Mudcard answers

- **How do we compare different methods to measure feature importance? Such as the random shuffle today, MI score, and to see feature weights that we did in one of the homework.**
  - I'm not sure what you mean by comparing different methods
  - Can you elaborate and post on Ed discussion?
- **Some articles say XGboost is the improed version of gradient boosting or randomforests. Is this true, or are there scenarios where randomforests would be preferred over XGBoost?**
  - yes, that's right
  - random forest is the lowest in terms of complexity or improvements, then gradient boosting, and then XGBoost
  - as I said many times, you need to try as many ML algorithms as you can so you could use both
  - there is no preferred ML algorithm because we do not know beforehand which will be the most predictive on your dataset
  - random forest can sometimes be more accurate than XGBoost for example
- **I am sort of confused by the feature importance in the linear regression coefficients. What's the difference between the scaled and not scaled version?**
  - if the features are not scaled, the coefficients are determined by two things:
    - the average feature value
    - the actual importance of the feature
  - e.g., if a feautre average is large but the target variable is small, the coefficient needs to be small to bring the feature value down to the target variable's average
  - this ambiguity is removed when all features have the same mean and stdev
- **Is it worth always using the standard scaler on categorical/ordinal features after they've been preprocessed for the first time? So that if linear/logistic regression models work best we can use coefficients for feature importance?**
  - yes
  - always try linear models first so scale all features
- **In the code from class you use something called ,pickle. I looked up the documentation for it and still dont understand what it does can you explain it further?**
  - it is one way to save python objects to a file so you can use them later

Local feature importance metrics

By the end of this module, you will be able to

- Describe motivation behind local feature importance metrics
- Apply SHAP
- Describe LIME

Local feature importance metrics

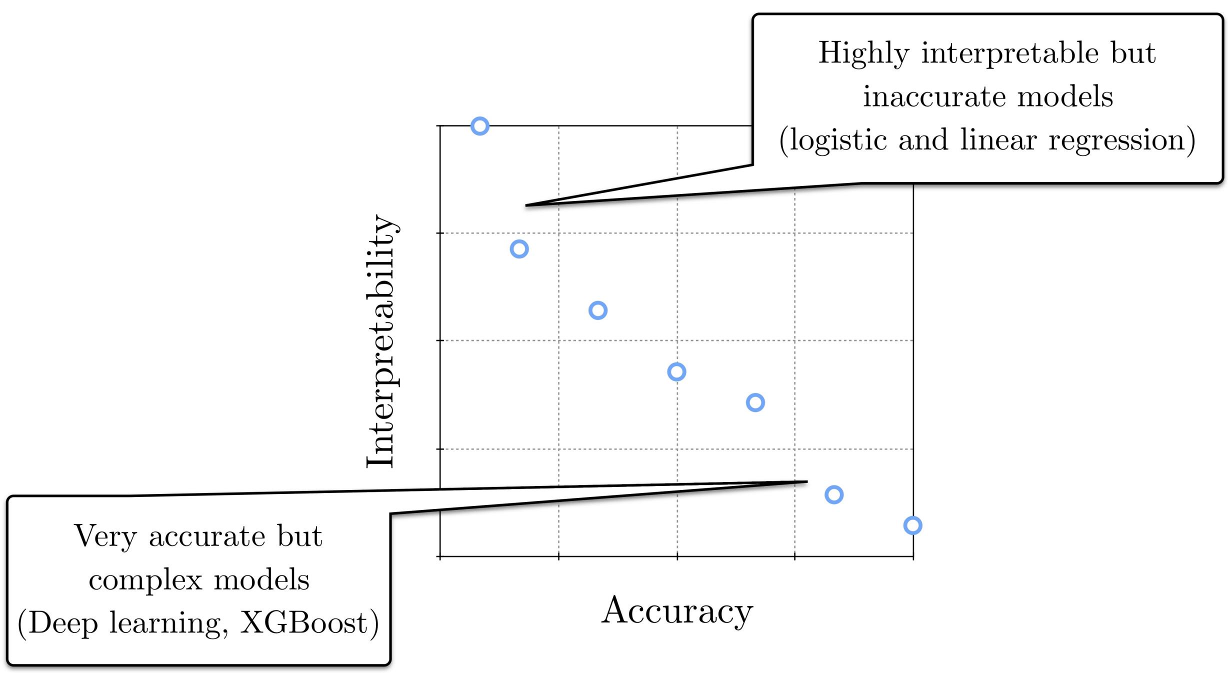
By the end of this module, you will be able to

