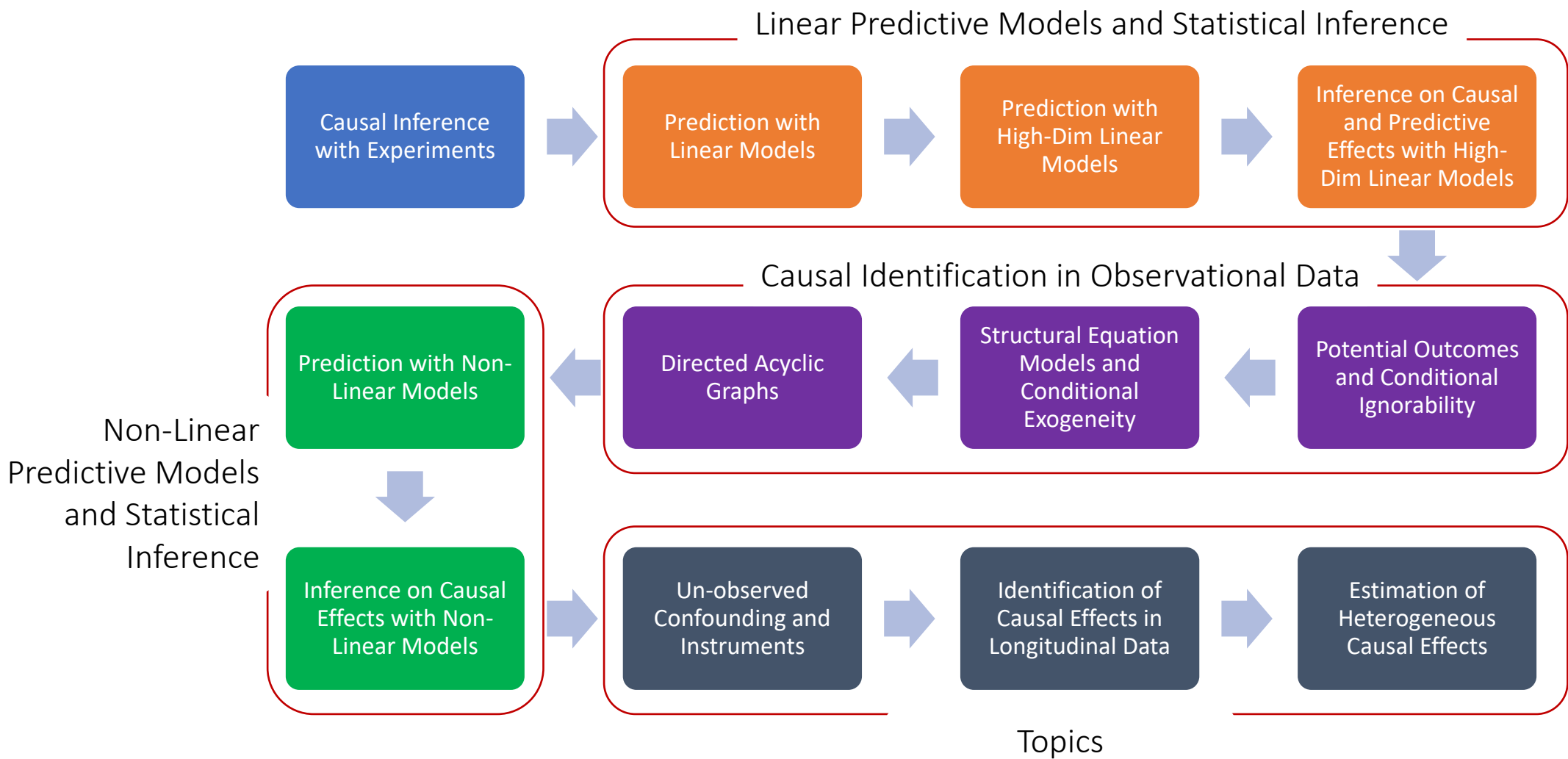
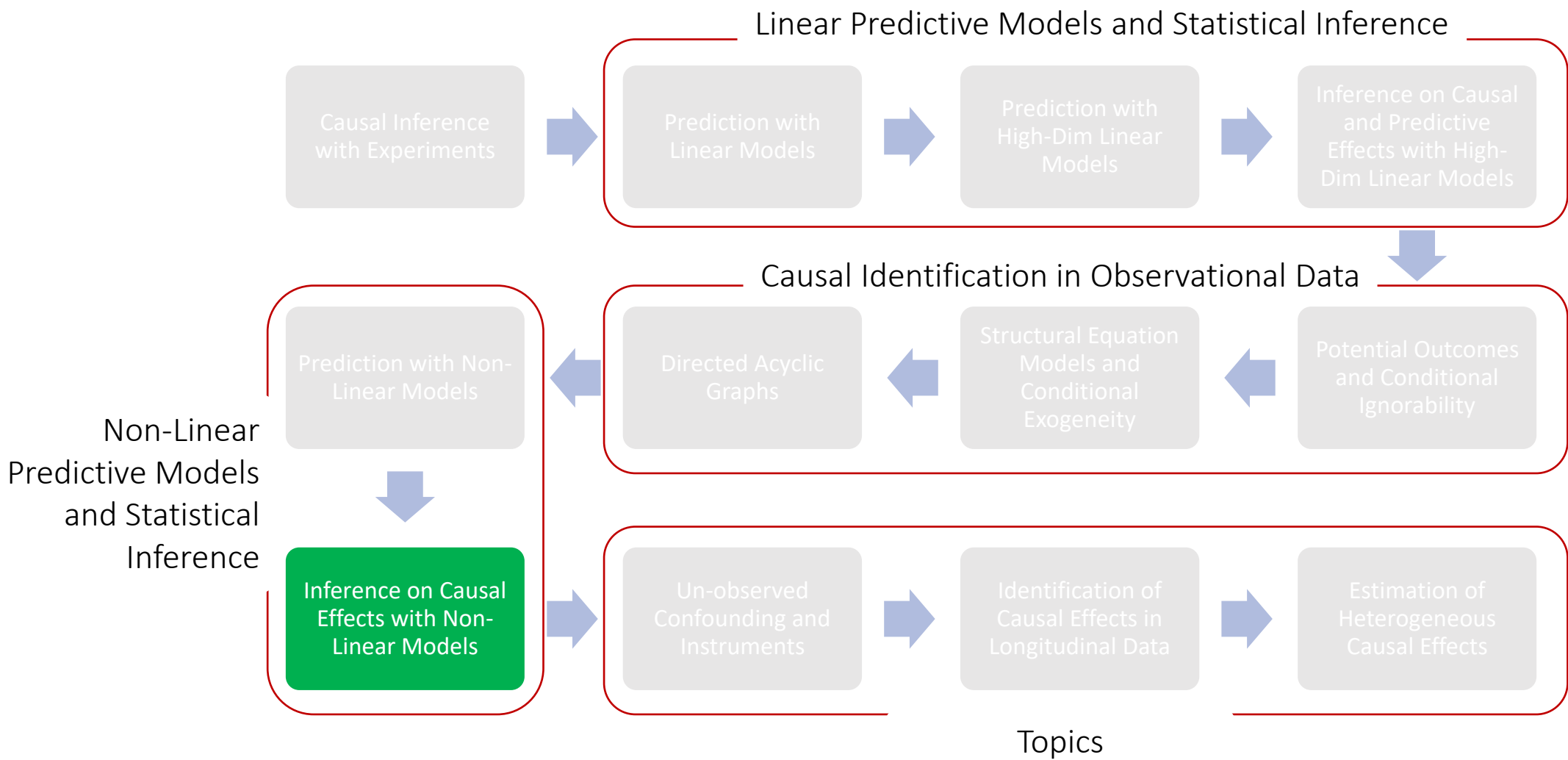


