Código para execução em novos dados; Plinia cauliflora - jabuticabeira

Extração de caracteristicas

Exigências: esm2_t30_150M_UR50D, modelo ESM-2 DE 150M de parametros

In [135…

```
import esm
import torch
import pandas as pd
import numpy as np
import os


def extract_and_save_embeddings_to_csv(df_fasta, model, alphabet, device, batch_
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print("Dispositivo:", device)
    model, alphabet = esm.pretrained.esm2_t30_150M_UR50D()
    model = model.to(device)
    model.eval()
    batch_converter = alphabet.get_batch_converter()
    all_rows = []
    saved_batches = 0

    batch_sequences = []
    batch_headers = []

    for i, row in enumerate(df_fasta.itertuples(index=False)):
        header, sequence = row.header, row.sequence
        batch_headers.append(header)
        batch_sequences.append((header, sequence))

        if len(batch_sequences) == batch_size or i == len(df_fasta) - 1:
            try:
                batch_labels, batch_strs, batch_tokens = batch_converter(batch_s
                batch_tokens = batch_tokens.to(device)

                with torch.no_grad():
                    results = model(batch_tokens, repr_layers=[30], return_conta
                    token_representations = results["representations"][30]

                    for j, tokens_len in enumerate((batch_tokens != alphabet.pad
                        seq_embedding = token_representations[j, 1:tokens_len -
                        all_rows.append([batch_headers[j]] + seq_embedding.tolis

            except RuntimeError as e:
                print(f"Erro de memória em {batch_headers}: {e}")
                torch.cuda.empty_cache()
                continue

            # Libera GPU
            del batch_tokens, results, token_representations
            torch.cuda.empty_cache()
```

```
            batch_sequences = []
            batch_headers = []

            if (i + 1) % save_every == 0 or i == len(df_fasta) - 1:
                save_path = f"{output_prefix}"
                print(f"Salvando {len(all_rows)} embeddings em {save_path}...")

                df_out = pd.DataFrame(all_rows)
                df_out.to_csv(save_path, index=False, header=False)
                saved_batches += 1
                all_rows = []
    print("Extração finalizada.")

def make_fasta(fasta_path):
    headers = []
    seqs = []

    with open(fasta_path, 'r') as f:
        current_header = None
        current_seq = []

        for line in f:
            line = line.strip()
            if line.startswith(">"):
                if current_header is not None:
                    headers.append(current_header)
                    seqs.append(''.join(current_seq))
                current_header = line[1:].split()[0]
                current_seq = []
            else:
                current_seq.append(line)
        if current_header is not None:
            headers.append(current_header)
            seqs.append(''.join(current_seq))

    df_fasta = pd.DataFrame({"header": headers, "sequence": seqs})
    print(f"Total de sequências: {len(df_fasta)}")
    print(df_fasta.head())
    return df_fasta, device, model, alphabet

fasta_path = "datasets/jabuticaba/jabuticaba.fasta" # Caminho do arquivo fasta
output_feats = "datasets/jabuticaba/jabuticaba_feats.csv" # Caminho para salvar


df_fasta, device, model, alphabet = make_fasta(fasta_path)
extract_and_save_embeddings_to_csv(df_fasta, model, alphabet, device, batch_size
```

```
Total de sequências: 83
       header                                              sequence
0  A0A384TSM3  MTAILERRESESLWGRFCNWITSTENRLYIGWFGVLMIPTLLTATS...
1  A0A384SYQ7  MKTLYSLRRFYPVETLFNGTLALAGRDQETTGFAWWAGNARLINLS...
2  A0A384SYR4  MSPQTETKASVGFKAGVKDYKLNYYTPDYETKDTDILAAFRVTPQP...
3  A0A384SYS1            MTIDRTYPIFTVRWLAVHGLAVPTVSFLGSISAMQFIQR
4  A0A384SYS4  MQGRLSAWLVKHGLVHRSLGFDYQGIETLQIKPEDWHSIAVILYVY...
Dispositivo: cpu
Salvando 83 embeddings em datasets/jabuticaba/jabuticaba_feats.csv...
Extração finalizada.
```

