Mud card

- **If one feature only has, for example, one value, does it make sense to do permutation importance test?**
  - indeed it does not. if a feature has a single value, you should drop it because no predictive power can be extracted from it.
  - if your dataset has only one feature (but multiple values), permutation importance doesn't make sense because all the predictive power above baseline comes from that one feature
- **When we use gridsearchcv, what value should we use for the iid parameter?**
  - from the manual: `If True, return the average score across folds, weighted by the number of samples in each test set. In this case, the data is assumed to be identically distributed across the folds, and the loss minimized is the total loss per sample, and not the mean loss across the folds. If False, return the average score across folds.`
  - if you data is iid, set it to true

Local feature importance metrics

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

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

Quick review of global feature importances

- the goal of feature importance metrics is to inspect your model and make sure the predictions are reasonable
  - opening up the black box a bit
- some methods come with built-in metrics
  - linear and logistic regression: coefficient can be used to rank features only if **all features are scaled**!
  - linear SVM only: `.coef_`
  - random forest: `.feature_importances_`
  - XGBoost: 5 different metrics are implemented
- permutation feature importance
  - model-agnostic, it can be used with any supervised ML method
  - computationally efficient because the model doesn't need to be retrained
  - cons: strongly correlated features, feature interactions

Local feature importance metrics

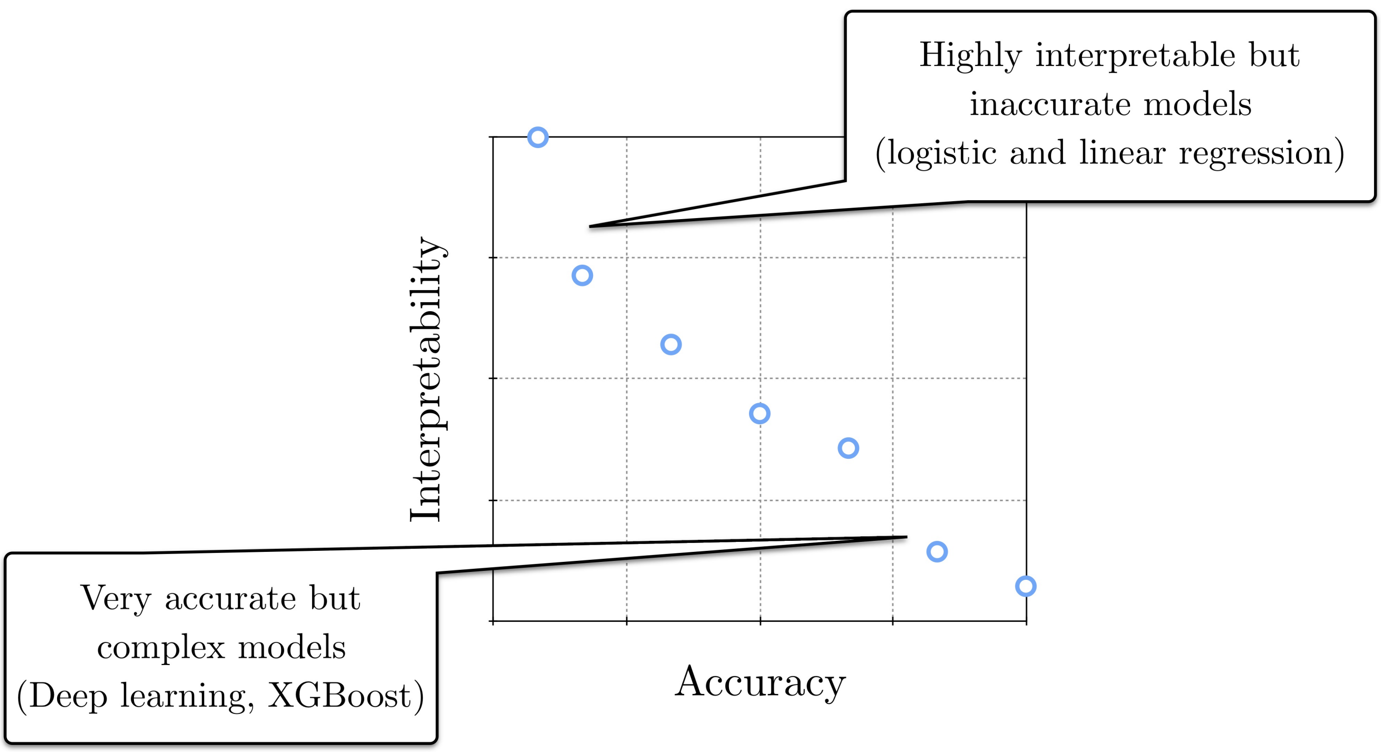
By the end of this lecture, you will be able to