MS&E 228: Inference with Modern Non-Linear Prediction

Vasilis Syrgkanis

MS&E, Stanford





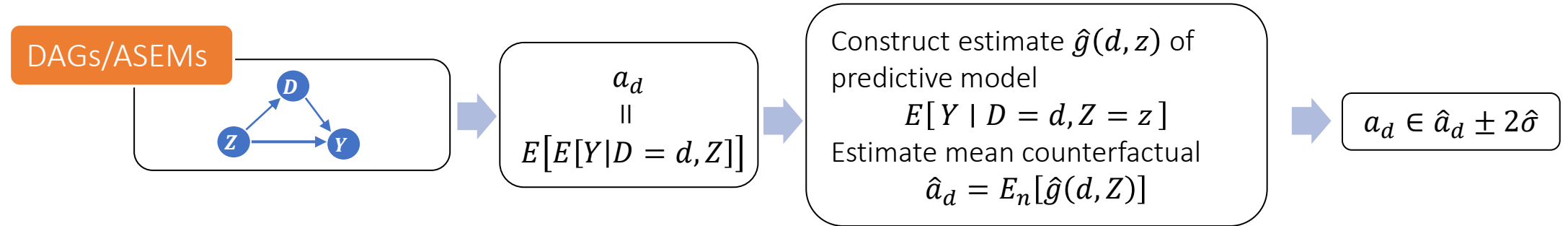
Goals for Today

- Revisiting and expanding on theory from last time
- Sample-splitting and cross-fitting
- Example application

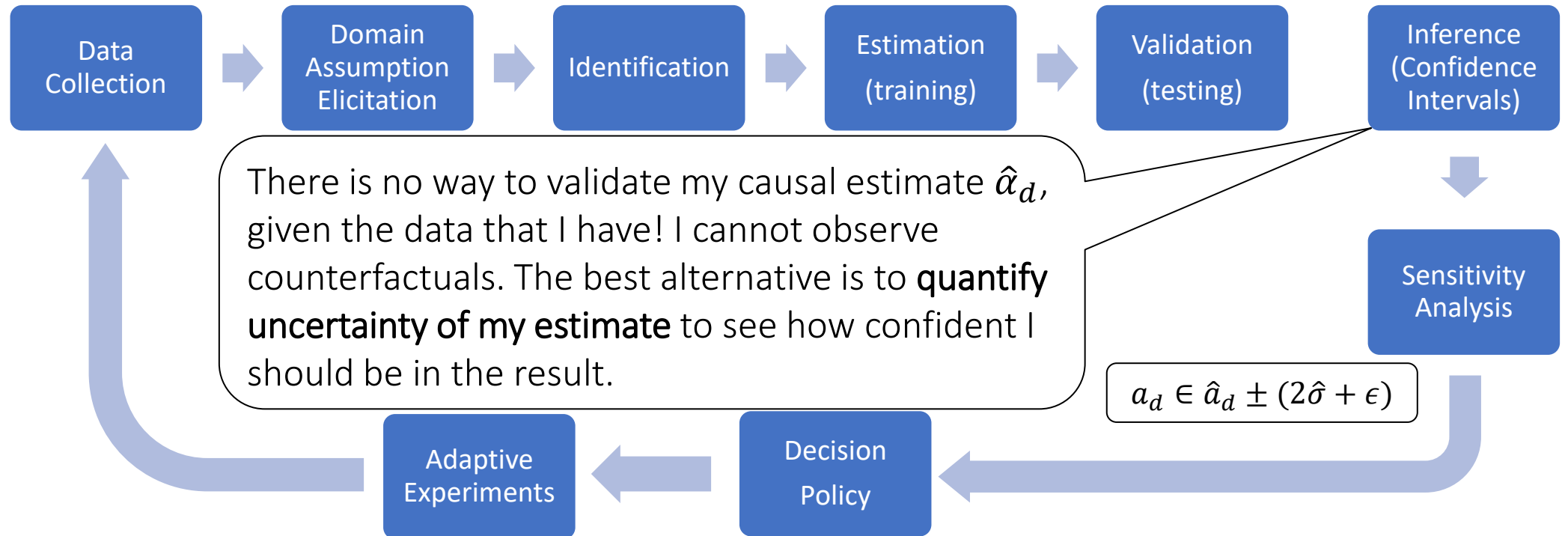
Recap of Last Lecture

Causal Inference Pipeline

Theory



Practice



Semi-Parametric Moment Restrictions

- Observe samples Z_1, \dots, Z_n i.i.d. from data distribution D
- Distribution D satisfies vector of moment restrictions
$$M(\theta_0, g_0) := E_{Z \sim D}[m(Z; \theta_0, g_0)] = 0$$
- $\theta_0 \in \mathbb{R}^d$ finite dimensional target parameter of interest
- $g_0 \in G$ potentially infinite dimensional parameter we don't care (nuisance)
- g_0 is un-known and needs to be estimated from data

