Session 5: Economics Approaches to Machine Learning

2021 August 12

Dr. Richard M. Crowley
rcrowley@smu.edu.sg
http://rmc.link/

Main applications

Bias, #1: Quantifying bias in wages

- Based on the City of Chicago wage data set

Dependent Variable

- Annual salary

Independent Variables

- Job title
- Department
- Full time / part time
- Salaried or hourly
- Female

> This is a simple test to showcase the toolchain for SHAP

Bias, #2: Political bias in hate speech classification

Dependent Variable

- Offensive speech

Independent Variables

- “Non-offensive” tweets from left-wing/right-wing/neutral groups
- Non-offensive tweets from GermEval 1 & 2
- Tweet topics

Word-level examination

From Wich, Bauer and Groh (2020 WOAH)

Causal ML: Quantifying the impact of 401(k)s on wealth

- An illustrative implementation of using Double ML for causality
- The key motivator for the method is the

Dependent Variable

- Net financial assets

Independent Variables

- Treatment: 401K eligibility
- Age
- Income
- Family size
- Years of education
- Marital status
- Two-earner status indicator
- Defined benefit pension indicator
- IRA participation
- Home ownership indicator

From the web appendix of Chernozhukov et al. (2017 AER)

Introduction to Bias using SHAP

An example of quantifying bias

- Data: City of Chicago salaries
  - 33,586 employees
- Trained using a simple XGBoost model
- Features:
  - Job title
  - Department
  - Full time / part time
  - Salaried or hourly
  - Female

> Is there gender bias in annual compensation?

The data

```python
vars = ['Job.Titles', 'Department', 'Full.Time', 'Salaried', 'Female']
df[vars]
```

```
##                                   Job.Titles         Department  Full.Time  \
## 0                                   SERGEANT             POLICE          1
## 1       POLICE OFFICER (ASSIGNED AS DETECTIVE)           POLICE          1
## 2                                      Other   GENERAL SERVICES          1
## 3                                      Other        WATER MGMNT          1
## 4                                      Other        TRANSPORTN           1
## ...                                      ...                ...        ...
## 33581                         POLICE OFFICER             POLICE          1
## 33582                         POLICE OFFICER             POLICE          1
## 33583                         POLICE OFFICER             POLICE          1
## 33584                         POLICE OFFICER             POLICE          1
## 33585                                  Other              Other          1
##
##         Salaried  Female
## 0              1     0.0
## 1              1     1.0
## 2              1     1.0
## 3              1     0.0
## 4              0     0.0
## ...          ...     ...
## 33581          1     1.0
```

One hot encoding categorical data

- Pandas has a function for this, `pd.get_dummies()`
  - `prefix=` lets us name the columns of the output
- As `pd.get_dummies()` outputs a new data frame only containing the new columns, we need to join them back
  - `df.join()` makes this quick and easy

```
one_hot1 = pd.get_dummies(df['Job.Titles'], prefix='Job.Titles')
one_hot2 = pd.get_dummies(df['Department'], prefix='Department')

df = df.join(one_hot1)
df = df.join(one_hot2)
```

Prepping XGBoost

We did this in Session 2

```python
vars = one_hot1.columns.tolist() + \
       one_hot2.columns.tolist() + \
       ['Full.Time', 'Salaried', 'Female']
dtrain = xgb.DMatrix(df[vars], label=df['Salary'], feature_names=vars)
```

```python
param = {
    'booster': 'gbtree',                # default -- tree based
    'nthread': 8,                       # number of threads to use for parallel processing
    'objective': 'reg:squarederror',    # RMSE error
    'eval_metric': 'rmse',              # maximize ROC AUC
    'eta': 0.3,                         # shrinkage; [0, 1], default 0.3
    'max_depth': 6,                     # maximum depth of each tree; default 6
    'gamma': 0,                         # set above 0 to prune trees, [0, inf], default 0
    'min_child_weight': 1,              # higher leads to more pruning of tress, [0, inf], default 1
    'subsample': 1,                     # Randomly subsample rows if in (0, 1), default 1
}
num_round=30
```

Building our model and prepping SHAP

- We call `xgb.train()` to fit our XGBoost model

```python
model_xgb = xgb.train(param, dtrain, num_round)
```

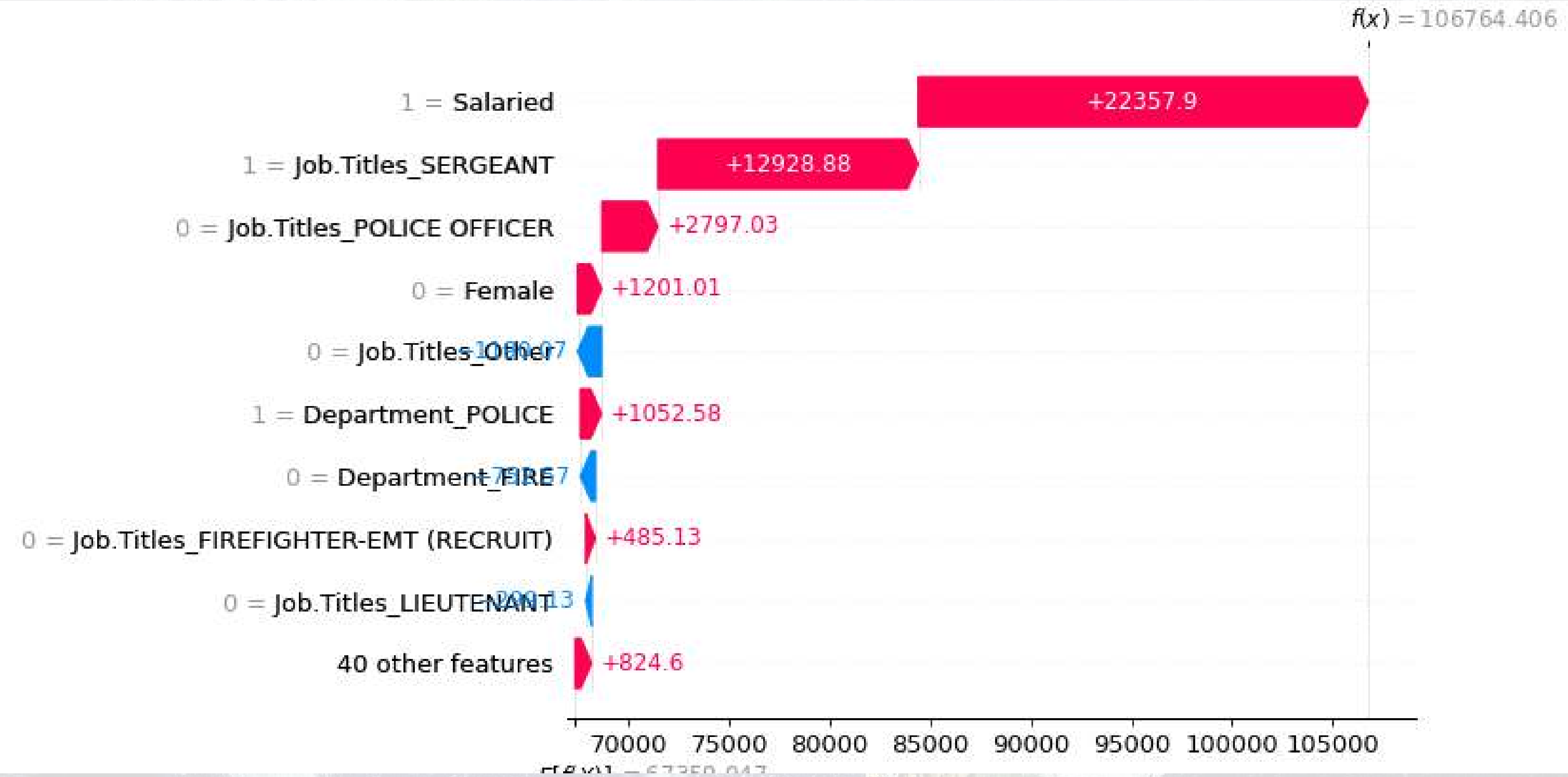
- Since XGBoost is a tree-based model, we will use SHAP's `shap.TreeExplainer()` function to analyze the model
- Since we only have in-sample data, we will compute SHAP on the same data the XGBoost model was fit to
- We will also prepare a small sample for more CPU-intense analyses

```python
explainer = shap.TreeExplainer(model_xgb)
shap_values = explainer(df[vars])

df_small = df.sample(frac=0.01)
shap_values_small = explainer(df[vars])
```

