12_feat-importances

June 24, 2022

CPSC 330
Applied Machine Learning

1 Lecture 12: Feature importances

UBC 2022 Summer

Instructor: Mehrdad Oveisi

1.1 Imports

[1]:
```
import os
import string
import sys
from collections import deque

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

sys.path.append("code/.")

import seaborn as sns
from plotting_functions import *
from sklearn import datasets
from sklearn.compose import ColumnTransformer, make_column_transformer
from sklearn.dummy import DummyClassifier, DummyRegressor
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression, Ridge
from sklearn.model_selection import (
    GridSearchCV,
    RandomizedSearchCV,
    cross_val_score,
```

```
    cross_validate,
    train_test_split,
)
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder, StandardScaler
from sklearn.svm import SVC, SVR
from sklearn.tree import DecisionTreeClassifier
from utils import *

%matplotlib inline
```

1.2 Learning outcomes

From this lecture, students are expected to be able to:

- Interpret the coefficients of linear regression for ordinal, one-hot encoded categorical, and scaled numeric features.
- Explain why interpretability is important in ML.
- Use `feature_importances_` attribute of `sklearn` models and interpret its output.
- Use `eli5` to get feature importances of non `sklearn` models and interpret its output.
- Apply SHAP to assess feature importances and interpret model predictions.
- Explain force plot, summary plot, and dependence plot produced with shapely values.

1.3 Data

In this lecture, we'll be using Kaggle House Prices dataset, the dataset we used in lecture 2. As usual, to run this notebook you'll need to download the data. Unzip the data into a subdirectory called `data`. For this dataset, train and test have already been separated. We'll be working with the train portion in this lecture.

[2]:
```
df = pd.read_csv("data/housing-kaggle/train.csv")
train_df, test_df = train_test_split(df, test_size=0.10, random_state=123)
train_df.head()
```

[2]:
```
        Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
302    303          20       RL        118.0    13704   Pave   NaN      IR1
767    768          50       RL         75.0    12508   Pave   NaN      IR1
429    430          20       RL        130.0    11457   Pave   NaN      IR1
1139  1140          30       RL         98.0     8731   Pave   NaN      IR1
558    559          60       RL         57.0    21872   Pave   NaN      IR2

     LandContour Utilities  … PoolArea PoolQC Fence MiscFeature MiscVal  \
302          Lvl    AllPub  …        0    NaN   NaN         NaN       0
767          Lvl    AllPub  …        0    NaN   NaN        Shed    1300
429          Lvl    AllPub  …        0    NaN   NaN         NaN       0
1139         Lvl    AllPub  …        0    NaN   NaN         NaN       0
558          HLS    AllPub  …        0    NaN   NaN         NaN       0
```

```
      MoSold YrSold  SaleType  SaleCondition  SalePrice
302        1   2006        WD         Normal     205000
767        7   2008        WD         Normal     160000
429        3   2009        WD         Normal     175000
1139       5   2007        WD         Normal     144000
558        8   2008        WD         Normal     175000

[5 rows x 81 columns]
```

- The prediction task is predicting `SalePrice` given features related to properties.
- Note that the **target is numeric**, not categorical.

[3]:
```
train_df.shape
```

[3]:
```
(1314, 81)
```

1.3.1 Let's separate `X` and `y`

[4]:
```
X_train = train_df.drop(columns=["SalePrice"])
y_train = train_df["SalePrice"]

X_test = test_df.drop(columns=["SalePrice"])
y_test = test_df["SalePrice"]
```

1.3.2 Let's identify feature types

[5]:
```
drop_features = ["Id"]
numeric_features = [
    "BedroomAbvGr",
    "KitchenAbvGr",
    "LotFrontage",
    "LotArea",
    "OverallQual",
    "OverallCond",
    "YearBuilt",
    "YearRemodAdd",
    "MasVnrArea",
    "BsmtFinSF1",
    "BsmtFinSF2",
    "BsmtUnfSF",
    "TotalBsmtSF",
    "1stFlrSF",
    "2ndFlrSF",
    "LowQualFinSF",
    "GrLivArea",
    "BsmtFullBath",
    "BsmtHalfBath",
```

```
    "FullBath",
    "HalfBath",
    "TotRmsAbvGrd",
    "Fireplaces",
    "GarageYrBlt",
    "GarageCars",
    "GarageArea",
    "WoodDeckSF",
    "OpenPorchSF",
    "EnclosedPorch",
    "3SsnPorch",
    "ScreenPorch",
    "PoolArea",
    "MiscVal",
    "YrSold",
]
```

[6]:
```
ordinal_features_reg = [
    "ExterQual",
    "ExterCond",
    "BsmtQual",
    "BsmtCond",
    "HeatingQC",
    "KitchenQual",
    "FireplaceQu",
    "GarageQual",
    "GarageCond",
    "PoolQC",
]
ordering = [
    "Po",
    "Fa",
    "TA",
    "Gd",
    "Ex",
]  # if N/A it will just impute something, per below
ordering_ordinal_reg = [ordering] * len(ordinal_features_reg)
ordering_ordinal_reg
```

[6]:
```
[['Po', 'Fa', 'TA', 'Gd', 'Ex'],
 ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
 ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
 ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
 ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
 ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
 ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
 ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
```

```
 ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
 ['Po', 'Fa', 'TA', 'Gd', 'Ex']]
```