Natural Estimation Algorithm (sample-splitting)

Split the data in half (S_1, S_2)

- On first half S_1 , estimate \hat{g} of g_0
- On second half S_2 , $\hat{\theta}$ is solution w.r.t. θ of empirical plug-in moment equation:

$$M_n(\theta, \hat{g}) := \frac{1}{n_2} \sum_{i \in S_2} m(Z_i; \theta, \hat{g}) = 0$$

Natural Estimation Algorithm (cross-fitting)

- Split data in K parts, S_1, \dots, S_K
- For each part k , estimate \hat{g}_k using data from all parts except S_k
- $\hat{\theta}$ is solution w.r.t. θ of cross-fitted empirical moment equation:

$$\frac{1}{n} \sum_{k=1}^K \sum_{i \in S_k} m(Z_i; \hat{\theta}, \hat{g}_k) = 0$$

Neyman Orthogonality

Moment $M(\theta, g)$ is **Neyman orthogonal** if for any $v \in G - g_0$:

$$D_g M(\theta_0, g_0)[v] := \left. \frac{\partial}{\partial t} M(\theta_0, g_0 + t v) \right|_{t=0} = 0$$

Main Theorem

If moment is Neyman orthogonal and RMSE of \hat{g} is $o_p(n^{-1/4})$, plus regularity conditions

$$\sqrt{n} (\hat{\theta} - \theta_0) \rightarrow N \left(0, J_0^{-1} \Sigma (J_0^{-1})^\top \right)$$

where $J_0 := \nabla_{\theta} M(\theta_0, g_0)$ and $\Sigma := E[m(Z; \theta_0, g_0) m(Z; \theta_0, g_0)^\top]$

Two Main Applications

- Average Treatment Effect in identification by conditioning and a binary treatment
- Doubly robust moment which combines identification by regression and identification via propensity scores
- Average Treatment Effect in identification by conditioning under a partially linear model; response linear in treatment
- Residual-on-residual moment which generalizes the partialling out approach (FWL) to non-linear models

Expanding More on Theory

Main Theorem (expanded) Define RMSE: $\|h\|_{L^2} = \sqrt{E[h(X)^2]}$

- If moment is **Neyman orthogonal** and RMSE of \hat{g} goes down at rate $n^{1/4}$, plus regularity conditions

$$n^{1/4} \|\hat{g} - g_0\|_{L^2} \approx 0$$

Jacobian of moments with respect to parameter; relates to identification strength

- Then the estimate $\hat{\theta}$ is **asymptotically linear**

$$\sqrt{n}(\hat{\theta} - \theta_0) \approx \sqrt{n} E_n[\phi_0(Z)], \quad \phi_0(Z) = -J_0^{-1} m(Z; \theta_0, g_0), \quad J_0 := \partial_{\theta} E[m(Z; \theta_0, g_0)]$$

influence function

influence is a linear transformation of the moment; transforming from

- Consequently, it is **asymptotically normal**

$$\sqrt{n}(\hat{\theta} - \theta_0) \underset{d}{\sim} N(0, V), \quad V = E[\phi_0(Z)\phi_0(Z)']$$

Covariance of the influence function

- Confidence intervals** for any projection based on estimate of variance are asymptotically valid

$$\ell' \theta \in \left[\ell' \hat{\theta} \pm c \sqrt{\frac{\ell' \hat{V} \ell}{n}} \right],$$

$$\hat{V} = \text{Var}_n(\hat{\phi}(Z)),$$

Empirical variance of approximate influence

$$\hat{\phi}(Z) := -\hat{J}^{-1} m(Z; \hat{\theta}, \hat{g}),$$

Approximate influence function

$$\hat{J} = \partial_{\theta} E_n[m(Z; \hat{\theta}, \hat{g})]$$

Empirical average of jacobian

Python Pseudocode

```
# General DML pseudocode
def general_dml(Z, ell, nfolds, moment, jacobian, nuisance_estimator):
    # construct out-of-fold predictions from the nuisance estimator
    ghat = cross_val_predict(nuisance_estimator, Z, cv=nfolds)
    # use these predictions to define the empirical moment function
    # with respect to theta
    avg_moment = lambda theta: np.mean(moment(Z, theta, ghat), axis=0)
    # solve for the empirical moment equation equals zero
    thetahat = fsolve(avg_moment)
    # calculate empirical jacobian with respect to theta, evaluated
    # at the estimates thetahat and ghat
    Jhat = np.mean(jacobian(Z, thetahat, ghat), axis=0)
    # construct approximate influence function for each sample
    phihat = - moment(Z, thetahat, ghat) @ np.linalg.pinv(Jhat).T
    # variance estimate
    var = phihat.T @ phihat / phihat.shape[0]
    # estimate and standard error for any projection ell of the parameter
    point = ell @ thetahat
    stderr_ell = np.sqrt(ell.T @ var @ ell / phihat.shape[0])
    return point, stderr_ell
```

Solve that
moment = 0
wrt θ

Estimate \hat{g} out of fold and predict

Construct cross-fitted moment function
 $\theta \rightarrow E_n[m(Z; \theta, \hat{g})]$

Calculate
 $\hat{J} := \partial_{\theta} E_n[m(Z; \hat{\theta}, \hat{g})]$

Approximate influence
function
 $Z \rightarrow \hat{\phi}(Z) := \hat{J}^{-1} m(Z; \hat{\theta}, \hat{g})$

Empirical covariance of
estimate
 $\hat{V} = E_n[\hat{\phi}(Z) \hat{\phi}(Z)']$

Main Theorem (linear moments)

- If moments are linear

$$m(Z; \theta, g) = v(Z; g) - a(Z; g)\theta$$

- Estimate is closed form:

$$\hat{\theta} = \hat{J}^{-1} E_n[v(Z; g)], \quad \hat{J} = E_n[a(Z; g)]$$

- Then the estimate $\hat{\theta}$ is *asymptotically linear*

$$\sqrt{n}(\hat{\theta} - \theta_0) \approx \sqrt{n} E_n[\phi_0(Z)], \quad \phi_0(Z) = -J_0^{-1} m(Z; \theta_0, g_0), \quad J_0 := E[a(Z; g_0)]$$

- Consequently, it is *asymptotically normal*

$$\sqrt{n}(\hat{\theta} - \theta_0) \sim_a N(0, V), \quad V := E[\phi_0(Z)\phi_0(Z)']$$