Aplicando o modelo para predições

In [137…
```python
import pandas as pd

feats_path = output_feats # Caminho onde estão os descritores
ints_path = "datasets/jabuticaba/jabuticaba_interactions.csv" # Caminho para as

df_feats = pd.read_csv(feats_path, header=None)
df_feats = df_feats.rename(columns={0: "protein1"})
df_combined = pd.read_csv(ints_path)
df_combined = df_combined[['protein1','protein2']]
df_merged = df_combined.merge(df_feats, on="protein1", how="left")
df_feats = df_feats.rename(columns={'protein1': "protein2"})
df_merged = df_merged.merge(df_feats, on="protein2", how="left")
```

In [139…
```python
feature_cols = df_merged.columns[2:]
X = df_merged[feature_cols]
del df_merged
```

In [160…
```python
import joblib
from sklearn.metrics import accuracy_score, classification_report, confusion_mat
from sklearn.neural_network import MLPClassifier
import warnings
from sklearn.exceptions import InconsistentVersionWarning
warnings.filterwarnings("ignore", category=InconsistentVersionWarning)


scaler_path = "modelo/scaler_planta_920sc.pkl"  # Caminho do modelo de normaliza
mlp_loaded = "modelo/MLP_model_planta(512, 256, 128, 64)_e25_920sc.pkl" # Caminh

scaler = joblib.load(scaler_path)
mlp_loaded = joblib.load(mlp_loaded)
X_scaled = scaler.transform(X)


y_proba = mlp_loaded.predict_proba(X_scaled)[:, 1]
y_pred = mlp_loaded.predict(X_scaled)  # Rotulo pretivo para cada interação
```

In [ ]:
```python
```

Alinhamento e BLAST

Exigências: ter o modelo BLAST baixado (ncbi-blast-2.16.0+)

Construindo o banco de dados para alinhamento

In [ ]:
```python
import subprocess

makeblastdb_path = r"...\ncbi-blast-2.16.0+\bin\makeblastdb.exe" # Caminho até o
input_fasta_db = r"...\fasta\plantas_0_50_920sc_max1000.fasta" # Caminho até o F
cmd = [
    makeblastdb_path,
    "-in", input_fasta_db,
    "-dbtype", "prot"  # Para banco de proteínas
]
try:
    result = subprocess.run(cmd, capture_output=True, text=True, check=True)
```

```
    print("Banco de dados criado com sucesso!")
    print(result.stdout)
except subprocess.CalledProcessError as e:
    print("Erro ao criar o banco de dados:")
    print(e.stderr)
```

Alinhando as sequências

In [ ]:
```
import subprocess
import pandas as pd

blastp_path = r"C:\Users\Bruno\Downloads\ncbi-blast-2.16.0+\bin\blastp.exe" # Ca
query_fasta = "Metazoa/jabuticaba/jabuticaba.fasta" # Sequências que serão alinh
output_file = "Metazoa/jabuticaba/alinhamento_planta920_0-50sp_e_jabuticaba.txt"
db_path = input_fasta_db  # Banco de dados referência do alinhamento

cmd = [
    blastp_path,
    "-query", query_fasta,
    "-db", db_path,
    "-out", output_file,
    "-outfmt",  "6 qseqid sseqid pident ppos nident mismatch gapopen qlen slen l
    "-max_target_seqs", "5",
    "-num_threads", "16"
]

try:
    result = subprocess.run(cmd, capture_output=True, text=True, check=True)
    print("BLAST concluído com sucesso!")
    print(f"Resultados salvos em: {output_file}")
except subprocess.CalledProcessError as e:
    print("Erro ao executar o BLAST:")
    print(e.stderr)


colunas = [
    "qseqid", "sseqid", "pident", "ppos", "nident", "mismatch", "gapopen",
    "qlen", "slen", "length", "qstart", "qend", "sstart", "send", "evalue", "bit
]

df = pd.read_csv(output_file, sep="\t", names=colunas)
df.to_csv(output_file, sep="\t", index=False)
print("Arquivo salvo com cabeçalho.")
```