- **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 (http://yann.lecun.com/exdb/mnist/)](http://yann.lecun.com/exdb/mnist/) for a good example

Local feature importance metrics

By the end of this lecture, 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
- unfortunately the package is not installed on the hub
    - you can still use it on your laptop
    - add `- shap=0.31` to the data1030.yml file
- it is based on Shapely values from game theory
- read more here (https://arxiv.org/abs/1802.03888), here (https://github.com/slundberg/shap), and here (https://christophm.github.io/interpretable-ml-book/shap.html)

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?

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

- 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  Fe
male   

   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, random_state = random_state)
    # splitter for _other
    kf = StratifiedKFold(n_splits=n_folds,shuffle=True,random_state=rand
om_state)
    # create the pipeline: preprocessor + supervised ML method
    cat_ftrs = ['workclass','education','marital-status','occupation','r
elationship','race','sex','native-country']
    cont_ftrs = ['age','fnlwgt','education-num','capital-gain','capital-
loss','hours-per-week']
    # 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_estimator
s =  100,random_state=random_state))
    # 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 = True,n_jobs=-1,verbose=10)
    # 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_feature_names(cat_ftrs))
    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

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    5.0s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:    8.4s
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed:    8.6s
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   16.0s
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   20.0s
[Parallel(n_jobs=-1)]: Done  45 tasks      | elapsed:   23.8s
[Parallel(n_jobs=-1)]: Done  56 out of  64 | elapsed:   27.9s remainin
g:    4.0s
[Parallel(n_jobs=-1)]: Done  64 out of  64 | elapsed:   31.6s finished

