

Supplementary Code S3 - Analysis script

Python 3.12 / numpy / scipy / pandas / statsmodels

Cellular Physiology and Biochemistry — Alnaqqash, Ayadi and Hasan

Reproduces every statistic reported in the manuscript from Supplementary Data S1 alone.

The runnable file (.py) accompanies this listing; this PDF is provided for reading and review.

Command: `python Supplementary_Code_S3_analysis.py Supplementary_Data_S1.csv > output.txt`

```
1  """
2  Supplementary Code S3
3  =====
4  Reproduces every statistic reported in the manuscript
5
6      "Diagnostic Performance of the Neutrophil-to-Lymphocyte Ratio for the
7      Identification of Insulin Resistance in Patients with Fibrotic Chronic
8      Hepatitis C: A Case-Control Study"
9
10 from the de-identified dataset (Supplementary Data S1) alone, and prints the
11 pre-specified sensitivity analysis requested by the reviewer in a form that
12 can be transcribed directly into Section 3.9, Table 9 and the Abstract.
13
14 INPUT
15 ----
16 A CSV file with one row per participant and the following columns
17 (add or rename as needed in the COLUMNS mapping below):
18
19     study_id      sequential identifier, no personal data
20     group          'control' | 'CHC-IR' | 'CHC+IR'   (clinical allocation)
21     age            years
22     sex            'M' | 'F'
23     neut           absolute neutrophil count, x109/L
24     lymph          absolute lymphocyte count, x109/L
25     alt, ast, ggt, alp, tbil, tprot
26     glucose        fasting plasma glucose, mg/dL
27     insulin        fasting insulin, uIU/mL
28     crp, il6, tnfa
29
30 NLR and HOMA-IR are always recomputed from the primary measurements. Where the
31 file also carries recorded values (columns nlr and homa_ir), the two are compared
32 and the recorded values are then used, so that the output matches the rounding
33 used in the published tables. Where it does not, the recomputed values are used.
34
35 USAGE
36 ----
37     pip install numpy scipy pandas statsmodels scikit-learn
38     python Supplementary_Code_S3_analysis.py Supplementary_Data_S1.csv > output.txt
39
40 Tested with Python 3.12, numpy 2.x, scipy 1.1x, pandas 2.x/3.x, statsmodels 0.14.
41 """
42
43 import sys
44 import warnings
45
46 import numpy as np
47 import pandas as pd
48 from scipy import stats, optimize
49 import statsmodels.api as sm
50 import statsmodels.formula.api as smf
51
52 warnings.filterwarnings('ignore') # separation makes the ML fit warn by design
53
54 IR_THRESHOLD = 2.6 # a priori HOMA-IR cut-off defining insulin resistance
55 GLUCOSE_TO_MMOL = 1 / 18.0 # mg/dL -> mmol/L
56
57 COLUMNS = dict(group='group', age='age', sex='sex', neut='neut', lymph='lymph',
58                 alt='alt', ast='ast', ggt='ggt', alp='alp', tbil='tbil',
59                 tprot='tprot', glucose='glucose', insulin='insulin',
60                 crp='crp', il6='il6', tnfa='tnfa',
61                 # optional: the values as recorded in the study database. When