Explaining a single observation

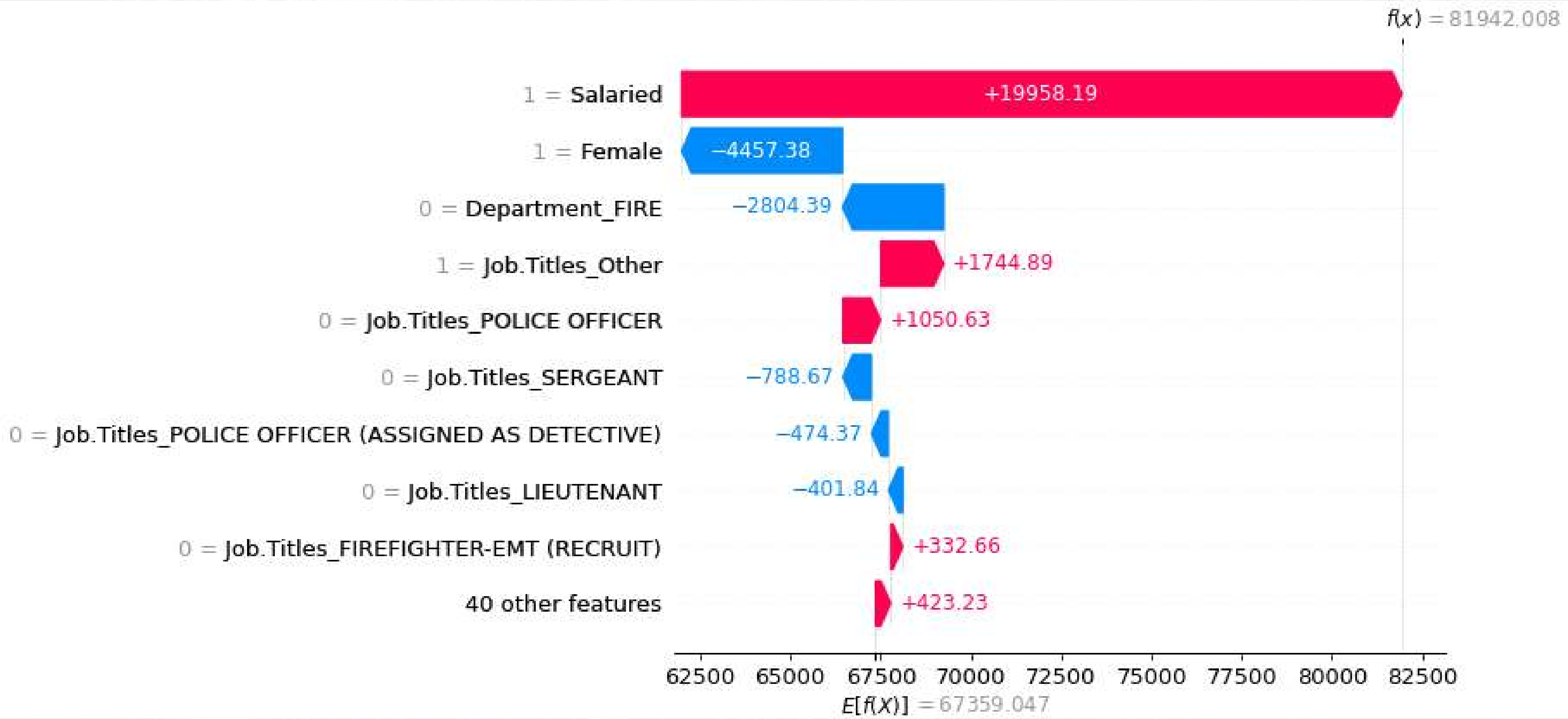
```python
shap.plots.waterfall(shap_values[0])
```



Here we see that having `Female=0` was the fourth most influential feature in the model, and that it led to a *higher* predicted salary

Explaining a single observation

```python
shap.plots.waterfall(shap_values[2])
```



Here we see that having `Female=1` was the second most influential feature in the model, and that it led to a *lower* predicted salary

What exactly is SHAP?

> Aims to provide an explanation of the importance of model inputs in explaining model output

- Game theoretic and theory driven
- Unifies six other methods that tried to address this problem
- It is a model itself, a model to explain models
- Provides a simple to understand output

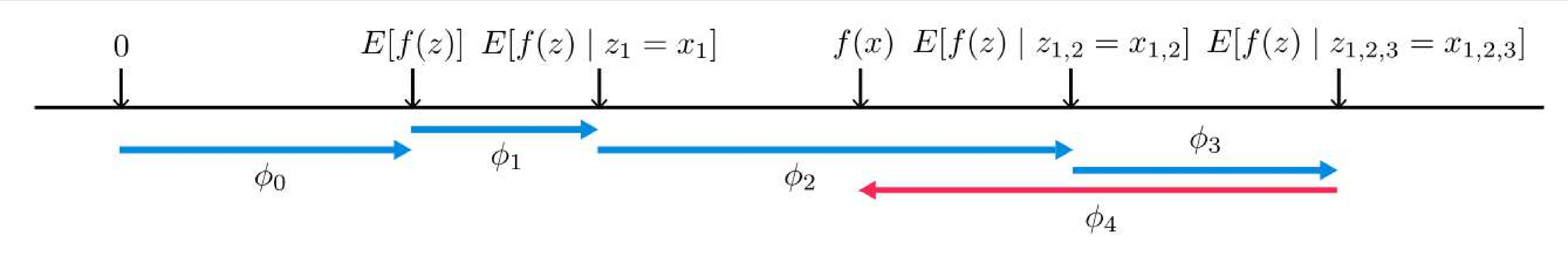
> SHAP: *SH*apley *A*dditive ex*P*lanations

- Based on Shapley, 1953, “A value for n-person games.”
- SHAP itself is from Lundberg and Lee (2017)

Principles of SHAP

1. Local accuracy
   - The simple model is able to accurately predict a model output on small subsets of the data
2. Missingness
   - SHAP only uses data the original model had access to
   - If data was missing from the original model, SHAP won’t use it
3. Consistency
   - Akin to transitivity conditions in utility theory (Savage Axioms)
     - But instead of “utility,” we have “simplified model’s input’s contribution”