- **Describe motivation behind local feature importance metrics**
- Apply SHAP
- Describe LIME

Motivation

- can we trust the model?
    - global feeature importance: does the model make predictions based on reasonable features?
    - local feature importance: can we trust the model's prediction for one specific data point?
- global feature importance is often not enough especially when you work with human data
    - medical: the doctor needs to be able to explain the reasoning behind the model prediction to the patient
    - finance: customer wants to know why they were declined a loan/mortgage/credit card/etc

Motivation



- local feature importance improves the interpretability of complex models
- check out this page for a good example

Local feature importance metrics

By the end of this module, you will be able to

- Describe motivation behind local feature importance metrics
- **Apply SHAP**
- Describe LIME

SHAP values

- one way to calculate local feature importances
- it is based on Shapely values from game theory
- read more here, here, and here

Cooperative game theory

- A set of *m* players in a coalition generate a surplus.
- Some players contribute more to the coalition than others (different bargaining powers).
- How important is each player to the coalition?
- How should the surplus be divided fairly amongst the players?

Cooperative game theory applied to feature attribution

- A set of *m* features in a model generate a prediction.
- Some features contribute more to the model than others (different predictive powers).
- How important is each feature to the model?
- How should the prediction be divided amongst the features?

How is it calculated?

$\Phi_i = \sum_{S\subseteq M\setminus i} \frac{|S|!(M - |S| - 1)!} {M!} [f_x(S\cup i) - f_x(S)]$

- Φ_i - the contribution of feature i
- M - the number of features
- S - a set of features excluding i, a vector of 0s and 1s (0 if a feature is missing)
- $|S|$ - the number of features in S
- $f_x(S)$ - the prediction of the model with features S

How is it calculated?

$\Phi_i = \sum_{S\subseteq M\setminus i} \color{blue}{\frac{|S|!(M - |S| - 1)!} {M!}}\color{red}{[f_x(S\cup i) - f_x(S)]}$

- the difference feature i makes in the prediction:
    - $f_x(S\cup i)$ - the prediction with feature i
    - $f_x(S)$ - the prediction without feature i

- loop through all possible ways a set of S features can be selected from the M features excluding i
- weight the contribution based on how many ways we can select $|S|$ features

In [1]:

```
import numpy as np
import pandas as pd
import xgboost
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder
import matplotlib.pylab as plt

df = pd.read_csv('data/adult_data.csv')
label = 'gross-income'
y = LabelEncoder().fit_transform(df[label])
df.drop(columns=[label],inplace=True)
X = df
ftr_names = X.columns
print(X.head())
print(y)
```

```
   age          workclass  fnlwgt  education  education-num  \
0   39          State-gov   77516  Bachelors             13
1   50   Self-emp-not-inc   83311  Bachelors             13
2   38            Private  215646    HS-grad              9
3   53            Private  234721       11th              7
4   28            Private  338409  Bachelors             13

        marital-status         occupation   relationship   race     sex  \
0        Never-married       Adm-clerical  Not-in-family  White    Male
1   Married-civ-spouse    Exec-managerial        Husband  White    Male
2             Divorced  Handlers-cleaners  Not-in-family  White    Male
3   Married-civ-spouse  Handlers-cleaners        Husband  Black    Male
4   Married-civ-spouse     Prof-specialty           Wife  Black  Female

   capital-gain  capital-loss  hours-per-week  native-country
0          2174             0              40   United-States
1             0             0              13   United-States
2             0             0              40   United-States
3             0             0              40   United-States
4             0             0              40            Cuba
[0 0 0 ... 0 0 1]
```