Visualizando o alinhamento

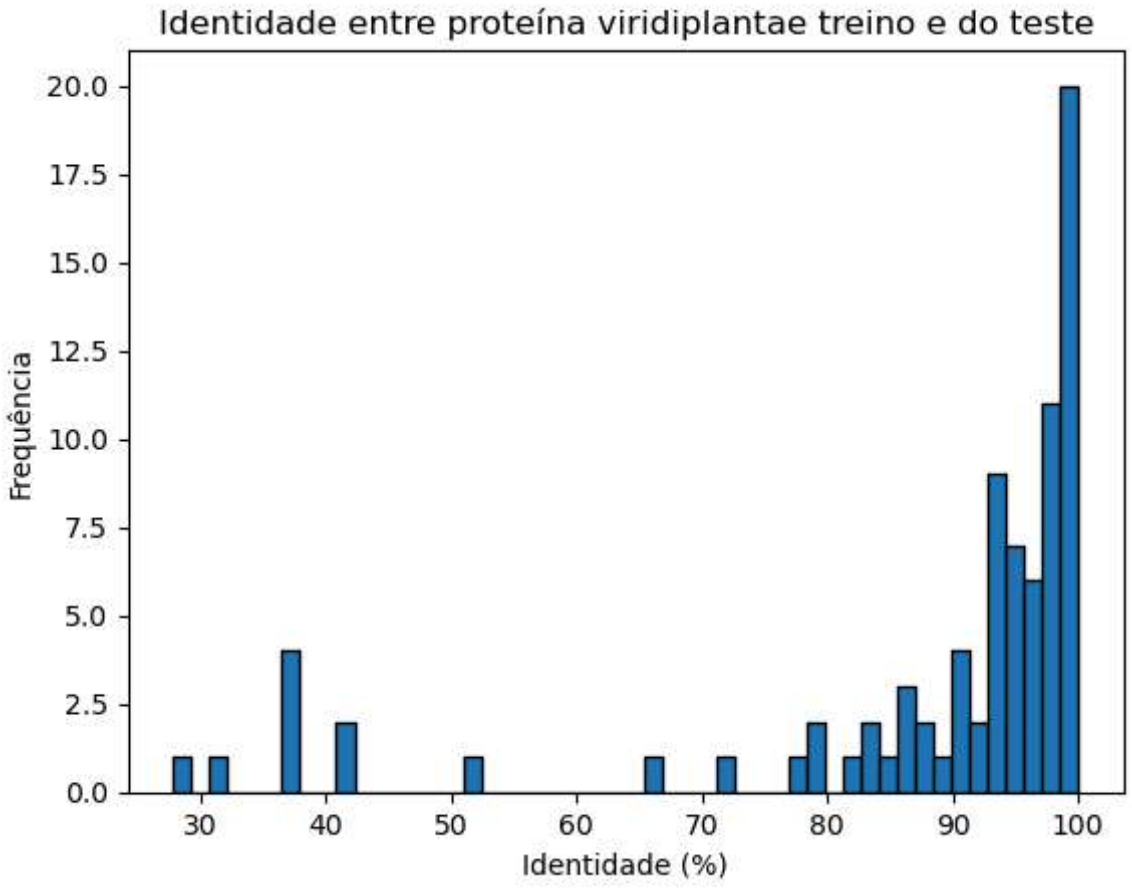
In [163…
```
import pandas as pd
import matplotlib.pyplot as plt

colunas = [
    "qseqid", "sseqid", "pident", "ppos", "nident", "mismatch", "gapopen",
    "qlen", "slen", "length", "qstart", "qend", "sstart", "send", "evalue", "bit
]
df = pd.read_csv(
    "datasets/jabuticaba/alinhamento_planta920_0-50sp_e_jabuticaba.txt",
    sep="\t",
    header=None,
```

```
    names=colunas
)


df_max = df.loc[df.groupby("qseqid")["pident"].idxmax()]
df_top5 = df.sort_values(by=["qseqid", "pident"], ascending=[True, False])  # Or
df_top5 = df_top5.groupby("qseqid").head(1)  # Pega os 5 maiores pident por qseq
df_max = df_top5.groupby("qseqid")["pident"].mean().reset_index()

plt.hist(df_max["pident"], bins=50, edgecolor='black')
plt.xlabel("Identidade (%)")
plt.ylabel("Frequência")
plt.title("Identidade entre proteína viridiplantae treino e do teste")
plt.show()
```



In [ ]:

Usando o alinhamento para filtrar as predições segundo a identidade

In [48]:
```
import pandas as pd

def filtrar_interacoes_por_dois_pidents(df_primeiro, df_combined, intervalo_max,
    try:
        inicio_max, fim_max = map(float, intervalo_max.split("-"))
        inicio_min, fim_min = map(float, intervalo_min.split("-"))
    except:
        raise ValueError("Os intervalos devem estar no formato 'inicio-fim', por

    # Conjunto de proteínas válidas
```

```
    proteinas_validas = set(df_primeiro["qseqid"])

    # Filtra só as interações onde as duas proteínas estão presentes
    df_filtrado = df_combined[
        (df_combined["protein1"].isin(proteinas_validas)) &
        (df_combined["protein2"].isin(proteinas_validas))
    ].copy()

    # Dicionário com pident de cada proteína
    pident_dict = df_primeiro.set_index("qseqid")["pident"].to_dict()

    # Mapeia o pident
    df_filtrado["pident1"] = df_filtrado["protein1"].map(pident_dict)
    df_filtrado["pident2"] = df_filtrado["protein2"].map(pident_dict)

    # Calcula máximo e mínimo
    df_filtrado["pident_max"] = df_filtrado[["pident1", "pident2"]].max(axis=1)
    df_filtrado["pident_min"] = df_filtrado[["pident1", "pident2"]].min(axis=1)

    # Aplica os dois filtros
    df_resultado = df_filtrado[
        (df_filtrado["pident_max"] > inicio_max) & (df_filtrado["pident_max"] <=
        (df_filtrado["pident_min"] > inicio_min) & (df_filtrado["pident_min"] <=
    ]
    return df_resultado
```

In [171…

```
output_csv = ints_path # Caminho até as interações
pred_out = 'datasets/jabuticaba/predicoes.csv' # Caminho para as predições alinh

prec_media = 0

# Precisão esperada para predições aleatórias (explicadas no artigo)
precisao = [
          [0.18220489600252193, ["80-100", "80-100"]],
          [0.15044098847502113, ["80-100", "40-80"]],
          [0.12058105819435874, ["80-100", "0-40"]],
          [0.12211578799528093, ["40-80", "40-80"]],
          [0.0856663911592308, ["40-80", "0-40"]],
          [0.04085226207740271, ["0-40", "0-40"]]
]

faixas = [
    ["80-100", "80-100"],
    ["80-100", "40-80"],
    ["80-100", "0-40"],
    ["40-80", "40-80"],
    ["40-80", "0-40"],
    ["0-40", "0-40"]
]

df_combined = pd.read_csv(output_csv)
df_combined['Label_predito'] = y_pred

lista_resultados = []
for i in range(len(faixas)):
    df_resultado = filtrar_interacoes_por_dois_pidents(df_max, df_combined, faix
    lista_resultados.append(df_resultado)
    prec_media += len(df_resultado[df_resultado['Label_predito'] == 1])*precisao

prec_media = prec_media/len(df_todos_resultados[df_todos_resultados['Label_predi
```

```
df_todos_resultados = pd.concat(lista_resultados, ignore_index=True)
df_todos_resultados = df_todos_resultados[['protein1', 'protein2', 'Label_predit
df_todos_resultados['precisao'] = prec_media
```

In [173…
```
df_todos_resultados = df_todos_resultados[df_todos_resultados['Label_predito'] =
df_todos_resultados.to_csv(pred_out)
```

In [175…
```
df_todos_resultados
```

Out[175…

	protein1	protein2	Label_predito	pident1	pident2	precisao
0	A0A384TSM3	A0A384SYQ7	1	99.433	99.789	0.17095
2	A0A384TSM3	A0A384SYS1	1	99.433	100.000	0.17095
3	A0A384TSM3	A0A384SYS4	1	99.433	96.835	0.17095
4	A0A384TSM3	A0A384SYW0	1	99.433	97.647	0.17095
5	A0A384TSM3	A0A384SYX1	1	99.433	92.877	0.17095
...	...	...	...	...	...	...
3363	A0A384SYV7	F1LJR5	1	79.167	37.500	0.17095
3379	A0A384T8X7	A0A384TUK1	1	41.667	37.500	0.17095
3388	A0A384T8W0	A0A384TST5	1	27.848	32.143	0.17095
3390	A0A384T8W0	A0A077HBN1	1	27.848	37.931	0.17095
3391	A0A384T8W0	A0A8K1SYL6	1	27.848	36.667	0.17095

955 rows × 6 columns