Intuition of SHAP

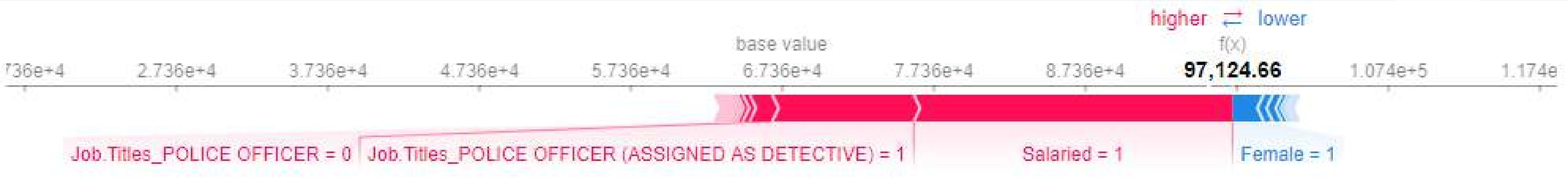


- SHAP is defined by a series of [conditional] expectation of the impact of an input
- For linear models, order of selecting inputs has no effect
- For nonlinear models, SHAP averages inputs' conditional expected impact over all possible orderings

Charting with SHAP

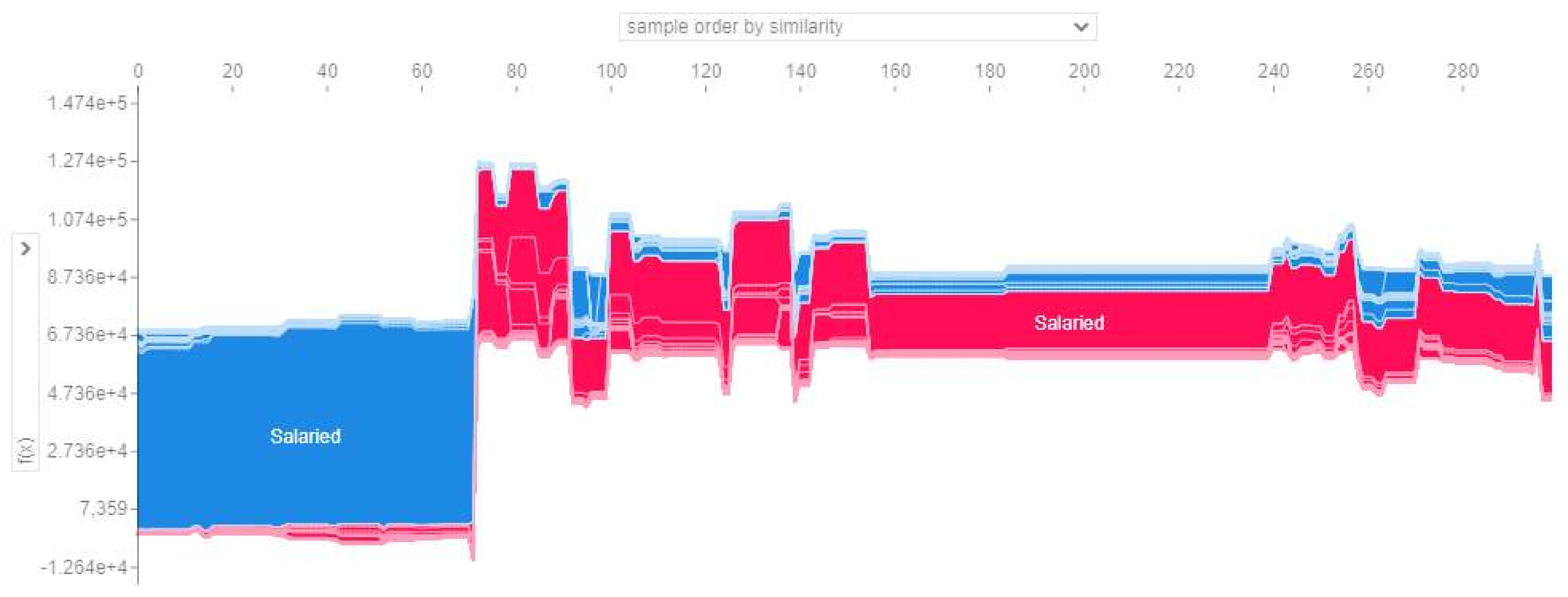
A more concise point visualization

```python
shap.plots.force(shap_values[1])
```



Aggregating across the data

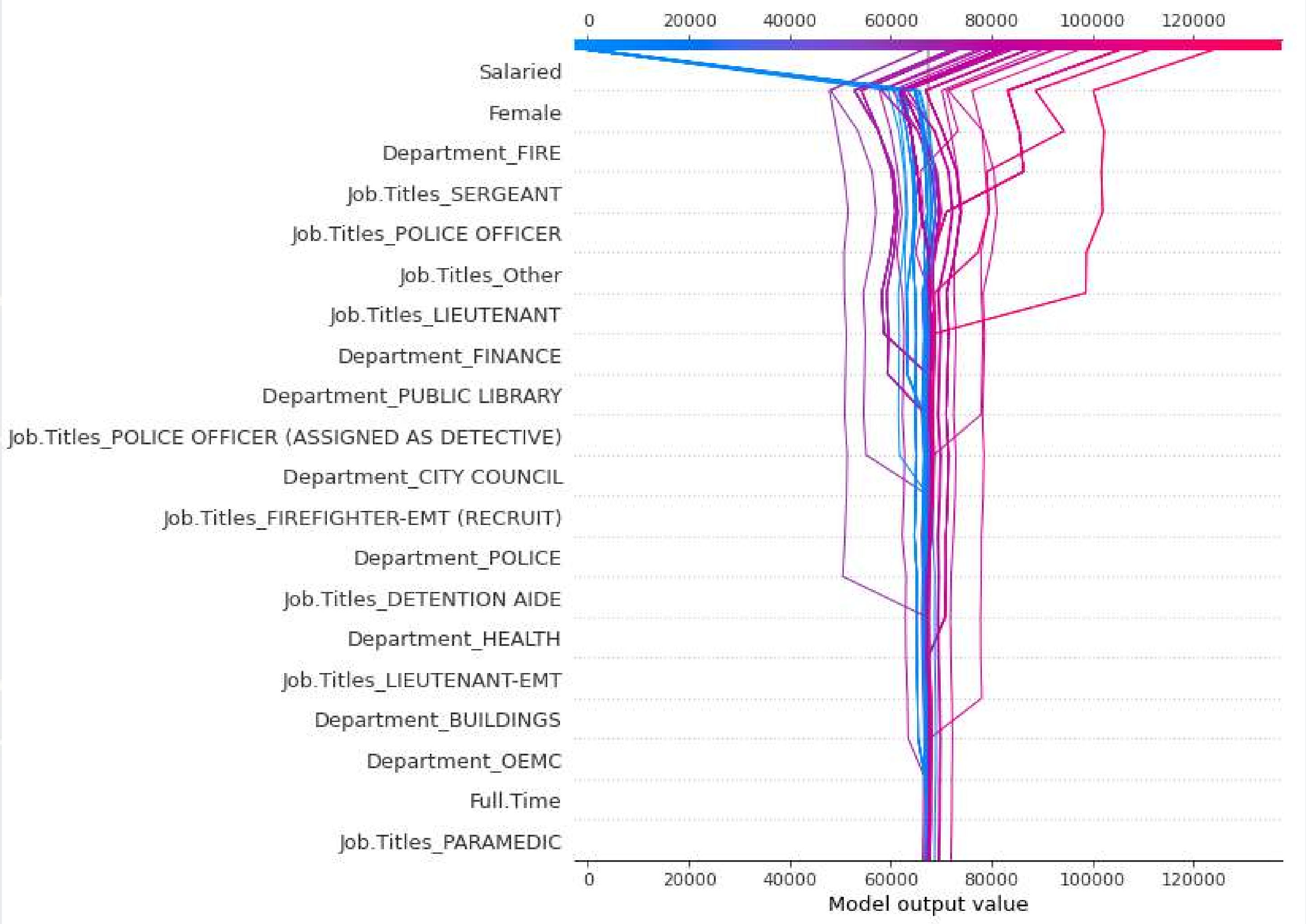
```python
N=300
shap.plots.force(explainer.expected_value, shap_values.sample(N).values, feature_names=vars)
```