- *Confidence intervals* for any projection based on estimate of variance are asymptotically valid

$$\ell' \theta \in \left[\ell' \hat{\theta} \pm c \sqrt{\frac{\ell' \hat{V} \ell}{n}} \right], \quad \hat{V} = \text{Var}_n(\hat{\phi}(Z)), \quad \hat{\phi}(Z) := -\hat{J}^{-1} m(Z; \hat{\theta}, \hat{g}), \quad \hat{J} = E_n[a(Z; \hat{g})]$$

Python Pseudocode

```
# General DML pseudocode for linear moments: m(Z; theta, g)
def general_dml_linear_moment(Z, ell, nfolds, alpha, nu, jacob):
    # construct out-of-fold predictions from the nuisance estimator
    ghat = cross_val_predict(nuisance_estimator, Z, cv=nfolds)
    # use these predictions to define the empirical moment equation
    # and solve explicitly with respect to theta
    Jhat = np.mean(alpha(Z, ghat), axis=0)
    avg_nu = np.mean(nu(Z, ghat), axis=0)
    # solve for the empirical moment equation equals zero
    invJhat = np.linalg.pinv(Jhat)
    thetahat = invJhat @ avg_nu
    # construct approximate influence function for each sample
    phihat = - (nu(Z, ghat) - alpha(Z, ghat) @ thetahat) @ invJhat.T
    # variance estimate
    var = phihat.T @ phihat / phihat.shape[0]
    # estimate and standard error for any projection ell of the parameter
    point = ell @ thetahat
    stderr_ell = np.sqrt(ell.T @ var @ ell / phihat.shape[0])
    return point, stderr_ell
```

Estimate \hat{g} out of fold and predict

cross-fitted jacobian $\hat{J} := E_n[a(Z; \hat{g})]$

cross-fitted offset $E_n[v(Z; \hat{g})]$

Closed form solution: $\theta = \hat{J}^{-1} E_n[v(Z; \hat{g})]$

Approximate influence function
 $Z \rightarrow \hat{\phi}(Z) := \hat{J}^{-1} m(Z; \hat{\theta}, \hat{g})$

Empirical covariance of estimate
 $\hat{V} = E_n[\hat{\phi}(Z) \hat{\phi}(Z)']$

ATE under Conditional Exogeneity

- We observe n samples Z_1, \dots, Z_n where $Z_i = (X_i, D_i, Y_i)$

- Want to estimate average effect θ_0 , which satisfies:

$$M(\theta_0, g_0) := E[g_0(1, X) - g_0(0, X) - \theta_0] = 0$$

- Where:

$$g_0(D, X) := E[Y \mid D, X]$$

- We want to be able to use ML to learn regression function g !

Better Moment for ATE

- Add a “debiasing” correction:

$$\tilde{M}(\theta, g, a) = M(\theta, g) + E[a(D, X) (Y - g(D, X))]$$

- What is a_0 ? Should be such that

$$E[a_0(D, X) g(D, X)] = E[g(1, X) - g(0, X)]$$

- The following works: inverse propensity scoring

$$a_0(D, X) = \frac{D}{\Pr[D = 1|X]} - \frac{1 - D}{\Pr[D = 0|X]}$$

- Doubly robust estimation algorithm

Main Theorem (expanded) Define RMSE: $\|h\|_{L^2} = \sqrt{E[h(X)^2]}$

- If moment is Neyman orthogonal and RMSE of \hat{g} goes down at rate $n^{1/4}$, plus regularity conditions

$$n^{1/4} \|\hat{g} - g_0\|_{L^2} \approx 0$$

- Then the estimate $\hat{\theta}$ is *asymptotically linear*

$$\sqrt{n}(\hat{\theta} - \theta_0) \approx \sqrt{n} E_n[\phi_0(Z)], \quad \phi_0(Z) = -J_0^{-1} m(Z; \theta_0, g_0), \quad J_0 := \partial_{\theta} E[m(Z; \theta_0, g_0)]$$

- Consequently, it is *asymptotically normal*

$$\sqrt{n}(\hat{\theta} - \theta_0) \sim_a N(0, V), \quad V := E[\phi_0(Z)\phi_0(Z)']$$

- *Confidence intervals* for any projection based on estimate of variance are asymptotically valid

$$\ell' \theta \in \left[\ell' \hat{\theta} \pm c \sqrt{\frac{\ell' \hat{V} \ell}{n}} \right], \quad \hat{V} = \text{Var}_n(\hat{\phi}(Z)), \quad \hat{\phi}(Z) := -\hat{J}^{-1} m(Z; \hat{\theta}, \hat{g}), \quad \hat{J} = \partial_{\theta} E_n[m(Z; \hat{\theta}, \hat{g})]$$

Inference with Doubly Robust Algorithm

- Moment is Neyman orthogonal

$$m(Z; \theta, g) := g(1, X) - g(0, X) - a(D, X) (Y - g(D, X)) - \theta$$

- If RMSE of propensity and regression model goes down at rate $n^{1/4}$, plus regularity conditions
- Then the estimate $\hat{\theta}$ is *asymptotically linear*

$$\sqrt{n}(\hat{\theta} - \theta_0) \approx \sqrt{n} E_n[\phi_0(Z)], \quad \phi_0(Z) = -m(Z; \theta_0, g_0), \quad J_0 := \mathbf{1}$$

- Consequently, it is *asymptotically normal*

$$\sqrt{n}(\hat{\theta} - \theta_0) \sim_a N(0, V), \quad V := E[m(Z; \theta_0, g_0)^2]$$

- *Confidence intervals* for any projection based on estimate of variance are asymptotically valid

$$\ell' \theta \in \left[\ell' \hat{\theta} \pm c \sqrt{\frac{\ell' \hat{V} \ell}{n}} \right], \quad \hat{V} = \text{Var}_n(\hat{\phi}(Z)), \quad \hat{\phi}(Z) := -m(Z; \hat{\theta}, \hat{g}), \quad \hat{J} = \mathbf{1}$$

