Branch: master

Find file | Copy path

causallib / examples / ipw.ipynb

ehudkr Initial upload

fba7ff9 on May 6, 2019

1 contributor

Raw | Blame | History

1149 lines (1149 sloc) 547 KB

Inverse Probability Weighting Model

Inverse probability weighting is a basic model to obtain average effect estimation.

It calculates the probability of each sample to belong to its group,
and use its inverse as the weight of that sample:

$$w_i = \frac{1}{\Pr[A = a_i | X_i]}$$

In [1]:
```
%matplotlib inline
from causallib.datasets import load_smoking_weight
from causallib.estimation import IPW
from causallib.evaluation import PropensityEvaluator
from sklearn.linear_model import LogisticRegression
```

Data:

The effect of quitting to smoke on weight loss.
Data example is taken from Hernan and Robins Causal Inference Book (https://www.hsph.harvard.edu/miguel-hernan/causal-inference-book/)

In [2]:
```
data = load_smoking_weight()
data.X.join(data.a).join(data.y).head()
```

Out[2]:

	age	race	sex	smokeintensity	smokeyrs	wt71	active_1	active_2	edu
0	42	1	0	30	29	79.04	0	0	0
1	36	0	0	20	24	58.63	0	0	1
2	56	1	1	20	26	56.81	0	0	1
3	68	1	0	3	53	59.42	1	0	0
4	40	0	0	20	19	87.09	1	0	1

Model:

The causal model has a machine learning model at its core, provided as `learner` parameter.
This ML model will be used to predict probability of quit smoking given the covariates.
These probabilities will be used to obtain w_i.
Then, we'll estimate average balanced outcome using Horvitz–Thompson estimator:

$$\hat{E}[Y^a] = \frac{1}{\sum_{i\,:\,A_i = a} w_i} \cdot \sum_{i\,:\,A_i = a} w_i y_i$$

Lastly, we'll use these average counterfactual outcome estimation to predict the effect:

$$\hat{E}[Y^1] - \hat{E}[Y^0]$$

In [5]:
```
# Train:
learner = LogisticRegression(solver="liblinear")
ipw = IPW(learner)
ipw.fit(data.X, data.a)
```

Out[5]:
```
IPW(truncate_eps=None, use_stabilized=False,
    learner=LogisticRegression(C=1.0, class_weight=None, dual=Fal
se, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='l
iblinear',
          tol=0.0001, verbose=0, warm_start=False))
```

In [9]:
```
# We can now preict the weight of each individual:
ipw.compute_weights(data.X, data.a).head()
```

Out[9]:
```
0    1.127390
1    1.196073
2    1.153628
3    1.851189
4    1.361857
dtype: float64
```

In [10]:
```
# Estimate average outcome
outcomes = ipw.estimate_population_outcome(data.X, data.a, data.y
)
outcomes
```

Out[10]:
```
0    1.770965
1    5.284842
dtype: float64
```

In [12]:
```
# Estimate the effect:
effect = ipw.estimate_effect(outcomes[1], outcomes[0])
effect
```

Out[12]:
```
diff    3.513876
dtype: float64
```

We can see that on average, indiviudals who quit smoking gained 3.5 Lbs
on the course of 11 years

Non-default parameters

We just saw a simple example hiding many model's parameters.
We now dig a bit deeper to every stage.

Model definition

Machine learning model:
Any scikit-learn model can be specified (even pipelines)

In [19]:
```
learner = LogisticRegression(penalty="l1", C=0.01, max_iter=500,
```

```
solver='liblinear')
```

IPW model has two additional parameters:

- `truncate_eps`: a caliper value to trim very small or very large probabilities
- `stabilized`: Whether to scale weights with treatment prevalence

In [27]:
```
truncate_eps = 0.2
ipw = IPW(learner, truncate_eps=truncate_eps, use_stabilized=False)
```

In [28]:
```
ipw.fit(data.X, data.a);
```

Weight prediction options

Now we can predict the probability of quit smoking (`treatment_values=1`)
and validate our truncation worked:

In [29]:
```
probs = ipw.compute_propensity(data.X, data.a, treatment_values=1)
probs.between(truncate_eps, 1-truncate_eps).all()
```

```
Fraction of values being truncated: 0.23499.
```

Out[29]:
```
True
```

During the "predict" phase (i.e. computing weights or probabilities),
We can alter the parameters we placed during initiation:

In [35]:
```
probs = ipw.compute_propensity(data.X, data.a, treatment_values=1, truncate_eps=0.0)
probs.between(truncate_eps, 1-truncate_eps).all()
```

```
Fraction of values being truncated: 0.00000.
```

Out[35]:
```
False
```

We can even predict stabilized weights.
However, we will get a warning.
This is because treatment prevalence is an estimation of the trainning data.
During `fit`, when the model had it's initial values, `use_stabilized` was `False` (default).
So when coming to `compute_weights` now, the model will use the prevalence from the provided data to estimate treatment prevalence.
This is not a big deal here, since we compute on the same data we trained on, but this does not have to be the general case.
(This warning would not exists if we redefine the model with `use_stabilized=True` and re-train it)