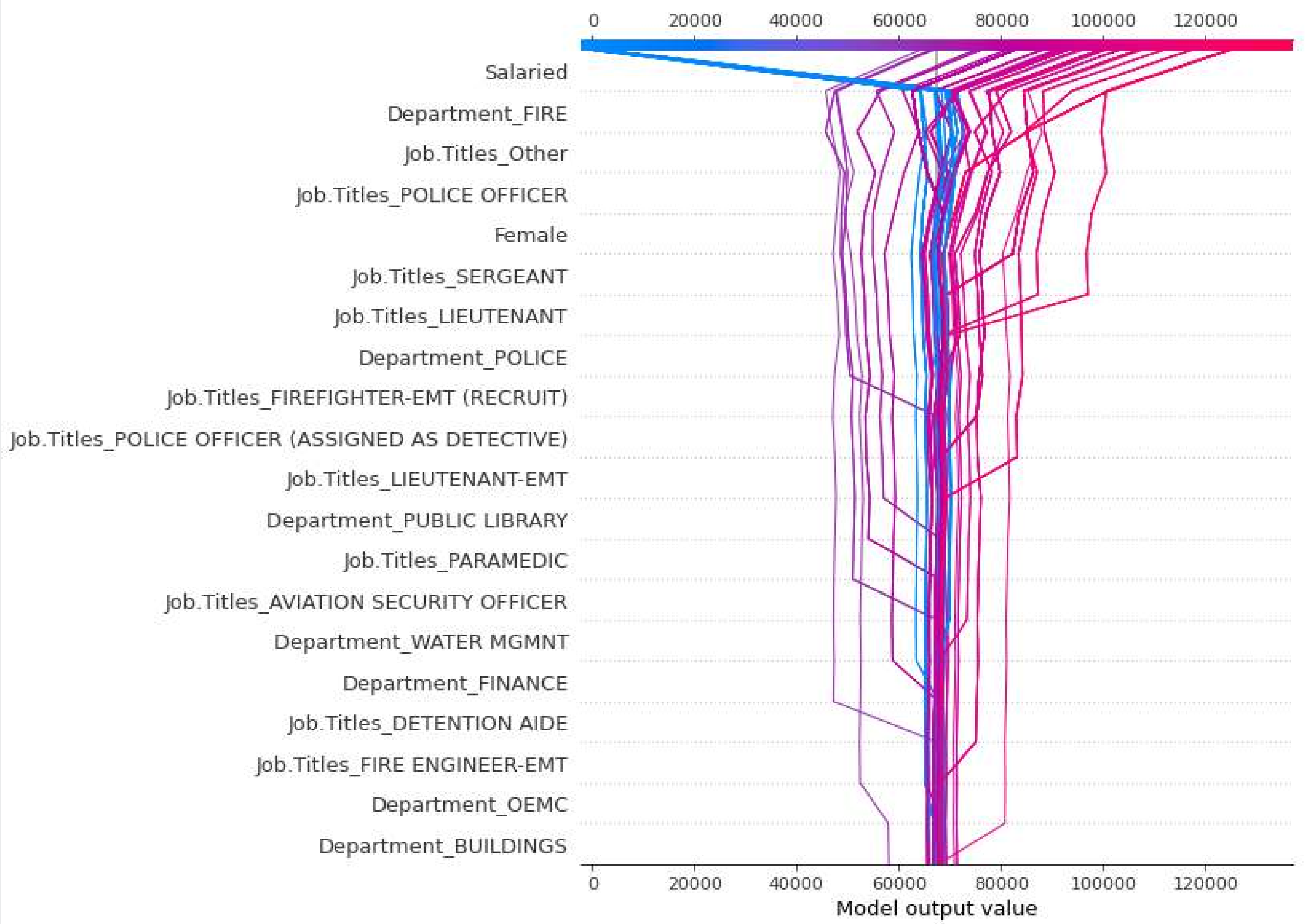
Seeing more variables' impact

- A "Decision plot" uses a line chart to show the impact of more measures across the data

```python
shap.decision_plot(
  explainer.expected_value,
  explainer.shap_values(df_small[df_small.Female==1][vars
  feature_names=vars)
```



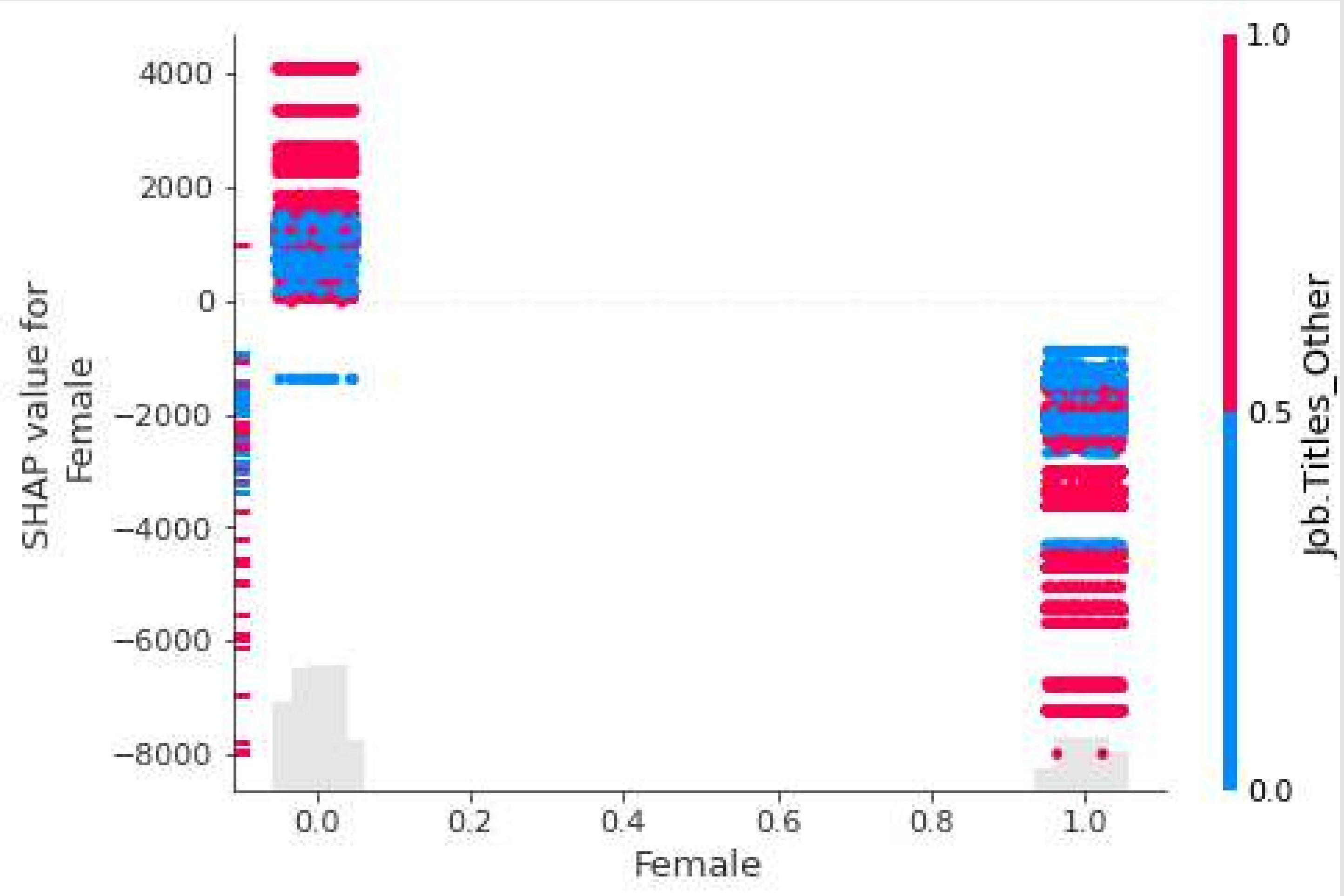

```python
shap.decision_plot(
  explainer.expected_value,
  explainer.shap_values(df_small[df_small.Female==0][vars
  feature_names=vars)
```