Python Pseudocode

```
cv = KFold(n_splits=nfolds, shuffle=True, random_state=123)
yhat0, yhat1 = np.zeros(y.shape), np.zeros(y.shape)
# we will fit a model  $E[Y | D, X]$  by fitting a separate model for  $D=0$ 
# and a separate model for  $D=1$ .
for train, test in cv.split(X, y):
    # train a model on training data that received zero and predict on all test data
    yhat0[test] = modely.fit(X[train][D[train]==0], y[train][D[train]==0]).predict(X[test])
    # train a model on training data that received one and predict on all test data
    yhat1[test] = modely.fit(X[train][D[train]==1], y[train][D[train]==1]).predict(X[test])
# prediction for observed treatment
yhat = yhat0 * (1 - D) + yhat1 * D
# propensity scores
Dhat = cross_val_predict(modeld, X, D, cv=cv, method='predict_proba', n_jobs=-1)[: , 1]
Dhat = np.clip(Dhat, trimming, 1 - trimming)
# doubly robust quantity for every sample
drhat = yhat1 - yhat0 + (y - yhat) * (D/Dhat - (1 - D)/(1 - Dhat))
point = np.mean(drhat)
var = np.var(drhat)
stderr = np.sqrt(var / X.shape[0])
return point, stderr, yhat, Dhat, y - yhat, D - Dhat, drhat
```

Partially Linear Model

- Relevant in many applications: dose-response curve in healthcare, effect of price on demand, return-on-investment
- Assume conditional exogeneity

$$Y^{(d)} \perp D \mid X$$

- Assume partially linear response: $E[Y \mid D, X] = \theta_0 D + f_0(X)$

$$Y = \theta_0 D + f_0(X) + \epsilon, \quad E[\epsilon \mid D, X] = 0$$

- Parameter of interest θ_0 is constant marginal effect of treatment
- Orthogonal moment are normal equations after residualization

$$\tilde{Y} = \theta_0 \tilde{D} + \epsilon, \quad E[\epsilon \mid \tilde{D}] = 0 \Rightarrow E[(\tilde{Y} - \theta_0 \tilde{D}) \tilde{D}] = 0$$

Orthogonal Method: Double ML

- **Double ML.** Split samples in half

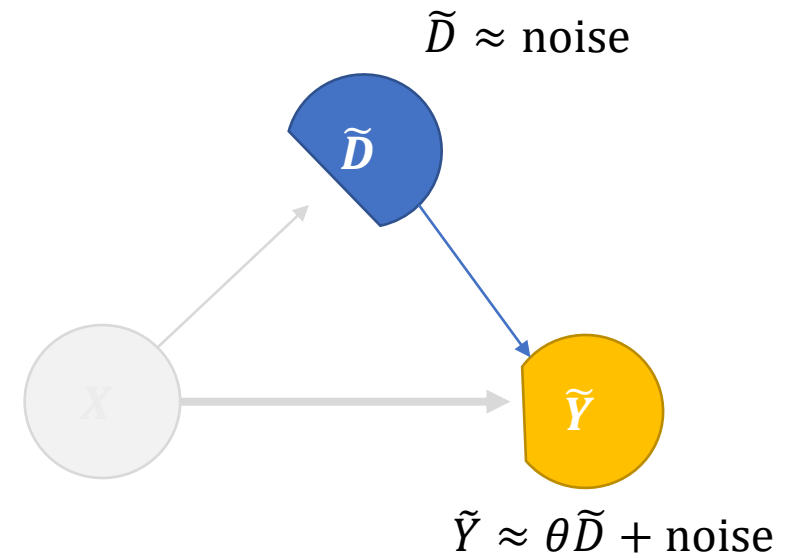
- Regress $Y \sim X$ with ML on first half, to get estimate $\hat{h}(S)$ of $E[Y|X]$
- Regress $D \sim X$ with ML on first half, to get estimate $\hat{p}(S)$ of $E[D|X]$
- Construct residuals on other half, $\tilde{D} := D - \hat{p}(X)$ and $\tilde{Y} := Y - \hat{h}(X)$
- Run OLS on residuals: $\tilde{Y} \sim \tilde{D}$ to get $\hat{\theta}$

- OLS equivalent to solving moment condition:

$$E_n[(\tilde{Y} - \theta \tilde{D})\tilde{D}] = 0$$

- Orthogonal Moment condition:

$$M(\theta, h, p) = E \left[\left(Y - h(X) - \theta (D - p(X)) \right) (D - p(X)) \right]$$



Orthogonal Method: Double ML

- OLS equivalent to solving moment condition:

$$E[(\tilde{Y} - \theta \tilde{D})\tilde{D}] = 0$$

- Orthogonal Moment condition:

$$M(\theta, h, p) = E \left[\left(Y - h(X) - \theta (D - p(X)) \right) (D - p(X)) \right]$$

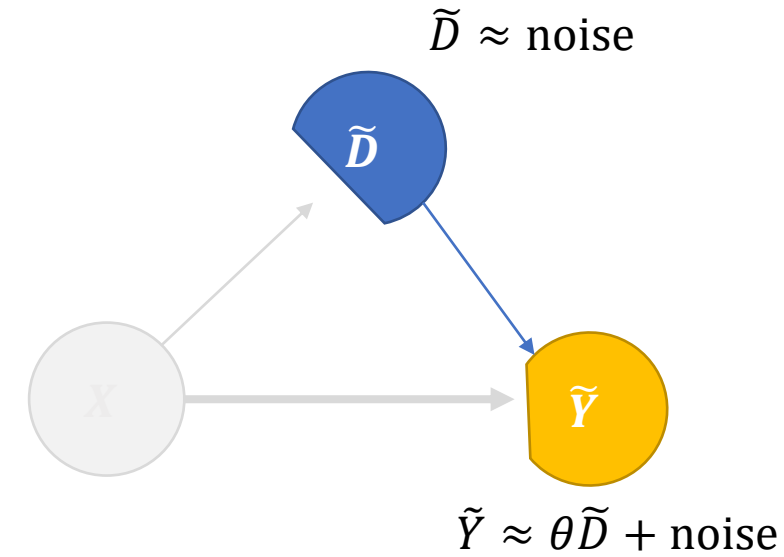
Verifying orthogonality

- Directional derivative with respect to h

$$\left. \partial_t E \left[\left(Y - h_0(X) - t v(X) - \theta_0 (D - p_0(X)) \right) (D - p_0(X)) \right] \right|_{t=0} = 0$$

- Directional derivative with respect to p

$$\left. \partial_t E \left[\left(Y - h_0(X) - \theta_0 (D - p_0(X) - t v(X)) \right) (D - p_0(X) - t v(X)) \right] \right|_{t=0} = 0$$