In [36]:
```
stabilized_weights = ipw.compute_weights(data.X, data.a, treatment_values=1,
                                         truncate_eps=0.0, use_st
```

```
                                              truncate_eps=0.0, use_st
abilized=True)
weights = ipw.compute_weights(data.X, data.a, treatment_values=1,
                              truncate_eps=0.0)
stabilized_weights.eq(weights).all()
```

```
Fraction of values being truncated: 0.00000.
Fraction of values being truncated: 0.00000.
```

```
D:\workspaces\MLHLS\CausalLib\causallib\estimation\ipw.py:119: Ru
ntimeWarning: Stabilized is asked, however, the model was not tra
ined using stabilization, and therefore, stabilized weights are t
aken from the provided treatment assignment.
  RuntimeWarning)
```

Out[36]: False

Since IPW utilizes probabilites, for each sample we can get a probability (or weight) for each treatment value

In [41]:
```
# ipw.compute_weight_matrix(data.X, data.a).head()
ipw.compute_propensity_matrix(data.X, data.a).head()
```

```
Fraction of values being truncated: 0.23499.
```

Out[41]:

	0	1
0	0.800000	0.200000
1	0.800000	0.200000
2	0.716160	0.283840
3	0.520581	0.479419
4	0.765621	0.234379

Effect estimation options

We can choose whether we wish for addtive (`diff`) or multiplicative (`ratio`) effect
(If outcome y was probabilites, we could also ask for odds-ratio (`or`))

Providing weights `w` is optional, if not provided the weights would be simply calulated again using the provided `X`.

In [47]:
```
outcomes = ipw.estimate_population_outcome(data.X, data.a, data.y
, w=weights)
effects = ipw.estimate_effect(outcomes[1], outcomes[0], effect_ty
pes=["diff", "ratio"])
effects
```

Out[47]:
```
diff     2.755327
ratio    2.142714
dtype: float64
```

Evaluation

Evaluation

We can also evaluate the performance of the IPW model

Simple evaluation

Evaluates a fitted model on the provided dataset

In [48]:
```python
evaluator = PropensityEvaluator(ipw)
results = evaluator.evaluate_simple(data.X, data.a, data.y, plots
=None)
```

```
Fraction of values being truncated: 0.23499.
Fraction of values being truncated: 0.23499.
Fraction of values being truncated: 0.23499.
Fraction of values being truncated: 0.23499.
```

`results` contains `models`, `plots` and `scores`, but since we did not ask for plots, and did not refit the model, our main interest is the scores. We have both the prediction performance scores and a table1 with standardized mean differences with and without balancing

In [52]:
```python
results.scores.prediction_scores
```

Out[52]:

	accuracy	precision	recall	f1	roc_auc	avg_precision	hinge
0	0.743934	0.541667	0.032258	0.06089	0.613656	0.36485	1.1164

In [54]:
```python
results.scores.covariate_balance.head()
```

Out[54]:

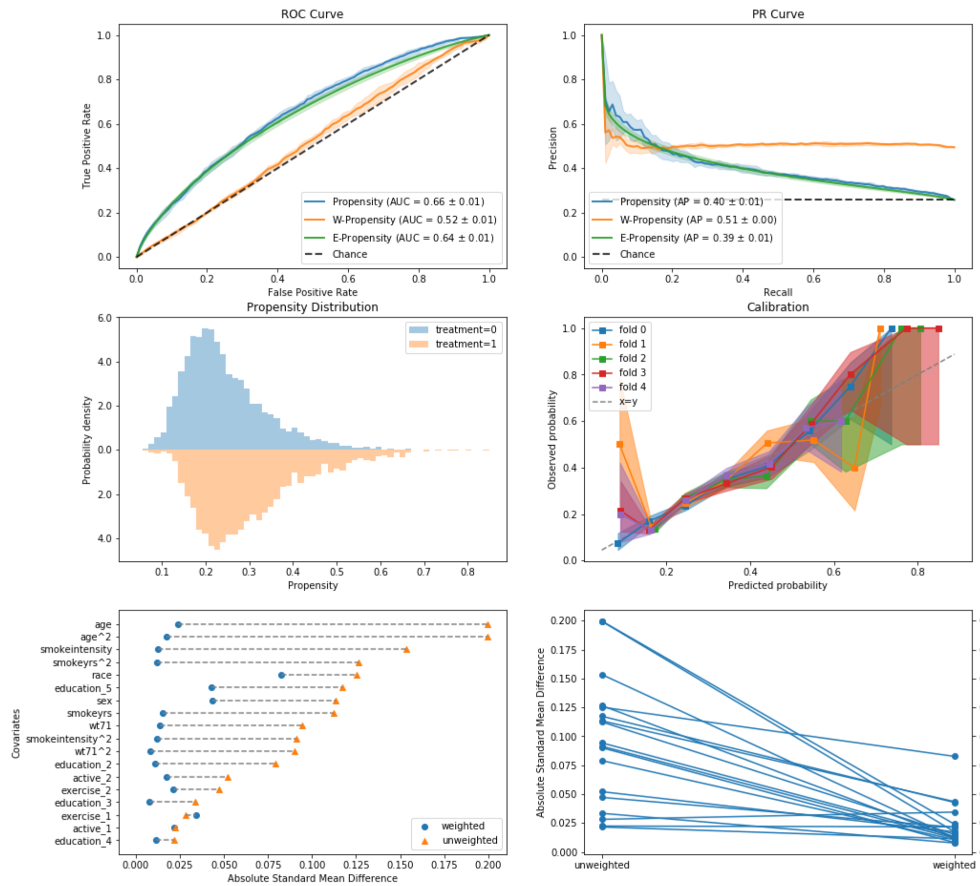
abs_smd	weighted	unweighted
covariate		
age	0.048604	0.199560
race	0.167319	0.125194
sex	0.145940	0.113323
smokeintensity	0.018511	0.153345
smokeyrs	0.021658	0.112470