Aggregate analysis of an individual variable

- If we want to see the full impact of “Female” on outcomes in our data, a scatter plot is useful

```python
shap.plots.scatter(shap_values[:,"Female"], color=shap_values)
```

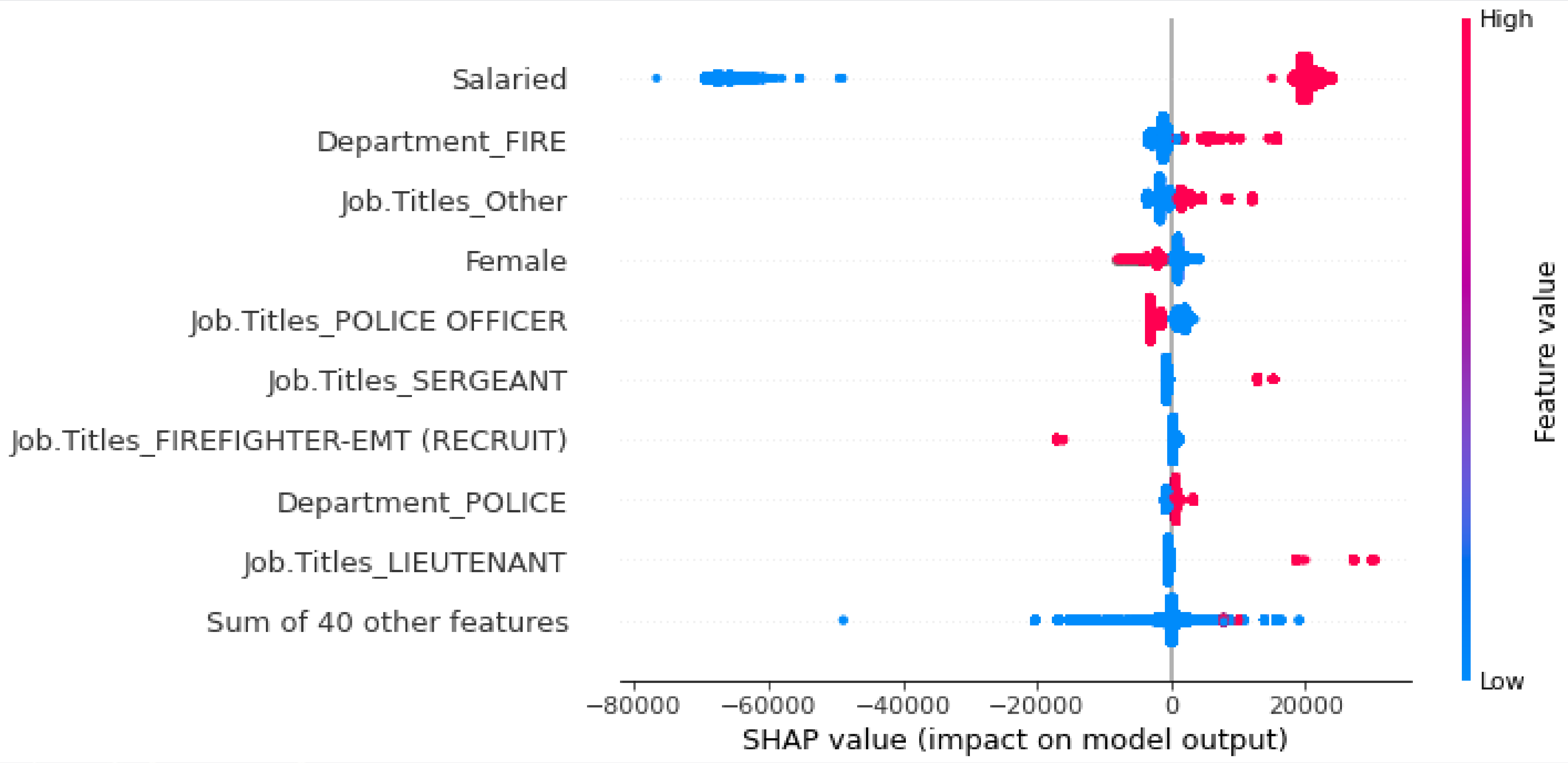


> Remember that our model is nonparametric! Signs can be different even when the variable doesn’t change due to interactive effects

Multiple scatterplots at once: Bee swarm

- If you want a concise way to present multiple variables, the bee swarm plot can be useful