0.8619087837837838
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))
```

js

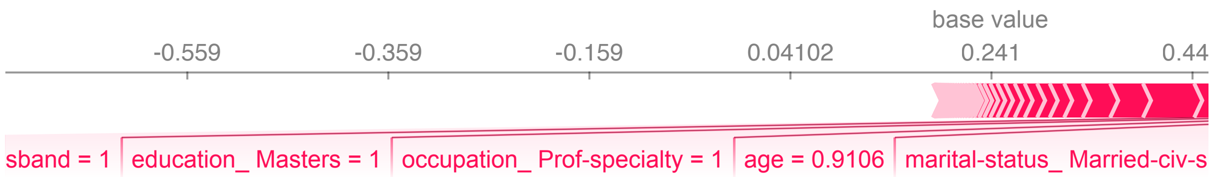
```
(6513, 108)
(2, 1000, 108)
```

Explain a point

In [5]:
```
index = 5 # the index of the point to explain
print(explainer.expected_value[1]) # we explain class 1 predictions
shap.force_plot(explainer.expected_value[1], shap_values[1][index,:], fe
atures = X_test_transformed[index,:],feature_names = feature_names)
```

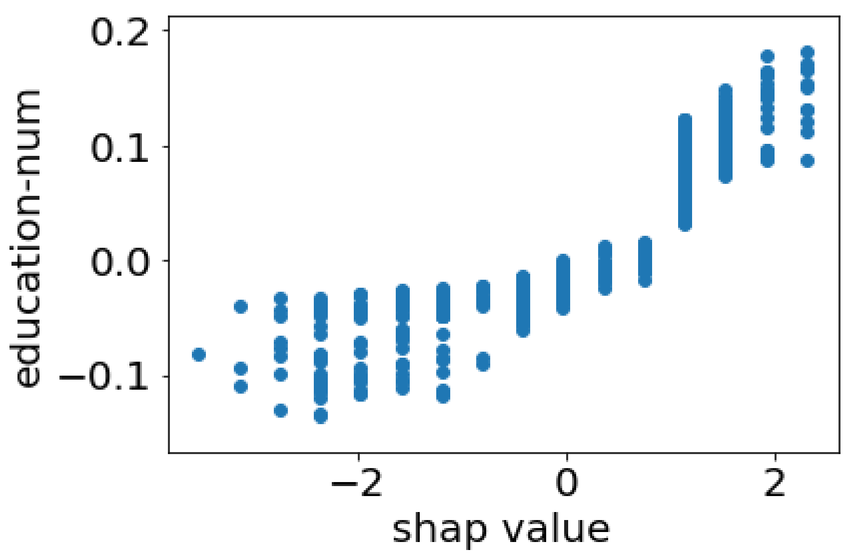
```
0.24102464680589678
```

Out[5]:



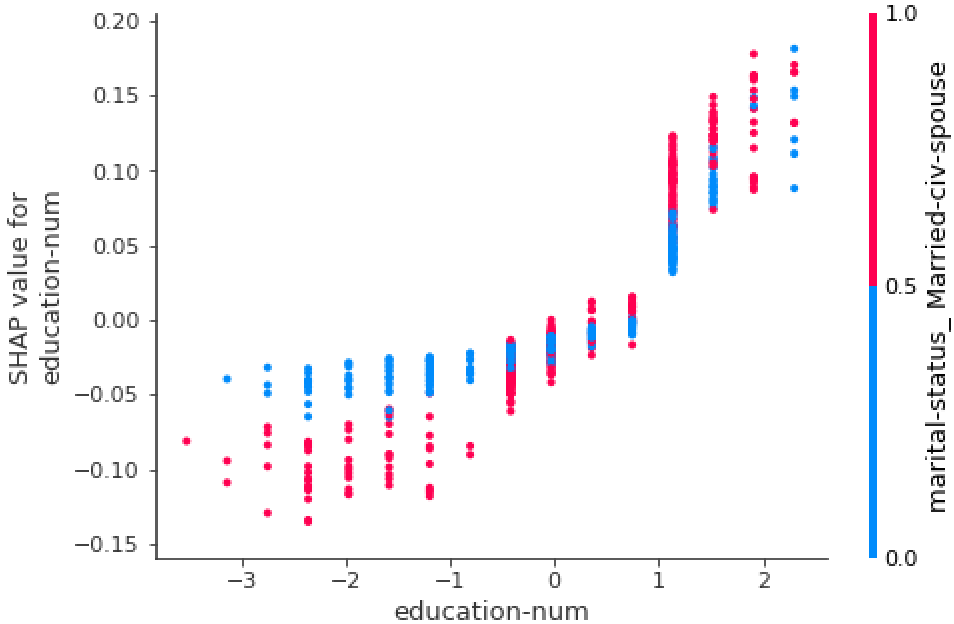
Feature value vs. shap value

In [13]:
```
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.xlabel('shap value')
plt.ylabel(ftr)
plt.show()
```



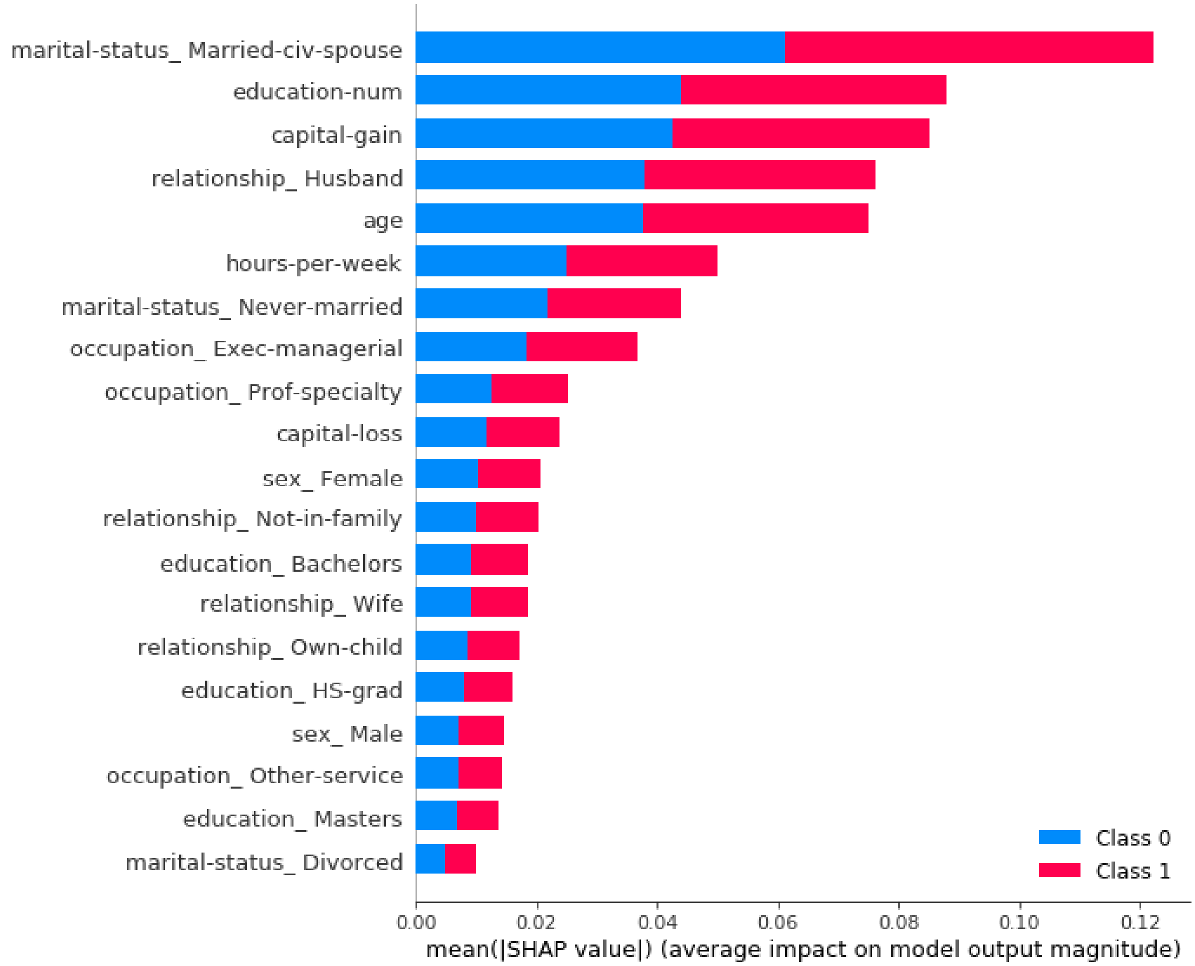
Dependence plot

In [7]:
```
shap.dependence_plot(ftr, shap_values[1], X_test_transformed[:1000], fea
ture_names=feature_names)
```



It can also be used for global feature importance

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



SHAP cons

- it can be numerically expensive
  - an efficient shap method was developed for trees, see [here (https://arxiv.org/abs/1905.04610)](https://arxiv.org/abs/1905.04610)