62                 # present these are used for the analysis and cross-checked
63                 # against the values recomputed from the primary measurements,
64                 # so that the output matches the rounding used in the article.
```

```

65         nlr='nlr', homa_ir='homa_ir')
66
67
68 # -----
69 # helpers
70 # -----
71
72 def wilson(k, n, alpha=0.05):
73     """Wilson score interval for a binomial proportion, as percentages."""
74     if n == 0:
75         return (np.nan, np.nan)
76     z = stats.norm.ppf(1 - alpha / 2)
77     p = k / n
78     d = 1 + z ** 2 / n
79     c = p + z ** 2 / (2 * n)
80     h = z * np.sqrt(p * (1 - p) / n + z ** 2 / (4 * n ** 2))
81     return (100 * (c - h) / d, 100 * (c + h) / d)
82
83
84 def auc_delong(y, x):
85     """
86     Non-parametric AUC with the DeLong variance.
87
88     Returns (auc, variance). The variance is exactly zero when the two
89     classes are completely separated, in which case no confidence interval
90     can be constructed and the caller should report 'not estimable'.
91     """
92     y = np.asarray(y, dtype=int)
93     x = np.asarray(x, dtype=float)
94     pos, neg = x[y == 1], x[y == 0]
95     m, n = len(pos), len(neg)
96
97     def kernel(a, b):
98         return np.where(a > b, 1.0, np.where(a == b, 0.5, 0.0))
99
100    mat = kernel(pos[:, None], neg[None, :])
101    auc = mat.mean()
102    v10 = mat.mean(axis=1) # structural components, positives
103    v01 = mat.mean(axis=0) # structural components, negatives
104    s10 = v10.var(ddof=1) if m > 1 else 0.0
105    s01 = v01.var(ddof=1) if n > 1 else 0.0
106    var = s10 / m + s01 / n
107    return auc, var
108
109
110 def auc_ci(auc, var, alpha=0.05):
111     """Logit-transformed DeLong interval; None when the variance is zero."""
112     if var <= 0 or auc in (0.0, 1.0):
113         return None
114     z = stats.norm.ppf(1 - alpha / 2)
115     lo_logit = np.log(auc / (1 - auc))
116     se_logit = np.sqrt(var) / (auc * (1 - auc))
117     lo, hi = lo_logit - z * se_logit, lo_logit + z * se_logit
118     return (1 / (1 + np.exp(-lo)), 1 / (1 + np.exp(-hi)))
119
120
121 def youden(y, x):
122     """
123     Threshold maximising the Youden index, using '>= threshold' as positive.
124     Candidate thresholds are the observed values, so the returned threshold is
125     always a value that actually occurs in the data.
126     """
127     y = np.asarray(y, dtype=int)
128     x = np.asarray(x, dtype=float)
129     best = None
130     for t in np.unique(x):
131         pred = x >= t
132         tp = int(np.sum(pred & (y == 1)))
133         fn = int(np.sum(~pred & (y == 1)))
134         tn = int(np.sum(~pred & (y == 0)))
135         fp = int(np.sum(pred & (y == 0)))
136         sens = tp / (tp + fn) if tp + fn else 0.0
137         spec = tn / (tn + fp) if tn + fp else 0.0
138         j = sens + spec - 1
139         if best is None or j > best[1]:

```

```

140     best = (float(t), j, tp, fn, tn, fp)
141     t, j, tp, fn, tn, fp = best
142     n_pos, n_neg, n = tp + fn, tn + fp, tp + fn + tn + fp
143     return dict(threshold=t, J=j, tp=tp, fn=fn, tn=tn, fp=fp,
144                 sens=100 * tp / n_pos, sens_ci=wilson(tp, n_pos),
145                 spec=100 * tn / n_neg, spec_ci=wilson(tn, n_neg),
146                 acc=100 * (tp + tn) / n, acc_ci=wilson(tp + tn, n))
147
148
149 def separated(y, x):
150     """True when the two classes do not overlap on x."""
151     y = np.asarray(y, dtype=int)
152     x = np.asarray(x, dtype=float)
153     return float(x[y == 0].max()) < float(x[y == 1].min())
154
155
156 def penalised_loglik(X, y, beta):
157     """Firth-penalised log-likelihood: log L(beta) + 0.5 * log|I(beta)|."""
158     eta = X @ beta
159     ll = np.sum(y * eta - np.logaddexp(0.0, eta))
160     p = 1 / (1 + np.exp(-eta))
161     I = X.T @ (X * (p * (1 - p))[:, None])
162     sign, logdet = np.linalg.slogdet(I)
163     return ll + 0.5 * logdet if sign > 0 else -np.inf
164
165
166 def _firth_max(X, y, start, fix=None):
167     """Maximise the penalised log-likelihood, optionally holding one coefficient fixed."""
168     k = X.shape[1]
169     if fix is None:
170         obj = lambda b: -penalised_loglik(X, y, b)
171         r = optimize.minimize(obj, start, method='Nelder-Mead',
172                             options=dict(maxiter=60000, maxfev=60000,
173                                           fatol=1e-12, xatol=1e-10))
174         return r.x, -r.fun
175     idx, val = fix
176     free = [j for j in range(k) if j != idx]
177
178     def obj(bf):
179         b = np.zeros(k)
180         b[idx] = val
181         b[free] = bf
182         return -penalised_loglik(X, y, b)
183
184     r = optimize.minimize(obj, np.delete(start, idx), method='Nelder-Mead',
185                         options=dict(maxiter=60000, maxfev=60000,
186                                       fatol=1e-12, xatol=1e-10))
187     return r.x, -r.fun
188
189
190 def firth_profile_ci(X, y, idx, alpha=0.05):
191     """
192     Profile penalised-likelihood confidence interval and penalised likelihood
193     ratio test for one coefficient.
194
195     Under complete or quasi-complete separation the profile is almost flat as
196     the coefficient increases, so the upper limit is legitimately very large.
197     That is a property of the data, not a failure of the routine: report the
198     lower limit as the informative bound and do not read the upper limit as a
199     calibrated effect size.
200     """
201     Xc = X.astype(float).copy()
202     Xc[:, 1:] = Xc[:, 1:] - Xc[:, 1:].mean(axis=0) # centre for stability
203     start, _ = firth_logit(Xc, y)
204     beta, ll0 = _firth_max(Xc, y, start)
205     crit = stats.chi2.ppf(1 - alpha, 1) / 2
206     g = lambda v: ll0 - _firth_max(Xc, y, beta, fix=(idx, v))[1] - crit
207     bounds = []
208     for direction in (-1, 1):
209         step, edge = 0.3, beta[idx]
210         for _ in range(60):
211             edge += direction * step
212             if g(edge) > 0:
213                 break
214             step *= 1.25