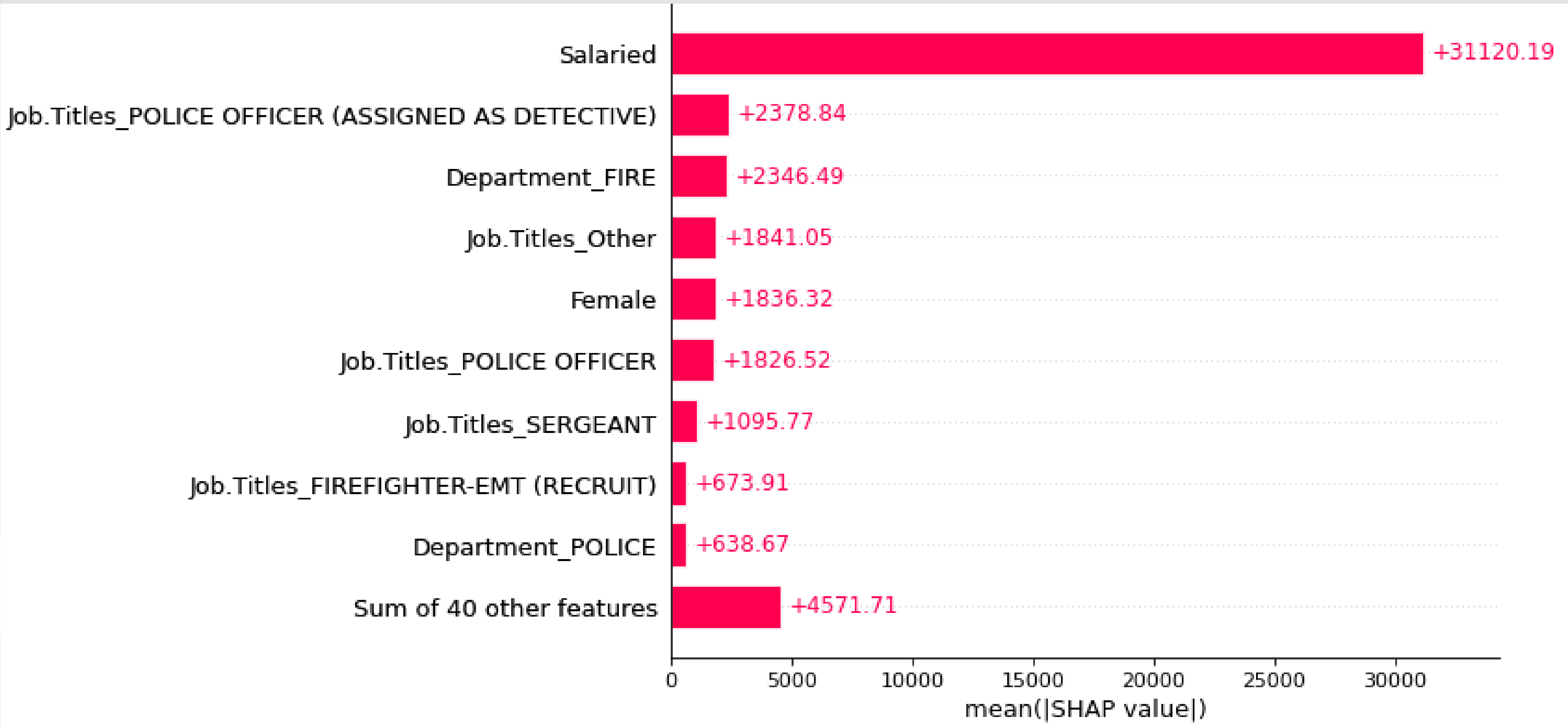
```
shap.plots.beeswarm(shap_values)
```



Importance plot

- Lastly, we can replicate XGBoost's importance plot using $|\text{SHAP}|$

```python
shap.plots.bar(shap_values)
```



> This may not be useful for XGBoost since it already has an importance metric, but many other models lack it

Addendum: Using R

- If you are working explicitly with XGBoost, there is a great `SHAPforxgboost` package
- To interface with the python `shap` package, you can use `shapper`
- There is also `shapr`, though it isn't as full-featured.

SHAP for hate speech bias

Paper background

> How does political bias in data impact hate speech classification?

- Baseline data is often a critical issue in measure construction
  - This paper takes a strong stance in demonstrating this, by using a fixed, unbiased sample of hate speech content
  - E.g., the “1” class is not impacted by any political bias, only the “0” class
- The authors aim to show how using a politically biased non-offensive baseline can induce bias in hate speech classification models

> From Wich, Bauer and Groh (2020 WOAH)

The models

- The authors construct hate speech detection models using a combination of four corpuses
  1. A baseline model from GermEval 1 & 2 that is politically neutral
  2. A set of politically left-wing tweets
  3. A set of politically right-wing tweets
- In order to see the effect of political leaning on the model, they also run the model on mixtures of corpuses that are 1/3 or 2/3 neutral, with the remaining text from one of the non-neutral corpuses

> Left-wing text induces a statistically significant divergence in model performance when more than 2/3 of the text is left-wing

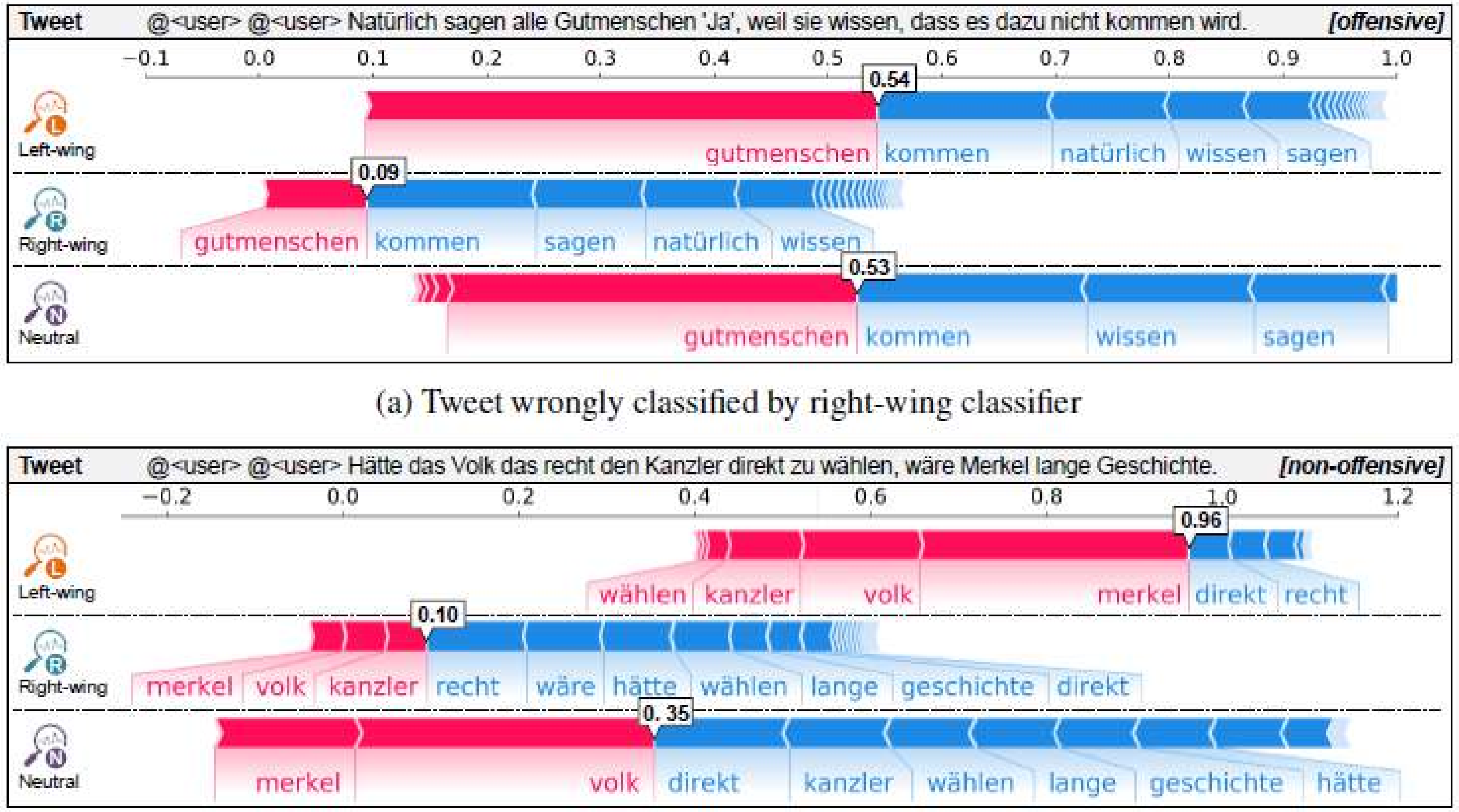
> Right-wing text induces a statistically significant divergence in model performance when only 1/3 of the text is right-wing