Main Theorem (expanded) Define RMSE: $\|h\|_{L^2} = \sqrt{E[h(X)^2]}$

- If moment is Neyman orthogonal and RMSE of \hat{g} goes down at rate $n^{1/4}$, plus regularity conditions

$$n^{1/4} \|\hat{g} - g_0\|_{L^2} \approx 0$$

- Then the estimate $\hat{\theta}$ is *asymptotically linear*

$$\sqrt{n}(\hat{\theta} - \theta_0) \approx \sqrt{n} E_n[\phi_0(Z)], \quad \phi_0(Z) = -J_0^{-1} m(Z; \theta_0, g_0), \quad J_0 := \partial_{\theta} E[m(Z; \theta_0, g_0)]$$

- Consequently, it is *asymptotically normal*

$$\sqrt{n}(\hat{\theta} - \theta_0) \sim_a N(0, V), \quad V := E[\phi_0(Z)\phi_0(Z)']$$

- *Confidence intervals* for any projection based on estimate of variance are asymptotically valid

$$\ell' \theta \in \left[\ell' \hat{\theta} \pm c \sqrt{\frac{\ell' \hat{V} \ell}{n}} \right], \quad \hat{V} = \text{Var}_n(\hat{\phi}(Z)), \quad \hat{\phi}(Z) := -\hat{J}^{-1} m(Z; \hat{\theta}, \hat{g}), \quad \hat{J} = \partial_{\theta} E_n[m(Z; \hat{\theta}, \hat{g})]$$

Inference with DML in PLR Setting

- If RMSE of propensity model and outcome model goes down at rate $n^{1/4}$, plus regularity conditions
- Then the estimate $\hat{\theta}$ is *asymptotically linear*

$$\sqrt{n}(\hat{\theta} - \theta_0) \approx \sqrt{n} E_n[\phi_0(Z)], \quad \phi_0(Z) = -J_0^{-1} (\tilde{Y} - \theta_0 \tilde{D}) \tilde{D}, \quad J_0 := E[\tilde{D}^2]$$

- Consequently, it is *asymptotically normal*

$$\sqrt{n}(\hat{\theta} - \theta_0) \sim_a N(0, V), \quad V := \frac{E[(\tilde{Y} - \theta_0 \tilde{D})^2 \tilde{D}^2]}{E[\tilde{D}^2]^2}$$

Jacobian of moments with respect to parameter; relates to identification strength
 $E[\tilde{D}^2] = E[\text{Var}(D|X)]$

- *Confidence intervals* for any projection based on estimate of variance are asymptotically valid

$$\ell' \theta \in \left[\ell' \hat{\theta} \pm c \sqrt{\frac{\ell' \hat{V} \ell}{n}} \right], \quad \hat{V} = \text{Var}_n(\hat{\phi}(Z)), \quad \hat{\phi}(Z) := -\hat{J}^{-1}(\hat{Y} - \hat{\theta} \hat{D}) \hat{D}, \quad \hat{J} = E_n[\hat{D}^2]$$

Python Pseudocode

```
cv = KFold(n_splits=nfolds, shuffle=True, random_state=123)
# out-of-fold predictions for y
yhat = cross_val_predict(modely, X, y, cv=cv, n_jobs=-1)
# out-of-fold predictions for D
Dhat = cross_val_predict(modeld, X, D, cv=cv, n_jobs=-1)
# calculate outcome and treatment residuals
resy = y - yhat
resD = D - Dhat
# final stage ols based point estimate and standard error
point = np.mean(resy * resD) / np.mean(resD**2)
epsilon = resy - point * resD
var = np.mean(epsilon**2 * D**2) / np.mean(D**2)**2
stderr = np.sqrt(var / X.shape[0])
return point, stderr
```

Practical Variants of Sample-Splitting

Cross-fitting and semi-cross-fitting

Cross-fitting

- Sample splitting is statistically lossy
- Only half of the data are used for the final parameter estimation
- Can we utilize all the data?
- *Cross-fitting*: analogous to cross-validation
- Use the second half to train g and predict on first half
- Then calculate parameter using all the data

Cross-fitting Estimation Algorithm

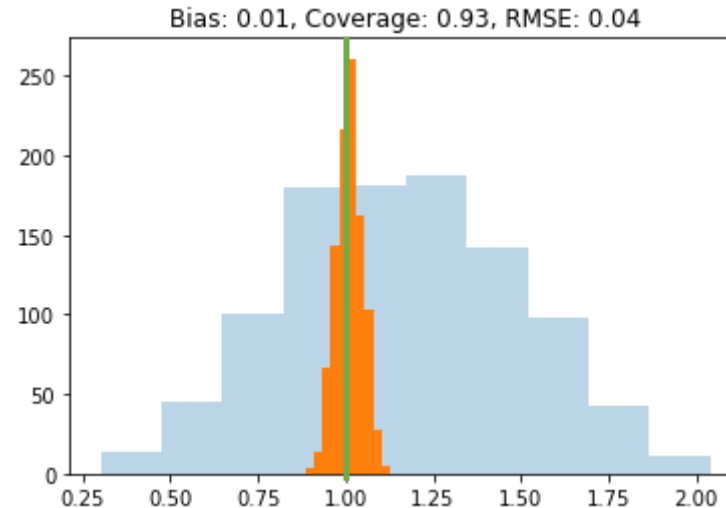
- Split the data in half
- On first half, estimate \hat{g}_1 of g_0 and predict on second half
- On second half, estimate \hat{g}_2 of g_0 and predict on first half
- On all data, solution $\hat{\theta}$ to empirical plug-in moment equation:

$$M_n(\hat{\theta}, \hat{g}) := \frac{1}{n} \sum_{i \in S_1} m(Z_i; \hat{\theta}, \hat{g}_2) + \frac{1}{n} \sum_{i \in S_2} m(Z_i; \hat{\theta}, \hat{g}_1) = 0$$

- In practice do this with $K \approx 3$ to 5 folds: for each fold k train on all other folds and predict on fold k