In [2]:

```
def ML_pipeline_kfold(X,y,random_state,n_folds):
    # create a test set
    X_other, X_test, y_other, y_test = train_test_split(X, y, test_size=0.2, ran
    # splitter for _other
    kf = StratifiedKFold(n_splits=n_folds,shuffle=True,random_state=random_state
    # create the pipeline: preprocessor + supervised ML method
    cat_ftrs = ['workclass','education','marital-status','occupation','relations
    cont_ftrs = ['age','fnlwgt','education-num','capital-gain','capital-loss','h
```

```
    # one-hot encoder
    categorical_transformer = Pipeline(steps=[
        ('onehot', OneHotEncoder(sparse=False,handle_unknown='ignore'))])
    # standard scaler
    numeric_transformer = Pipeline(steps=[
        ('scaler', StandardScaler())])
    preprocessor = ColumnTransformer(
        transformers=[
            ('num', numeric_transformer, cont_ftrs),
            ('cat', categorical_transformer, cat_ftrs)])
    pipe = make_pipeline(preprocessor,RandomForestClassifier(n_estimators =  100
    # the parameter(s) we want to tune
    param_grid = {'randomforestclassifier__max_depth': [10,30,100,300],
                  'randomforestclassifier__min_samples_split': [16, 32, 64, 128]
    # prepare gridsearch
    grid = GridSearchCV(pipe, param_grid=param_grid,cv=kf, return_train_score =
    # do kfold CV on _other
    grid.fit(X_other, y_other)
    feature_names = cont_ftrs + \
                list(grid.best_estimator_[0].named_transformers_['cat'][0].get_f
    return grid, np.array(feature_names), X_test, y_test
```

In [3]:

```
grid, feature_names, X_test, y_test = ML_pipeline_kfold(X,y,42,4)
print(grid.best_score_)
print(grid.score(X_test,y_test))
print(grid.best_params_)
```

```
Fitting 4 folds for each of 16 candidates, totalling 64 fits
0.862906941031941
0.8667280822969445
{'randomforestclassifier__max_depth': 100, 'randomforestclassifier__min_samples_
split': 64}
```

In [4]:

```
import shap
shap.initjs() # required for visualizations later on
# create the explainer object with the random forest model
explainer = shap.TreeExplainer(grid.best_estimator_[1])
# transform the test set
X_test_transformed = grid.best_estimator_[0].transform(X_test)
print(np.shape(X_test_transformed))
# calculate shap values on the first 1000 points in the test
shap_values = explainer.shap_values(X_test_transformed[:1000])
print(np.shape(shap_values))
```

```
(6513, 108)
(2, 1000, 108)
```

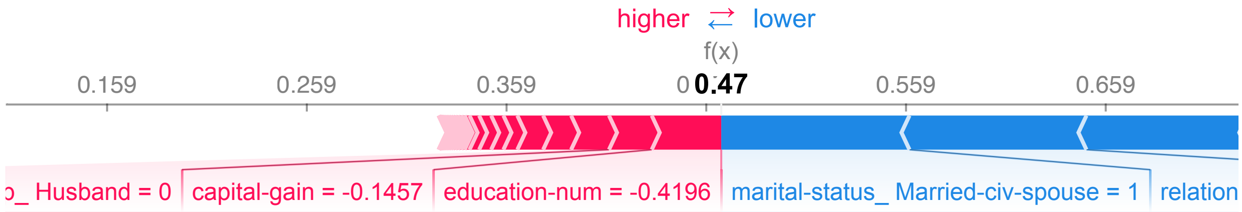
Explain a point

In [5]:

```
index = 1 # the index of the point to explain
print(explainer.expected_value[0]) # we explain class 0 predictions
shap.force_plot(explainer.expected_value[0], shap_values[0][index,:], features =
```

```
0.7589753531941029
```