Applying SHAP to the models

- Same workflow as we did, except tailored for a neural network
  - Just replace the TreeExplainer with DeepExplainer
- Conceptually, SHAP will behave the same across any nonlinear model
- Since their data is word-level, the features fed to SHAP will be one hot encoded vectors of words

> SHAP will weight the extent to which a word indicates the presence of hate speech, in [conditional] expectation

Examples of bias with SHAP



(a) Tweet wrongly classified by right-wing classifier

(b) Tweet wrongly classified by left-wing classifier

Figure 3: SHAP values for the two selected tweets

1. @user @user Of course, all **do-gooders** say “yes,” because they know that it won’t happen.
   - Tagged categorization: Offensive
2. If the **people** had the right to elect the chancellor directly, **Merkel** would have been history a long time ago.
   - Tagged categorization: Not offensive

What else could this paper have done?

1. Leverage the topic model to show if bias is generally pervasive when using biased corpuses
   - Or perhaps bias creeps in only in certain contexts
   - How? Examine SHAP at a per-topic level
2. Quantify the extent of bias
   - They already quantified the impact on model accuracy, but innacuracy doesn’t directly imply bias
   - How? Examine SHAP at the corpus level

Double ML: Theory

Background

- There are a number of relevant papers published in economics in recent years developing and using Double ML
- The method is developed largely from:
  - Chernozhukov et al. (2017 AER), “Double/debiased/Neyman machine learning of treatment effects”
  - Chernozhukov et al. (2018 Econometrics J), “Double/debiased machine learning for treatment and structural parameters.”

> Impact or overlap with methodological work by Susan Athey, Matthew Gentzkow, Trevor Hastie, Guido Imbens, Matt Taddy, and Stefan Wager

What is Double ML?

1. Split your sample as you would for K-fold cross validation, into sets $\{I_k\}_{k\in\{1,...,K\}}$
   - K sample of N/K observations each
   - Let $I_k^c = \cup\{I_j\}_{j\neq k}$
2. Construct K estimators using a machine learning estimator over nuisance parameters (e.g., controls) applied to the data I_K^c
3. Average the K estimators to obtain a final estimator
   - This average estimator is approximately unbiased and normally distributed
   - The estimator is also asymptotically efficient

> And repeat. Bootstrap this out and take the mean or median of the estimators

Where Double ML excels: Endogenous treatment

- Suppose a policy affects a subset of individuals (people, corporations, etc.)
- Suppose individuals have the ability to alter their treatment status
  - E.g., state laws (move), labor laws, etc.
- Linear controls may be insufficient to claim causality of the treatment on anything

> There are a lot of older methods that try to address this, though incompletely

1. Linear controls
2. Propensity score adjustments (e.g., weighting)
3. Matching methods
4. “doubly-robust” estimators

Why is machine learning needed?

- Suppose a true form of a specification is as follows
  - T is a treatment indicator, C is a vector of controls

$$Y = g_0(T, C) + \varepsilon_1$$
$$T = m_0(C) + \varepsilon_2$$

- We often assume g_0 to be something like $\alpha + \theta_0\, T + \gamma \cdot C$
- We often assume m_0 to be a constant (i.e., assume that T is exogenous)

> We know these assumptions aren't true!

Why is machine learning needed?

> How can we estimate a more general form for g_0 and m_0?

- We could use a more flexible econometric approach, such as including interactions between T and C
  - This is still very restrictive – purely linear
- We could include transformations of C and its interactions
  - This is still restrictive – T is additive separable
- We could use a nonparametric estimator!
  - This is where machine learning is very useful: efficient and reasonably accurate nonparametric estimation
    - LASSO, random forest, XGBoost, etc.

Model variants

- The models described in the last few slides are referred to as the “Interactive regression model” or IRM
- If you can separate your treatment effect from the controls but suspect nonlinear effects of controls, the “Partially linear regression model” or PLR is appropriate
  - Solves $Y = \theta_0 T + g_0(C) + \varepsilon_0$ and $T = m_0(C) + \varepsilon_2$
- There are also instrumental variable variants of both IRM and PLR

Reconciling these slides notation with the paper

- These slides use a somewhat simpler/accounting-oriented notation.
- Reconciliation from slides to papers:
  - T is D
  - C is X
  - ε_0 is U or ζ depending on the paper
  - ε_1 is V

Implementing DoubleML

Walking through an implementation of DoubleML

> Problem: How does 401k participation impact wealth?

- This problem is walked through in Chernozhukov et al. (2017 AER, Web Appendix)
  - The R code for the AER paper is available from AER as well
    - Quite clean code at that!
- We will implement this in python using the `DoubleML` library
  - Which Chernozhukov was involved in the development of

Importing the data

- Conveniently, the data is available from the DoubleML package

```python
# Grab the dataset
import doubleml.datasets
df = dml.datasets.fetch_401K('DataFrame')
df
```

```
##          nifa  net_tfa        tw  age      inc  fsize  educ  db  marr  \
## 0         0.0      0.0    4500.0   47   6765.0      2     8   0     0
## 1      6215.0   1015.0   22390.0   36  28452.0      1    16   0     0
## 2         0.0  -2000.0   -2000.0   37   3300.0      6    12   1     0
## 3     15000.0  15000.0  155000.0   58  52590.0      2    16   0     1
## 4         0.0      0.0   58000.0   32  21804.0      1    11   0     0
## ...       ...      ...       ...  ...      ...    ...   ...  ..   ...
## 9910  98498.0  98858.0  157858.0   52  73920.0      1    16   1     0
## 9911    287.0   6230.0   15730.0   41  42927.0      4    14   0     1
## 9912     99.0   6099.0    7406.0   40  23619.0      2    16   1     0
## 9913      0.0    -32.0    2468.0   47  14280.0      4     6   1     0
## 9914   4000.0   5000.0    8857.0   33  11112.0      1    14   0     0
##
##       twoearn  e401  p401  pira  hown
## 0           0     0     0     0     1
## 1           0     0     0     0     1
## 2           0     0     0     0     0
## 3           1     0     0     0     1
## 4           0     0     0     0     1
## ...       ...   ...   ...   ...   ...
## 9910        0     1     1     0     1
```

Using your own data

- We can also do this manually, by importing the Stata file from AER
- We then need to prep the data into the format `DoubleML` expects
  - This is fairly straightforward, just defining our Y, treatment, and control variables

```python
df = pd.read_stata('../../Data/S5_sipp1991.dta')

y = 'net_tfa'
treat = 'e401'
controls = [x for x in df.columns.tolist() if x not in [y, treat]]

df_dml = dml.DoubleMLData(df, y_col=y, d_cols=treat, x_cols=controls)
```