Natural Algorithm (Draft 3) Gone Right

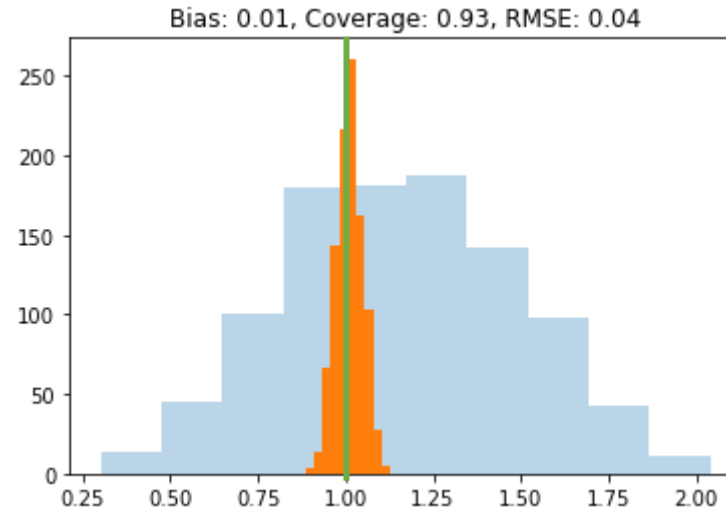
```
def dml2(X, D, y): # orthogonal dml with sample-splitting
    est_y = RandomForestRegressor(min_samples_leaf=20)
    yres = y - cross_val_predict(est_y, X, y, cv=3)
    est_t = RandomForestRegressor(min_samples_leaf=20)
    Dres = D - cross_val_predict(est_t, X, D, cv=3)
    theta = np.mean(yres * Dres) / np.mean(Dres**2)
    var = np.mean((Dres**2) * (yres - theta*Dres)**2) / np.mean(Dres**2)
    stderr = np.sqrt(var / X.shape[0])
    return theta, stderr
```



Natural Algorithm (Draft 3) Gone Right

```
from econml.dml import LinearDML

dml = LinearDML(model_y=RandomForestRegressor(min_samples_leaf=20),
                 model_t=RandomForestRegressor(min_samples_leaf=20))
est.fit(y, D, W=X).effect_inference()
```



Stacking and Model Selection

- If we want to choose among many models or perform stacking, we can just use a stacked or automl model in place of each ML model

Stacking ML Models

```
def dml2(X, D, y): # orthogonal dml with sample-splitting
    est_y = StackingRegressor([rf, nnet, gbf, lasso])
    yres = y - cross_val_predict(est_y, X, y, cv=3)
    est_t = StackingRegressor([rf, nnet, gbf, lasso])
    Dres = D - cross_val_predict(est_t, X, D, cv=3)
    theta = np.mean(yres * Dres) / np.mean(Dres**2)
    var = np.mean((Dres**2) * (yres - theta*Dres)**2) / np.mean(Dres**2)
    stderr = np.sqrt(var / X.shape[0])
    return theta, stderr
```

AutoML Models

```
from flaml import AutoML

def dml2(X, D, y): # orthogonal dml with sample-splitting
    est_y = AutoML()
    yres = y - cross_val_predict(est_y, X, y, cv=3)
    est_t = AutoML()
    Dres = D - cross_val_predict(est_t, X, D, cv=3)
    theta = np.mean(yres * Dres) / np.mean(Dres**2)
    var = np.mean((Dres**2) * (yres - theta*Dres)**2) / np.mean(Dres**2)
    stderr = np.sqrt(var / X.shape[0])
    return theta, stderr
```

Stacking and Model Selection

- If we want to choose among many models or perform stacking, we can just use a stacked or automl model in place of each ML model
- Model selection or stacking done many times within each training fold
- Computationally expensive and statistically lossy
- Can we use all the data to at least select among models?

Semi-Cross-fitting Estimation Algorithm

- Split the data in half (*in practice K folds*)
- On first half, estimate $\hat{g}_1^{(1)}, \dots, \hat{g}_1^{(L)}$ of g_0 and predict on second half
- On second half, estimate $\hat{g}_2^{(1)}, \dots, \hat{g}_2^{(L)}$ of g_0 and predict on first half
- Choose the model $\ell \in \{1, \dots, L\}$ that optimizes out-of-sample RMSE
- On all data, solution $\hat{\theta}$ to empirical plug-in moment equation:

$$M_n(\hat{\theta}, \hat{g}) := \frac{1}{n} \sum_{i \in S_1} m(Z_i; \hat{\theta}, \hat{g}_2^{(\ell)}) + \frac{1}{n} \sum_{i \in S_2} m(Z_i; \hat{\theta}, \hat{g}_1^{(\ell)}) = 0$$

Semi-Crossfitting

```
def dml2(X, D, y): # orthogonal dml with semi-crossfitting
    # cross val predict with many models
    est_y = [rf, gbf, lasso]
    yres = np.array([y - cross_val_predict(est, X, y, cv=3) for est in est_y])
    est_d = [rf, gbf, lasso]
    Dres = np.array([D - cross_val_predict(est, X, D, cv=3) for est in est_d])
    # select models with best out of fold performance
    best_y = np.argmin(np.mean(yres**2, axis=1))
    best_d = np.argmin(np.mean(Dres**2, axis=1))
    yres = yres[best_y]
    Dres = Dres[best_d]
    # go with their corresponding residuals
    theta = np.mean(yres * Dres) / np.mean(Dres**2)
    var = np.mean((Dres**2) * (yres - theta*Dres)**2) / np.mean(Dres**2)
    stderr = np.sqrt(var / X.shape[0])
    return theta, stderr
```

Semi-Crossfitting

- If the number of models L is small, then “spillover” is ok and approach still works. For practical purposes L should be thought as constant.
- Under further regularity, provably asymptotic normality holds if
$$\sqrt{\log(L)} = o(n^{1/4})$$

Semi-Cross-fitting with Stacking

- Split the data in half (*in practice K folds*)
- On first half, estimate $\hat{g}_1^{(1)}, \dots, \hat{g}_1^{(L)}$ of g_0 and predict on second half
- On second half, estimate $\hat{g}_2^{(1)}, \dots, \hat{g}_2^{(L)}$ of g_0 and predict on first half
- Construct weights $\alpha_1, \dots, \alpha_\ell$ on the models using all the data (stacking)

- On all data, solution $\hat{\theta}$ to empirical plug-in moment equation:

$$M_n(\hat{\theta}, \hat{g}) := \frac{1}{n} \sum_{i \in S_1} m(Z_i; \hat{\theta}, \hat{g}_2^{(\ell)}) + \frac{1}{n} \sum_{i \in S_2} m(Z_i; \hat{\theta}, \hat{g}_1^{(\ell)}) = 0$$