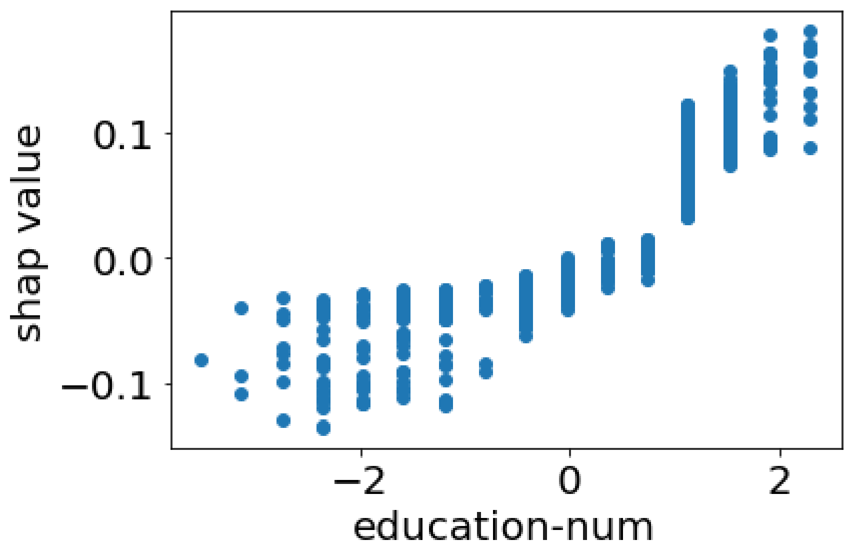
Out[5]:



Feature value vs. shap value

In [6]:

```
import matplotlib
matplotlib.rcParams.update({'font.size': 20})
ftr = 'education-num'
indx = np.argwhere(feature_names=='education-num')
plt.scatter(X_test_transformed[:1000,indx],shap_values[1][:,indx])
plt.ylabel('shap value')
plt.xlabel(ftr)
plt.show()
```

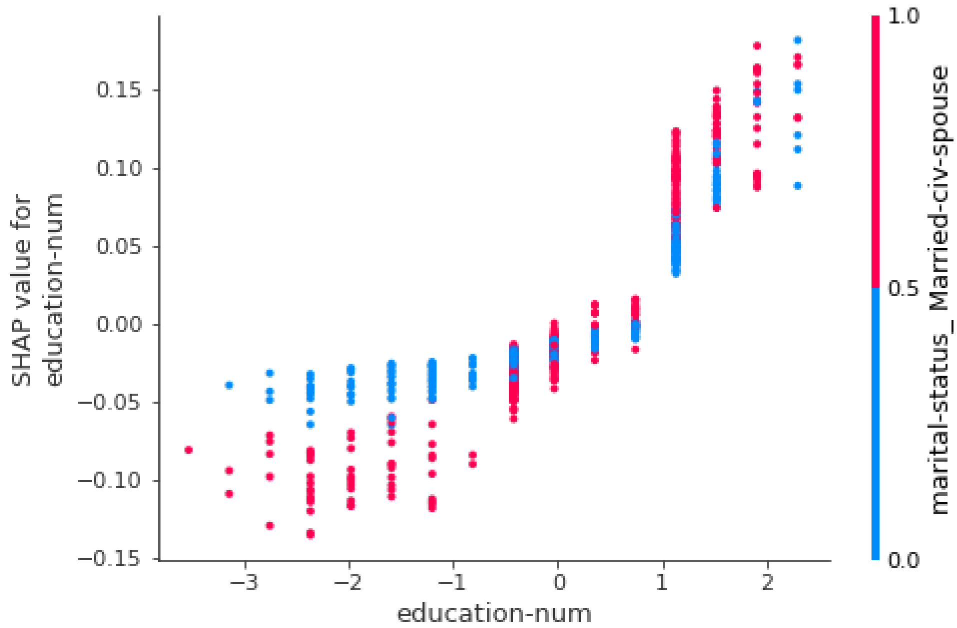


Dependence plot

In [7]:

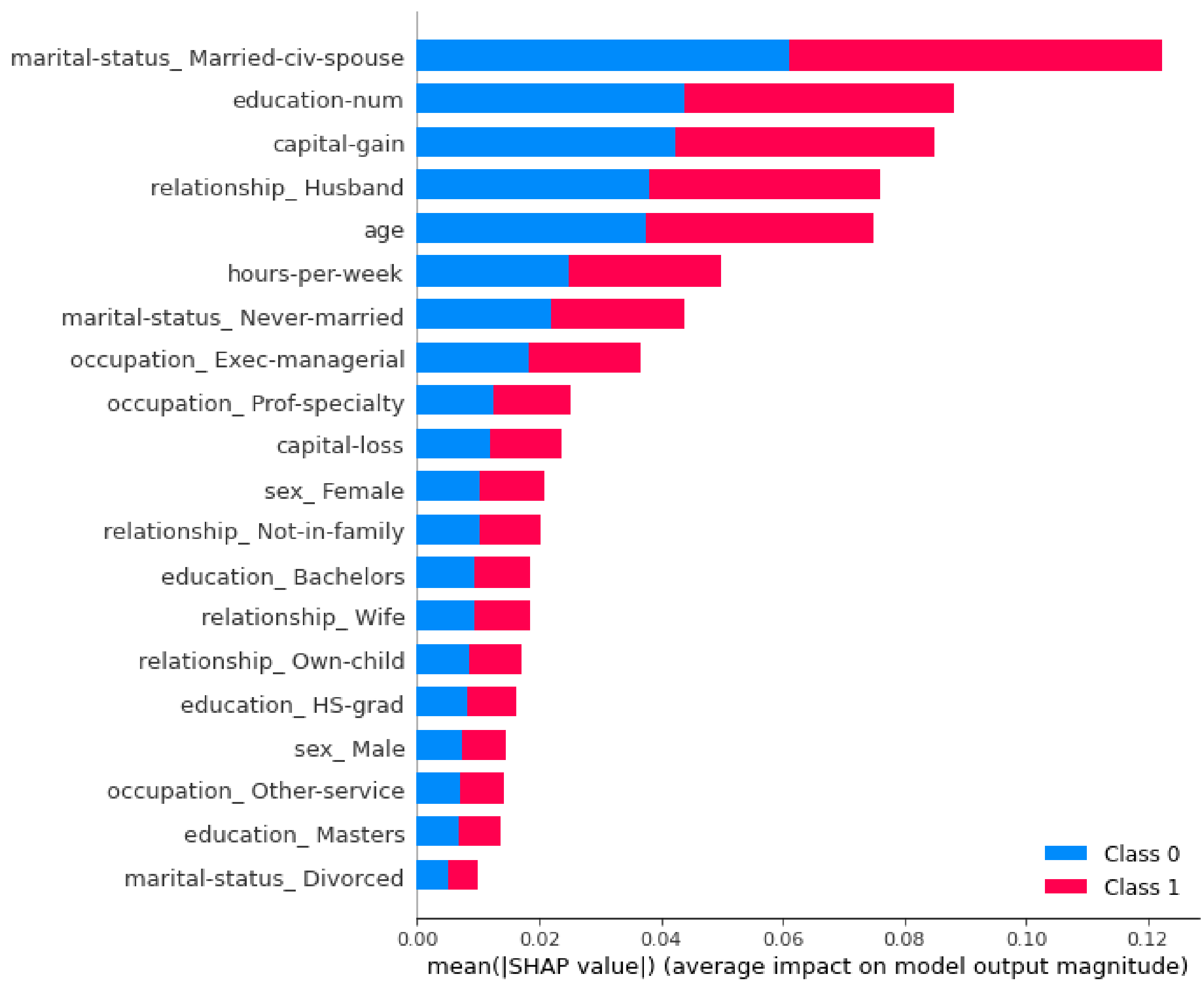
```
shap.dependence_plot(ftr, shap_values[1], X_test_transformed[:1000], feature_nam
```

```
Passing parameters norm and vmin/vmax simultaneously is deprecated since 3.3 and
will become an error two minor releases later. Please pass vmin/vmax directly to
the norm when creating it.
```



It can also be used for global feature importance

In [8]:
```
shap.summary_plot(shap_values, X_test_transformed[:1000],feature_names = feature
```



SHAP cons

- it can be numerically expensive
  - an efficient shap method was developed for trees, see here
- how to estimate $f_x(S)$?
  - this is not trivial because models cannot change the number of features they use
  - usually the values of the dropped features are replaced with the mean or 0
  - this is approximate but no one came up with a better way