- 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 lecture, you will be able to

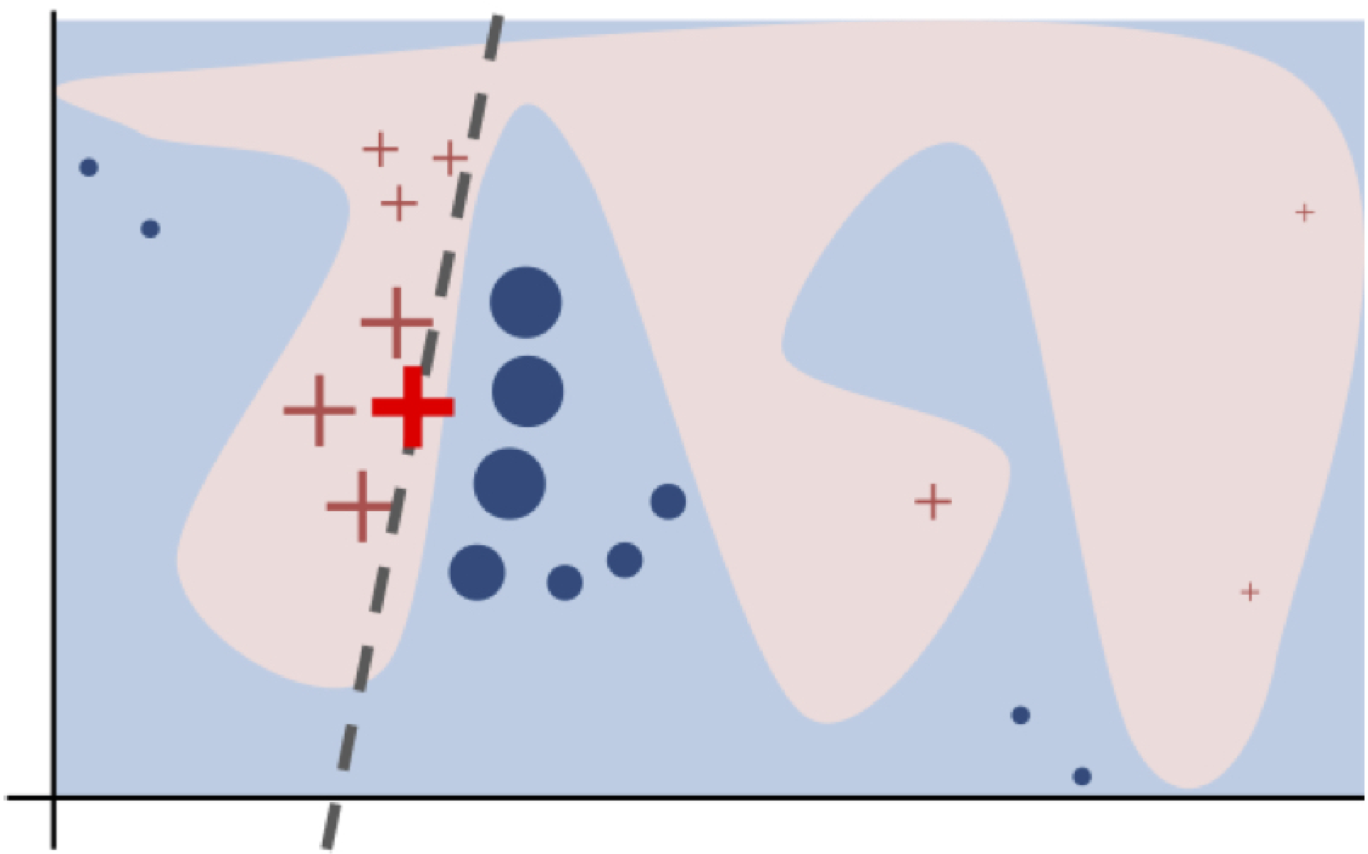
- Describe motivation behind local feature importance metrics
- Apply SHAP
- **Describe LIME**

Locally Interpretable Model-agnostic Explanations

- read about it [here (https://github.com/marcotcr/lime)](https://github.com/marcotcr/lime), [here (https://arxiv.org/abs/1602.04938)](https://arxiv.org/abs/1602.04938), and [here (https://christophm.github.io/interpretable-ml-book/lime.html)](https://christophm.github.io/interpretable-ml-book/lime.html)
- 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