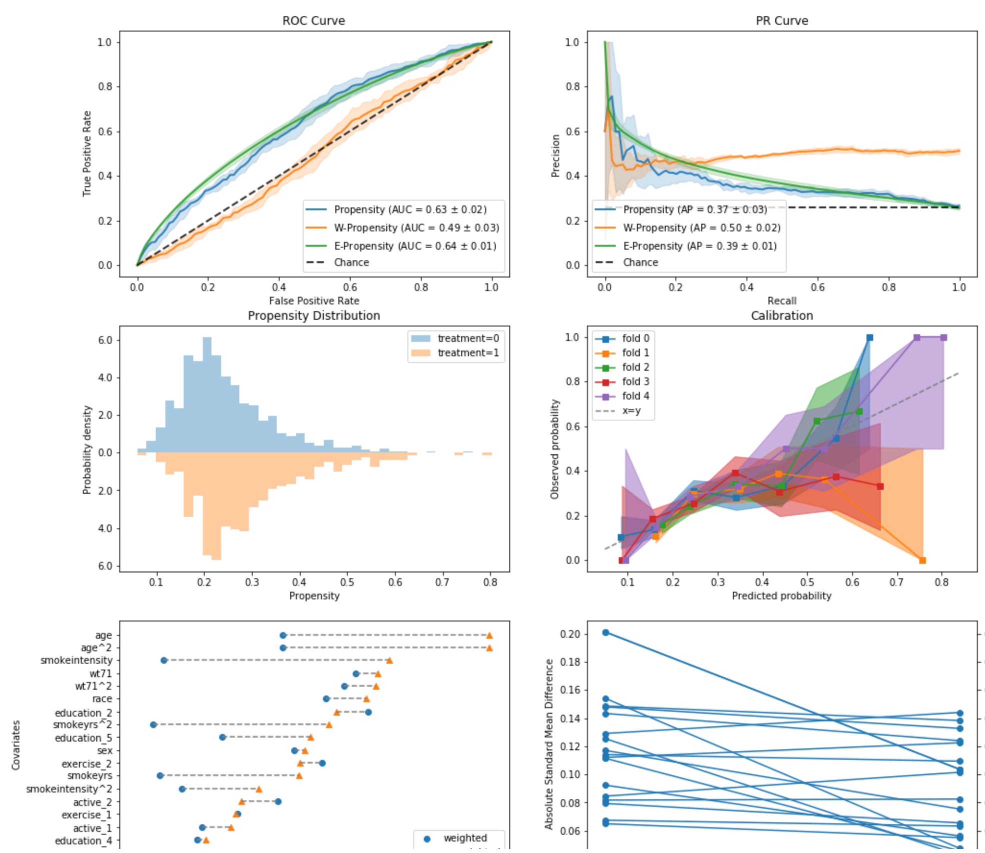
Thorough evaluation

Can check general model specification as it evaluates using cross-validation and refitting the model on each fold

In [56]:
```python
from sklearn import metrics
plots=["roc_curve", "pr_curve", "weight_distribution",
       "calibration", "covariate_balance_love", "covariate_balanc
e_slope"]
metrics = {"roc_auc": metrics.roc_auc_score,
```

```
           "avg_precision": metrics.average_precision_score,}
ipw = IPW(LogisticRegression(solver="liblinear"))
evaluator = PropensityEvaluator(ipw)
results = evaluator.evaluate_cv(data.X, data.a, data.y,
                               plots=plots, metrics_to_evaluate=
metrics)
```





In [57]:
```
results.scores.prediction_scores
```

Out[57]:

			roc_auc	avg_precision
phase	**fold**			
train	**0**	**0**	0.662740	0.409813
	1	**0**	0.656866	0.403830
	2	**0**	0.644306	0.389844
	3	**0**	0.670676	0.424142
	4	**0**	0.663786	0.394718
valid	**0**	**0**	0.653791	0.400107
	1	**0**	0.635352	0.345427
	2	**0**	0.644042	0.397700
	3	**0**	0.592295	0.323602
	4	**0**	0.621713	0.402692

In [60]:
```
print(len(results.models))
results.models[2]
```

```
5
```

Out[60]:
```
IPW(truncate_eps=None, use_stabilized=False,
    learner=LogisticRegression(C=1.0, class_weight=None, dual=Fal
se, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='l
iblinear',
```