```

```

215     bounds.append(optimize.brentq(g, min(beta[idx], edge),
216                                   max(beta[idx], edge), xtol=1e-6))
217     lrt = 2 * (ll0 - _firth_max(Xc, y, beta, fix=(idx, 0.0))[1])
218     return beta[idx], min(bounds), max(bounds), stats.chi2.sf(lrt, 1)
219
220
221 def firth_logit(X, y, max_iter=200, tol=1e-8):
222     """
223     Firth penalised-likelihood logistic regression (Newton-Raphson on the
224     penalised score). X must already include the intercept column.
225     Returns (beta, se). Use profile penalised likelihood for publication
226     intervals; the Wald se returned here is adequate for a cross-check.
227     """
228     X = np.asarray(X, dtype=float)
229     y = np.asarray(y, dtype=float)
230     beta = np.zeros(X.shape[1])
231     for _ in range(max_iter):
232         eta = X @ beta
233         p = 1 / (1 + np.exp(-eta))
234         W = p * (1 - p)
235         XW = X * W[:, None]
236         I = X.T @ XW
237         Iinv = np.linalg.pinv(I)
238         H = XW @ Iinv @ X.T
239         h = np.diag(H)
240         U = X.T @ (y - p + h * (0.5 - p))
241         step = Iinv @ U
242         beta_new = beta + step
243         if np.max(np.abs(step)) < tol:
244             beta = beta_new
245             break
246         beta = beta_new
247     eta = X @ beta
248     p = 1 / (1 + np.exp(-eta))
249     I = X.T @ (X * (p * (1 - p))[:, None])
250     se = np.sqrt(np.diag(np.linalg.pinv(I)))
251     return beta, se
252
253
254 # -----
255 # reporting blocks
256 # -----
257
258 def roc_block(label, y, x):
259     auc, var = auc_delong(y, x)
260     ci = auc_ci(auc, var)
261     yd = youden(y, x)
262     sep = separated(y, x)
263     print(f" {label}")
264     print(f"    n = {len(y)}    (positive {int(np.sum(y))}, negative {int(len(y) - np.sum(y))})")
265     print(f"    AUC                = {auc:.4f}")
266     print(f"    DeLong variance = {var:.3e}")
267     print(f"    f"{' -> zero: complete separation, CI not estimable' if var <= 0 else ''}")
268     print(f"    95 % CI          = " + ("not estimable" if ci is None else f"{ci[0]:.3f}-{ci[1]:.3f}")")
269     print(f"    complete separation of the two classes: {'YES' if sep else 'NO'}")
270     print(f"    Youden-optimal threshold = {yd['threshold']:.4g}    (J = {yd['J']:.4f})")
271     print(f"    sensitivity = {yd['sens']:.1f} % ({yd['sens_ci'][0]:.1f}-{yd['sens_ci'][1]:.1f})")
272     print(f"    f"[{yd['tp']} of {yd['tp'] + yd['fn']}]")
273     print(f"    specificity = {yd['spec']:.1f} % ({yd['spec_ci'][0]:.1f}-{yd['spec_ci'][1]:.1f})")
274     print(f"    f"[{yd['tn']} of {yd['tn'] + yd['fp']}]")
275     print(f"    accuracy      = {yd['acc']:.1f} % ({yd['acc_ci'][0]:.1f}-{yd['acc_ci'][1]:.1f})")
276     print(f"    class ranges: negative max = {np.asarray(x)[np.asarray(y) == 0].max():.4g}, "
277           f"positive min = {np.asarray(x)[np.asarray(y) == 1].min():.4g}")
278     return dict(auc=auc, var=var, ci=ci, youden=yd, separated=sep)
279
280
281 def regression_block(df, outcome='homa_ir'):
282     """
283     Linear regression of HOMA-IR on NLR adjusted for age and sex.
284
285     Note: this model is by construction identical in the primary and the
286     sensitivity analysis. HOMA-IR enters as a continuous outcome, the
287     covariates are unchanged, and reclassification only relabels patients
288     without adding or removing any. It is reported in both blocks for
289     completeness and as a check that the datasets are otherwise identical.