What is the data format used by DoubleML?

```
print(df_dml)
```

```
## ================== DoubleMLData Object ==================
##
## ------------------ Data summary      ------------------
## Outcome variable: net_tfa
## Treatment variable(s): ['e401']
## Covariates: ['nifa', 'tw', 'age', 'inc', 'fsize', 'educ', 'db', 'marr', 'twoearn', 'p401', 'pira', 'hown']
## Instrument variable(s): None
## No. Observations: 9915
##
## ------------------ DataFrame info    ------------------
## <class 'pandas.core.frame.DataFrame'>
## Int64Index: 9915 entries, 0 to 9914
## Columns: 14 entries, nifa to hown
## dtypes: float32(4), int8(10)
## memory usage: 329.2 KB
```

- Pandas dataframe
- A pre-specified outcome variable
- One or more treatment indicators
- One or more controls
- Optional instruments

Set up the Nuisance functions

- Recall that there are two functions, m_0 and g_0 that need to be solved for this method
- We can specify any form for these that we want, so long as they are consistent with Scikit-learn

g_0: *Continuous GBM*

```python
g_0 = GradientBoostingRegressor(
  loss='ls',
  learning_rate=0.01,
  n_estimators=1000,
  subsample=0.5,
  max_depth=2
  )
```

m_0: *Binary GBM*

```python
m_0 = GradientBoostingClassifier(
  loss='exponential',
  learning_rate=0.01,
  n_estimators=1000,
  subsample=0.5,
  max_depth=2
  )
```

Run the DML model: Average Treatment Effects

```python
# Fix the random number generator for replicability
np.random.seed(1234)
# Run the model
dml_model_irm = dml.DoubleMLIRM(df_dml, g_0, m_0)
# Output the model's findings
print(dml_model_irm.fit())
```

```
## ================== DoubleMLIRM Object ==================
##
## ------------------ Data summary      ------------------
## Outcome variable: net_tfa
## Treatment variable(s): ['e401']
## Covariates: ['nifa', 'tw', 'age', 'inc', 'fsize', 'educ', 'db', 'marr', 'twoearn', 'p401', 'pira', 'hown']
## Instrument variable(s): None
## No. Observations: 9915
##
## ------------------ Score & algorithm ------------------
## Score function: ATE
## DML algorithm: dml2
##
## ------------------ Resampling        ------------------
## No. folds: 5
## No. repeated sample splits: 1
## Apply cross-fitting: True
##
## ------------------ Fit summary       ------------------
##             coef     std err         t          P>|t|        2.5 %        97.5 %
## e401  3320.43343  383.604082  8.655887  4.890947e-18  2568.583245  4072.283614
```

Run the DML model: ATTE

- ATTE: Average Treatment Effects of the Treated

```python
# Run the model
dml_model_irm_ATTE = dml.DoubleMLIRM(df_dml, g_0, m_0, score='ATTE')
# Output the model's findings
print(dml_model_irm_ATTE.fit())
```

```
## ================== DoubleMLIRM Object ==================
##
## ------------------ Data summary      ------------------
## Outcome variable: net_tfa
## Treatment variable(s): ['e401']
## Covariates: ['nifa', 'tw', 'age', 'inc', 'fsize', 'educ', 'db', 'marr', 'twoearn', 'p401', 'pira', 'hown']
## Instrument variable(s): None
## No. Observations: 9915
##
## ------------------ Score & algorithm ------------------
## Score function: ATTE
## DML algorithm: dml2
##
## ------------------ Resampling        ------------------
## No. folds: 5
## No. repeated sample splits: 1
## Apply cross-fitting: True
##
## ------------------ Fit summary       ------------------
##               coef     std err          t          P>|t|         2.5 %         97.5 %
## e401  10081.312662  392.074708  25.712734  8.421563e-146  9312.860354   10849.764969
```

Other twists on the model

1. Change the machine learning backend
   - Our models used `dml2`
   - You can switch to `dml1` using `dml_procedure='dml1'`
   - `dml1` follows the math in these slides
     - Solve for a condition equal to zero for each model, and then average the estimators
     - `dml2` solves the for the average of the condition being equal to zero overall
2. Run multiple iterations of the model
   - The paper uses 100 iterations, emulate this by adding `n_rep=100`
3. Change the machine learning models fed to the DoubleML model
   - An example of using “Histogram-based Gradient Boosting” is in the Jupyter notebook
     - This is a much faster GBM-like model

Addendum: Using R

- The doubleML package is available in R as well

Conclusion

Wrap-up

SHAP

- A flexible model for understanding how other models use data
- Many visualization tools for different situations:
  - Individual observations
  - Individual measures
  - Aggregations over all observations
  - Importance plots

DoubleML

- Leveraging the nonparametric nature of ML models to improve causality
- Easy to gauge ATE and ATTE
- Extendable to instrumental variable problems

Packages used for these slides

Python

- doubleml
- numpy
- pandas
- scikit-learn
- shap
- xgboost

R

- kableExtra
- knitr
- reticulate
- revealjs

References

- Blevins, Cameron, and Lincoln Mullen. “Jane, John… Leslie? A Historical Method for Algorithmic Gender Prediction.” DHQ: Digital Humanities Quarterly 9, no. 3 (2015).
- Chernozhukov, Victor, Denis Chetverikov, Mert Demirer, Esther Duflo, Christian Hansen, and Whitney Newey. “Double/debiased/Neyman machine learning of treatment effects.” American Economic Review 107, no. 5 (2017): 261-65.
- Chernozhukov, Victor, Denis Chetverikov, Mert Demirer, Esther Duflo, Christian Hansen, Whitney Newey, and James Robins. “Double/debiased machine learning for treatment and structural parameters.” (2018): C1-C68.
- Lundberg, Scott, and Su-In Lee. “A unified approach to interpreting model predictions.” arXiv preprint, arXiv:1705.07874 (2017).
- Shapley, Lloyd. “A value for n-person Games.” Ann. Math. Study 28, Contributions to the Theory of Games, ed. by HW Kuhn, and AW Tucker (1953): 307-317.
- Wich, Maximilian, Jan Bauer, and Georg Groh. “Impact of politically biased data on hate speech classification.” In Proceedings of the Fourth Workshop on Online Abuse and Harms, pp. 54-64. 2020.

Custom code

```python
# Fully worked out DoubleML model using Histogram-based GBM

from sklearn.experimental import enable_hist_gradient_boosting  # noqa
from sklearn.ensemble import HistGradientBoostingClassifier, HistGradientBoostingRegressor

# set up the data
df = pd.read_stata('../Data/S5_sipp1991.dta')

y = 'net_tfa'
treat = 'e401'
controls = [x for x in df.columns.tolist() if x not in [y, treat]]

df_dml3 = dml.DoubleMLData(df, y_col=y, d_cols=treat, x_cols=controls)

#set up the nonparametric nuisance functions
g_0 = HistGradientBoostingRegressor(loss='least_squares',
                                    learning_rate=0.01,
                                    max_iter=1000,
                                    max_depth=2,
                                    early_stopping=False
                                   )
m_0 = HistGradientBoostingClassifier(loss='binary_crossentropy',
                                     learning_rate=0.01,
                                     max_iter=1000,
                                     max_depth=2,
                                     early_stopping=False
                                    )


np.random.seed(1234)
dml_model_ex_irm = dml.DoubleMLIRM(df_dml, g_0, m_0, n_folds=5, n_rep=100)
print(dml_model_ex_irm.fit())
```