Local feature importance metrics

By the end of this module, you will be able to

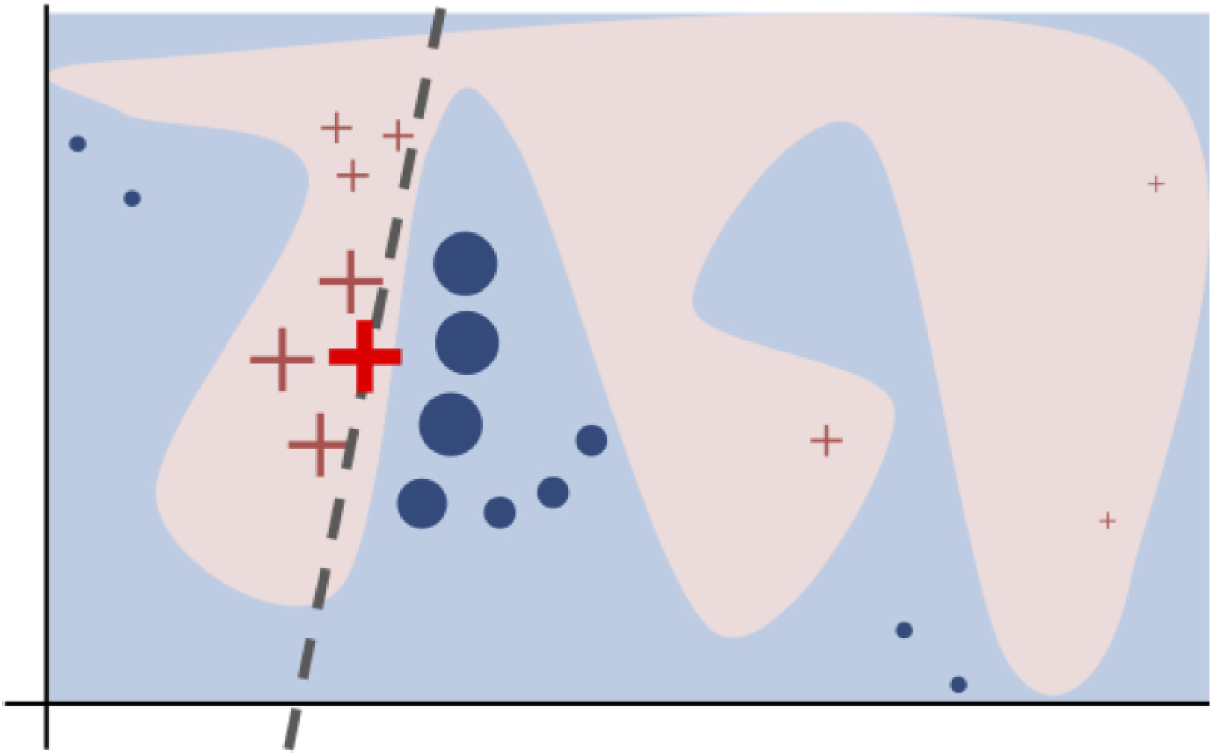
- Describe motivation behind local feature importance metrics
- Apply SHAP
- **Describe LIME**

Locally Interpretable Model-agnostic Explanations

- read about it here, here, and here
- classification and regression models can be complex and explaining the whole model is challenging
- let's focus on one point at a time
- generate an interpretable model (linear regression) in the local neighborhood of that one point
- study the coefficients of that model

LIME steps:

- select a data point you want to explain
- generate random samples
- weight the samples based on their distance from the data point of interest (exponential kernel)
- train a linear regression model (usually lasso) using the weighted samples
- study the local model around the point



Cons, the devil is in the details

- the random samples are not taken around the data point of interest
- how to define the half width of the kernel?
    - the explanation can be very sensitive to the kernel width
    - there is no good way to define/measure what a good kernel width is
- the distance measure treats each feature equally which can be problematic

Now you can

- Describe motivation behind local feature importance metrics
- Apply SHAP
- Describe LIME