```

```

290 """
291 m = smf.ols(f'{outcome} ~ nlr + age + C(sex)', data=df).fit()
292 b = m.params['nlr']
293 lo, hi = m.conf_int().loc['nlr']
294 print(f"      linear regression, B for NLR = {b:.3f} (95 % CI {lo:.3f} to {hi:.3f}), "
295       f"p = {m.pvalues['nlr']:.3g}, R2 = {m.rsquared:.3f}")
296 return m
297
298
299 def logistic_block(df):
300     """Conventional ML logistic regression, then Firth with profile PL interval."""
301     X = sm.add_constant(pd.DataFrame({
302         'nlr10': df['nlr'] * 10,          # per 0.1 unit of NLR
303         'age': df['age'],
304         'male': (df['sex'].astype(str).str.upper().str[0] == 'M').astype(float)
305     }))
306     y = df['ir'].astype(float).to_numpy()
307
308     try:
309         ml = sm.Logit(y, X).fit(disp=0, maxiter=500)
310         finite = np.all(np.isfinite(ml.params)) and np.max(np.abs(ml.params)) < 50
311         converged = bool(ml.mle_retvals.get('converged', False)) and finite
312         print(f"      maximum-likelihood logistic regression: "
313               f"{'converged to finite estimates' if converged else 'did NOT converge to finite estimates'}"
314               f"      (max |beta| = {np.max(np.abs(ml.params)):.1f})")
315     except Exception as e:
316         print(f"      maximum-likelihood logistic regression: failed ({type(e).__name__})")
317
318     i = list(X.columns).index('nlr10')
319     b, lo, hi, p = firth_profile_ci(X.to_numpy(), y, i)
320     print(f"      Firth penalised logistic regression, OR per 0.1 NLR = {np.exp(b):.2f} "
321           f"      (95 % profile penalised-likelihood CI {np.exp(lo):.2f} to {np.exp(hi):.0f}), "
322           f"      penalised LRT p = {p:.3g}")
323     if np.exp(hi) > 100:
324         print("      the upper limit is very wide because the data are separated or "
325               "      nearly so; only the lower limit is informative")
326
327
328 # -----
329 # main
330 # -----
331
332 def main(path):
333     if str(path).lower().endswith(('.xlsx', '.xlsm')):
334         probe = pd.read_excel(path, header=None, nrows=30)
335         hdr = 0
336         for i in range(len(probe)):
337             if probe.iloc[i].astype(str).str.contains('Subject_ID|study_id|group',
338                                                         case=False, na=False).any():
339                 hdr = i
340                 break
341         raw = pd.read_excel(path, header=hdr)
342         raw = raw.loc[:, ~raw.columns.astype(str).str.startswith('Unnamed')]
343         raw = raw.dropna(how='all').dropna(axis=1, how='all')
344     else:
345         raw = pd.read_csv(path)
346     df = raw.rename(columns={v: k for k, v in COLUMNS.items()}).copy()
347
348     # Derived variables. They are always recomputed from the primary
349     # measurements; where the study database also carries recorded values the
350     # two are compared, and the recorded values are then used so that the
351     # output matches the rounding used in the published tables.
352     nlr_calc = df['neut'] / df['lymph']
353     homa_calc = df['insulin'] * (df['glucose'] * GLUCOSE_TO_MMOL) / 22.5
354     for name, calc in (('nlr', nlr_calc), ('homa_ir', homa_calc)):
355         if name in df.columns and df[name].notna().all():
356             diff = (df[name] - calc).abs().max()
357             print(f"recorded vs recomputed {name}: max absolute difference = {diff:.4f}"
358                   f"      -> using recorded values")
359         else:
360             df[name] = calc
361             print(f"{name}: not recorded in the file; recomputed from primary measurements")
362
363     print("=" * 78)
364     print("SUPPLEMENTARY CODE S3 - reproduction of reported analyses")

