limited confusion.

References

Uni24 The UniProt Consortium, *UniProt: the Universal Protein Knowledgebase*, Nucleic Acids Research, 2024.

EC99 Webb, E. C., *Enzyme Nomenclature*, IUBMB, 1992.

Cos24 Coste, F., *Enzymatic annotation of protein sequences with a deep language model*, MERIT CNRS Network, 2024.

Thank you for your attention