FinalProjPlink2ipynb

March 11, 2024

[ ]:
```
#!/usr/bin/env python
# coding: utf-8



import argparse
import pandas as pd
import numpy as np
import gzip
from io import StringIO
from tqdm import tqdm
import statsmodels.api as sm
from multiprocessing import Pool

# Setup argparse for command line arguments
parser = argparse.ArgumentParser(description='Run GWAS analysis')
parser.add_argument('--pheno', type=str, help='Path to phenotype file',␣
 ↪required=True)
parser.add_argument('--out', type=str, help='Output prefix for result files',␣
 ↪required=True)
parser.add_argument('--plink_results', type=str, help="Path to the Plink␣
 ↪results file", required=True)

args = parser.parse_args()




# Use the arguments

pheno_path = args.pheno
output_prefix = args.out
plink_results_path = args.plink_results

print("Arguments parsed successfully.")
```

```
# Function to drop duplicate columns, keeping the first
def drop_duplicate_columns(df):
    return df.loc[:, ~df.columns.duplicated()]


#Reading in phenotype information from ps3_gwas.phen
phenotype_df = pd.read_csv(pheno_path, sep='\t', header=None,␣
 ↪names=['SampleID', 'PhenotypeValue'])
print("Phenotype information loaded.")

# Load the intermediate DataFrames
print("Loading df1")
df1 = pd.read_csv(f"{output_prefix}_df1.csv")

print(df1.head())
print("Loading df2")
df2 = pd.read_csv(f"{output_prefix}_df2.csv")
print("Loading df3")
df3 = pd.read_csv(f"{output_prefix}_df3.csv")
print("Loading df4")
df4 = pd.read_csv(f"{output_prefix}_df4.csv")


#Merging the dataframes
print("Merging all 4 dfs")
concatenated_df = pd.concat([df1, df2], axis=1)
concatenated_df_final = pd.concat([concatenated_df, df3, df4], axis=1)
concatenated_df_final.rename(columns={'Sample ID': 'SampleID'}, inplace=True)



print("Merging the SNP data with the phenotypes")


print(phenotype_df.head())

#Dropping all but one SampleID
concatenated_df_final = concatenated_df_final.T.drop_duplicates().T
# Keep only one 'SampleID' column by identifying unique columns after␣
 ↪transposition and dropping duplicates
concatenated_df_final = concatenated_df_final.loc[:, ~concatenated_df_final.
 ↪columns.duplicated()]


print("Merging with phenotype_df")
```

```
concatenated_df_final = concatenated_df_final.merge(phenotype_df,␣
 ↪on='SampleID', how='left')

concatenated_df_final.set_index('SampleID', inplace=True)


# Identifying and dropping columns with no variation (excluding 'SampleID' and␣
 ↪'PhenotypeValue')
snp_columns = [col for col in concatenated_df_final.columns if col not in␣
 ↪['SampleID', 'PhenotypeValue']]
columns_to_drop = [col for col in snp_columns if concatenated_df_final[col].
 ↪nunique() <= 1]
newdf_cleaned = concatenated_df_final.drop(columns=columns_to_drop)


"""
Note: this function can take quite a while to run,
and can be skipped for now if final dataframes are imported
"""

#Performing linear regression
print("BEGINNING LINEAR REGRESSION")
results_summary = []

for snp in tqdm(newdf_cleaned.columns[:-1], desc="Fitting Models on SNPs"):  #␣
 ↪Adjust the slice as necessary to skip non-SNP columns
    # Ensure there's variation in SNP data
    if newdf_cleaned[snp].nunique() > 1:
        X = sm.add_constant(newdf_cleaned[snp].astype(float))  # Ensure data is␣
 ↪float
        y = newdf_cleaned['PhenotypeValue'].astype(float)  # Ensure data is␣
 ↪float

        # Check for any remaining issues with the data
        if np.any(np.isnan(X)) or np.any(np.isnan(y)):
            print(f"Skipping SNP {snp} due to NaN values.")
            continue

        # Fit the model
        model = sm.OLS(y, X, missing='drop')  # 'missing='drop'' to handle␣
 ↪missing values
        result = model.fit()

        if result.pvalues.shape[0] > 1:  # Check if SNP coefficient exists
            summary = {
                'SNP': snp,
```

```
                'p-value': result.pvalues[1],  # p-value for SNP coefficient
                'beta': result.params[1]  # Beta coefficient for SNP
            }
            results_summary.append(summary)
        else:
            print(f"Model fitting issue with SNP {snp}. Likely due to constant␣
 ↪SNP values after dropping missing data.")
    else:
        print(f"No variation in SNP {snp}. Skipping.")

results_df = pd.DataFrame(results_summary)


results_df_sorted = results_df.sort_values(by='p-value', ascending=False)


linRegResults = results_df_sorted
print("LIN REG RESULTS:", linRegResults.head())



#Saving final linear regression results to csv
#results_df_sorted.to_csv("linRegResults.csv")
# Use the output_prefix variable to create a dynamic file name
print("Saving linRegResults to CSV")
linRegResults.to_csv(f"{output_prefix}_linRegResults.csv")


# # Metrics



#Reviewing metrics of our linear regression compared to the output of Plink
print("Reading in plink results")
plinkres = pd.read_csv(plink_results_path, delim_whitespace=True)

print(plinkres.head())




#Merging our linear regression results with the plink results on "SNP"
print("Merging our linear regression results with the plink results on SNP")

linregcompare = pd.merge(results_df_sorted[['SNP', 'p-value', 'beta']],␣
 ↪plinkres[['SNP', 'P', 'BETA']], on='SNP', suffixes=('_pred', '_true'))
```

```
#Calculating differences between pred and true values
print("Calculating differences between pred and true values")
linregcompare['p_value_diff'] = abs(linregcompare['p-value'] -␣
 ↪linregcompare['P'])
linregcompare['beta_diff'] = abs(linregcompare['beta'] - linregcompare['BETA'])


print("************************RESULTS*********************************")

# Computing MAE, MSE, and RMSE

# For p-values
mae_p_value = np.mean(linregcompare['p_value_diff'])
mse_p_value = np.mean(linregcompare['p_value_diff']**2)
rmse_p_value = np.sqrt(mse_p_value)

# For beta coefficients
mae_beta = np.mean(linregcompare['beta_diff'])
mse_beta = np.mean(linregcompare['beta_diff']**2)
rmse_beta = np.sqrt(mse_beta)

# Print the results
print(f"MAE for p-values: {mae_p_value}")
print(f"MSE for p-values: {mse_p_value}")
print(f"RMSE for p-values: {rmse_p_value}")

print(f"MAE for beta coefficients: {mae_beta}")
print(f"MSE for beta coefficients: {mse_beta}")
print(f"RMSE for beta coefficients: {rmse_beta}")
```