proj03.py

April 26, 2024

[ ]:
```python
import pandas as pd
import numpy as np
import statsmodels.formula.api as smf
from scipy import stats

def cgmwildboot(data, model, n_bootstraps, cluster, bootcluster, seed=1234):
        """
        This function performs wild bootstrap inference for clustered data as␣
 ↪proposed by Cameron, Gelbach, and Miller (2008).

        Args:
        data: pandas DataFrame
        model: statsmodels regression model
        n_bootstraps: int, number of bootstrap samples
        cluster: list, name of the cluster variable to use in bootstrap␣
 ↪regressions
        bootcluster: list, name of the cluster variable to use in bootstrap␣
 ↪sampling
        seed: int, random seed

        Returns:
        b_ests: numpy array, bootstrapped parameter estimates
        b_pvals: numpy array, bootstrapped p-values
        """

        np.random.seed(seed)
        df = data.copy(deep=True)

        # gather dependent variable and independent variables from model.model.
 ↪formula
        dep = model.model.endog_names
        indep = model.model.exog_names[1:]
        b_ests = []
        b_pvals = []
        b_bse = []
        df['yhat'] = model.predict(df[indep])
        df['ehat'] = model.resid
```

```
    for i in range(n_bootstraps):
        # generate rademacher weights for each cluster
        signs = df[bootcluster].drop_duplicates().apply(lambda x: np.
↪random.choice([-1, 1]))
        signs.index = df[bootcluster].drop_duplicates()
        df['sign'] = df[bootcluster].map(signs)
        # apply weights to residuals and add to predicted values
        df['we'] = df['ehat'] * df['sign']
        df['wy'] = df['yhat'] + df['we']
        df[dep] = df['wy']

        boot_model = smf.ols(model.model.formula, data=df).
↪fit(cov_type='cluster', cov_kwds={'groups': df[cluster]})
        b_ests.append(boot_model.params)
        b_bse.append(boot_model.bse)

    # remove constant
    length = len(indep) + 1
    b_ests = np.array(b_ests)[:,1:length]
    b_bse = np.array(b_bse)[:,1:length]

    for i, var in enumerate(indep):

        # calculate the wald statistic for each variable
        w_boot = (b_ests[:,i]-model.params[var]) / b_bse[:,i]

        # here for simplicity we assume H0: beta = 0 as we do this for␣
↪all variables, but should be adjusted if we want to generalize the function
        w = (model.params[var]-0) / model.bse[var]

        # calculate the p-value for the wald statistic
        pval = np.mean(np.abs(w_boot) > np.abs(w))
        b_pvals.append(pval)

    return b_ests, b_pvals
```