```

```

365 print("=" * 78)
366 print(f"file          : {path}")
367 print(f"participants    : {len(df)}")
368 print(f"missing values    : {int(df.isna().sum().sum())}")
369 print()
370 print(f"groups            : {df['group'].value_counts().to_dict()}")
371
372 # ----- classification check -----
373 print("\n" + "-" * 78)
374 print("1. CLASSIFICATION OF INSULIN RESISTANCE")
375 print("-" * 78)
376 df['ir_clinical'] = (df['group'] == 'CHC+IR').astype(int)
377 df['ir_strict'] = (df['homa_ir'] >= IR_THRESHOLD).astype(int)
378 chc = df[df['group'].isin(['CHC-IR', 'CHC+IR'])].copy()
379
380 disagree = df[df['ir_clinical'] != df['ir_strict']]
381 agree = len(df) - len(disagree)
382 print(f"a priori threshold: HOMA-IR >= {IR_THRESHOLD}")
383 print(f"clinical allocation reproduced in {agree} of {len(df)} "
384       f"({100 * agree / len(df):.1f} %)")
385 if len(disagree):
386     print("\ndiscordant participants (these drive the sensitivity analysis):")
387     cols = [c for c in ['study_id', 'group', 'insulin', 'glucose', 'homa_ir', 'nlr'] if c in disagree.columns]
388     print(disagree[cols].to_string(index=False,
389                                     float_format=lambda v: f"{v:.4f}"))
390 else:
391     print("\nno discordant participants; the sensitivity analysis is identical "
392           "to the primary analysis")
393
394 # ----- primary analysis -----
395 print("\n" + "-" * 78)
396 print("2. PRIMARY ANALYSIS (clinical allocation retained)")
397 print("-" * 78)
398 chc_p = chc.copy()
399 chc_p['ir'] = chc_p['ir_clinical']
400 for lbl, sub in (('insulin-resistant', chc_p[chc_p.ir == 1]),
401                 ('non-insulin-resistant', chc_p[chc_p.ir == 0])):
402     print(f" NLR, {lbl:22s} n = {len(sub):3d} "
403           f"mean +/- SD = {sub['nlr'].mean():.2f} +/- {sub['nlr'].std(ddof=1):.2f} "
404           f"range {sub['nlr'].min():.2f}-{sub['nlr'].max():.2f}")
405 t, p = stats.ttest_ind(chc_p[chc_p.ir == 1]['nlr'], chc_p[chc_p.ir == 0]['nlr'],
406                       equal_var=False)
407 print(f" Welch t = {t:.2f}, p = {p:.3g} (Welch F = {t ** 2:.1f})")
408 print()
409 roc_block("ROC: NLR for insulin resistance within CHC", chc_p['ir'], chc_p['nlr'])
410 print()
411 regression_block(chc_p)
412 logistic_block(chc_p)
413
414 # ----- sensitivity analysis -----
415 print("\n" + "-" * 78)
416 print(f"3. SENSITIVITY ANALYSIS (HOMA-IR >= {IR_THRESHOLD} applied strictly)")
417 print(" -> transcribe these values into Section 3.9, Table 9 and the Abstract")
418 print("-" * 78)
419 chc_s = chc.copy()
420 chc_s['ir'] = chc_s['ir_strict']
421 print(f" group sizes after reclassification: "
422       f"insulin-resistant n = {int(chc_s['ir'].sum())}, "
423       f"non-insulin-resistant n = {int((1 - chc_s['ir']).sum())}")
424 for lbl, sub in (('insulin-resistant', chc_s[chc_s.ir == 1]),
425                 ('non-insulin-resistant', chc_s[chc_s.ir == 0])):
426     print(f" NLR, {lbl:22s} n = {len(sub):3d} "
427           f"mean +/- SD = {sub['nlr'].mean():.2f} +/- {sub['nlr'].std(ddof=1):.2f} "
428           f"range {sub['nlr'].min():.2f}-{sub['nlr'].max():.2f}")
429 t, p = stats.ttest_ind(chc_s[chc_s.ir == 1]['nlr'], chc_s[chc_s.ir == 0]['nlr'],
430                       equal_var=False)
431 print(f" Welch t = {t:.2f}, p = {p:.3g} (Welch F = {t ** 2:.1f})")
432 print()
433 roc_block("ROC: NLR for insulin resistance within CHC (strict)", chc_s['ir'], chc_s['nlr'])
434 print()
435 regression_block(chc_s)
436 logistic_block(chc_s)
437
438 print("\n" + "-" * 78)

```

```
439 print("4. COMPARATOR MARKERS, PRIMARY ANALYSIS")
440 print("-" * 78)
441 for var, name in [('crp', 'CRP (mg/L)'), ('tnfa', 'TNF-alpha (pg/mL)'),
442                  ('il6', 'IL-6 (pg/mL)'), ('ggt', 'GGT (U/L)'),
443                  ('alt', 'ALT (U/L)'), ('neut', 'Neutrophils (x10^9/L)')]:
444     if var in chc_p.columns:
445         roc_block(name, chc_p['ir'], chc_p[var])
446         print()
447
448 print("=" * 78)
449 print("END")
450 print("=" * 78)
451
452
453 if __name__ == '__main__':
454     if len(sys.argv) != 2:
455         sys.exit(__doc__)
456     main(sys.argv[1])
```