Semi-Crossfitting with Stacking

```
def dml2(X, D, y): # orthogonal dml with semi-crossfitting and stacking
    # cross val predict with many models
    est_y = [rf, gbf, lasso]
    ypreds = np.array([cross_val_predict(est, X, y, cv=3) for est in est_y]).T
    est_d = [rf, gbf, lasso]
    Dpreds = np.array([cross_val_predict(est, X, D, cv=3) for est in est_d]).T
    # calculate stacked residuals by finding optimal coefficients
    # and weighing out-of-sample predictions by these coefficients
    yres = y - LinearRegression().fit(ypreds, y).predict(ypreds)
    Dres = D - LinearRegression().fit(Dpreds, D).predict(Dpreds)
    # go with the stacked residuals
    theta = np.mean(yres * Dres) / np.mean(Dres**2)
    var = np.mean((Dres**2) * (yres - theta*Dres)**2) / np.mean(Dres**2)
    stderr = np.sqrt(var / X.shape[0])
    return theta, stderr
```

Semi-Crossfitting

- If the number of models L is small, then “spillover” is ok and approach still works. For practical purposes L should be thought as constant.
- Under further regularity, provably asymptotic normality holds if
$$\sqrt{L} = o(n^{1/4})$$

Equivalent view of cross-fitting with stacking (lens of FWL theorem)

- Construct out of fold predictions based on many ML models
- Use these predictions as engineered features X in a simple OLS regression on D, X
- Use the coefficient and standard error of D from this final OLS

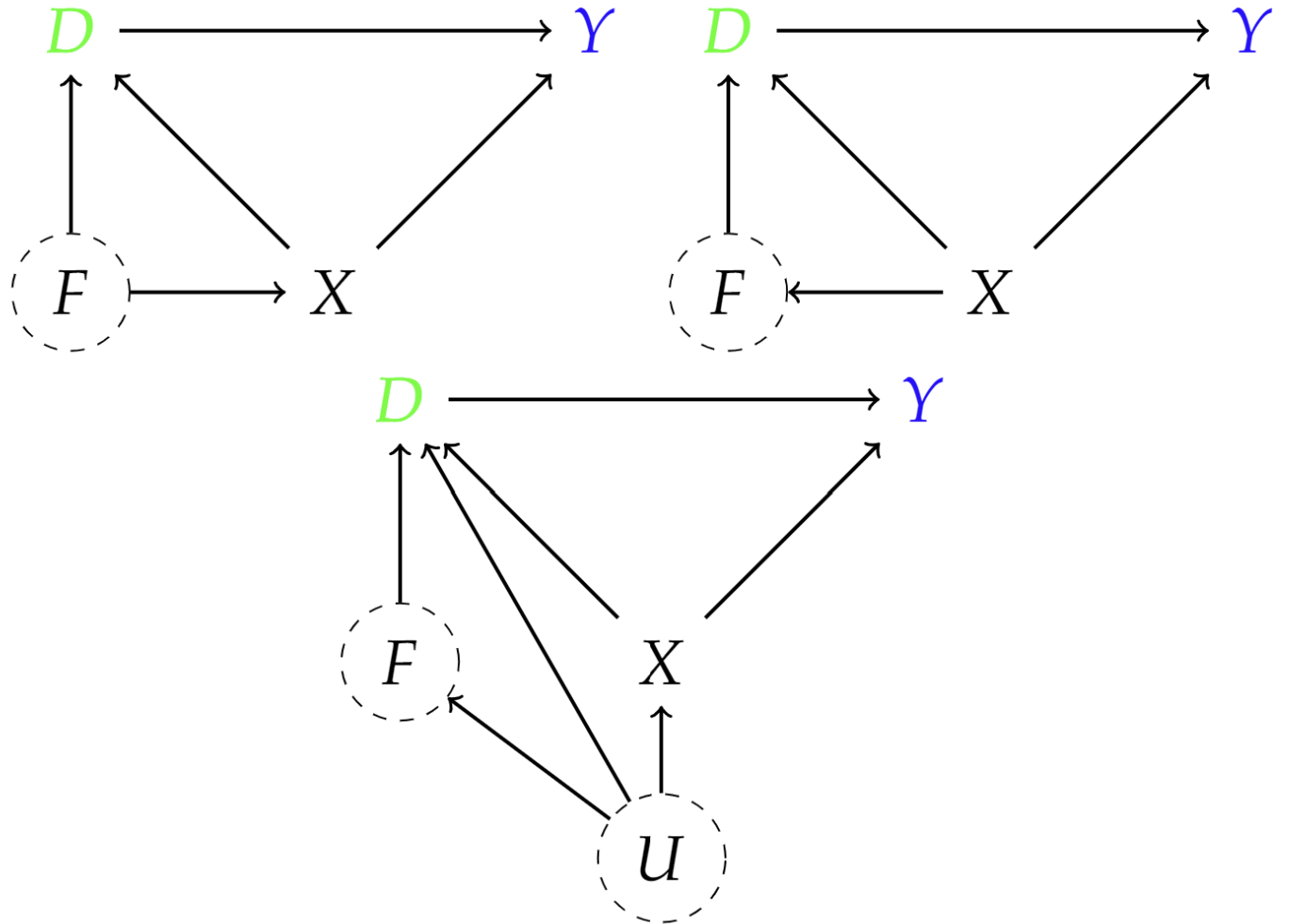
Application: Effect of 401k

Problem

- Impact of 401k eligibility on financial assets
- Working for a firm that offers access to 401k eligibility not randomly assigned
- After conditioning on important characteristics of the worker and job, assignment can be thought as exogenous
- Most important: income
- At least around 1991 (early in 401k deployment), workers would not really base their employment decision on 401k eligibility

Causal Diagrams

- F: firm characteristics
- D: eligibility for 401k
- Y: net financial assets
- X: age, income, family size, years of education, a married indicator, a two-earner status indicator, a defined benefit pension status indicator, an IRA participation indicator, and a home ownership indicator



Let's try it out!

Possible Violations

- F: firm characteristics
- D: eligibility for 401k
- Y: net financial assets
- X: age, income, family size, years of education, a married indicator, a two-earner status indicator, a defined benefit pension status indicator, an IRA participation indicator, and a home ownership indicator
- M: match amount