[7]:
```python
ordinal_features_oth = [
    "BsmtExposure",
    "BsmtFinType1",
    "BsmtFinType2",
    "Functional",
    "Fence",
]
ordering_ordinal_oth = [
    ["NA", "No", "Mn", "Av", "Gd"],
    ["NA", "Unf", "LwQ", "Rec", "BLQ", "ALQ", "GLQ"],
    ["NA", "Unf", "LwQ", "Rec", "BLQ", "ALQ", "GLQ"],
    ["Sal", "Sev", "Maj2", "Maj1", "Mod", "Min2", "Min1", "Typ"],
    ["NA", "MnWw", "GdWo", "MnPrv", "GdPrv"],
]
```

[8]:
```python
categorical_features = list(
    set(X_train.columns)
    - set(numeric_features)
    - set(ordinal_features_reg)
    - set(ordinal_features_oth)
    - set(drop_features)
)
categorical_features
```

[8]:
```
['MasVnrType',
 'Exterior2nd',
 'LandSlope',
 'RoofStyle',
 'Neighborhood',
 'SaleType',
 'Condition2',
 'Exterior1st',
 'SaleCondition',
 'MSZoning',
 'BldgType',
 'RoofMatl',
 'HouseStyle',
 'MiscFeature',
 'MoSold',
 'LotConfig',
 'MSSubClass',
 'Utilities',
 'CentralAir',
 'LotShape',
```

```
 'Street',
 'Foundation',
 'Electrical',
 'GarageFinish',
 'Condition1',
 'Alley',
 'LandContour',
 'Heating',
 'GarageType',
 'PavedDrive']
```

[9]:
```
from sklearn.compose import ColumnTransformer, make_column_transformer

numeric_transformer = make_pipeline(SimpleImputer(strategy="median"),␣
 ↪StandardScaler())
ordinal_transformer_reg = make_pipeline(
    SimpleImputer(strategy="most_frequent"),
    OrdinalEncoder(categories=ordering_ordinal_reg),
)

ordinal_transformer_oth = make_pipeline(
    SimpleImputer(strategy="most_frequent"),
    OrdinalEncoder(categories=ordering_ordinal_oth),
)

categorical_transformer = make_pipeline(
    SimpleImputer(strategy="constant", fill_value="missing"),
    OneHotEncoder(handle_unknown="ignore", sparse=False),
)

preprocessor = make_column_transformer(
    ("drop", drop_features),
    (numeric_transformer, numeric_features),
    (ordinal_transformer_reg, ordinal_features_reg),
    (ordinal_transformer_oth, ordinal_features_oth),
    (categorical_transformer, categorical_features),
)
```

[10]:
```
preprocessor.fit(X_train)
preprocessor.named_transformers_
```

[10]:
```
{'drop': 'drop',
 'pipeline-1': Pipeline(steps=[('simpleimputer',
SimpleImputer(strategy='median')),
                 ('standardscaler', StandardScaler())]),
 'pipeline-2': Pipeline(steps=[('simpleimputer',
SimpleImputer(strategy='most_frequent')),
```

```
                 ('ordinalencoder',
                  OrdinalEncoder(categories=[['Po', 'Fa', 'TA', 'Gd', 'Ex'],
                                             ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
                                             ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
                                             ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
                                             ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
                                             ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
                                             ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
                                             ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
                                             ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
                                             ['Po', 'Fa', 'TA', 'Gd',
'Ex']]))]),
 'pipeline-3': Pipeline(steps=[('simpleimputer',
SimpleImputer(strategy='most_frequent')),
                 ('ordinalencoder',
                  OrdinalEncoder(categories=[['NA', 'No', 'Mn', 'Av', 'Gd'],
                                             ['NA', 'Unf', 'LwQ', 'Rec', 'BLQ',
                                              'ALQ', 'GLQ'],
                                             ['NA', 'Unf', 'LwQ', 'Rec', 'BLQ',
                                              'ALQ', 'GLQ'],
                                             ['Sal', 'Sev', 'Maj2', 'Maj1',
                                              'Mod', 'Min2', 'Min1', 'Typ'],
                                             ['NA', 'MnWw', 'GdWo', 'MnPrv',
                                              'GdPrv']]))]),
 'pipeline-4': Pipeline(steps=[('simpleimputer',
                  SimpleImputer(fill_value='missing', strategy='constant')),
                 ('onehotencoder',
                  OneHotEncoder(handle_unknown='ignore', sparse=False))])}
```

[11]:
```
ohe_columns = list(
    preprocessor.named_transformers_["pipeline-4"]
    .named_steps["onehotencoder"]
    .get_feature_names_out(categorical_features)
)
new_columns = (
    numeric_features + ordinal_features_reg + ordinal_features_oth + ohe_columns
)
```

[12]:
```
X_train_enc = pd.DataFrame(
    preprocessor.transform(X_train), index=X_train.index, columns=new_columns
)
X_train_enc
```

[12]:
```
     BedroomAbvGr  KitchenAbvGr  LotFrontage   LotArea  OverallQual  \
302      0.154795     -0.222647     2.312501  0.381428     0.663680
767      1.372763     -0.222647     0.260890  0.248457    -0.054669
429      0.154795     -0.222647     2.885044  0.131607    -0.054669
```

```
1139     0.154795     -0.222647     1.358264 -0.171468    -0.773017
558      0.154795     -0.222647    -0.597924  1.289541     0.663680
…           …            …            …         …            …
1041     1.372763     -0.222647    -0.025381 -0.127107    -0.054669
1122     0.154795     -0.222647    -0.025381 -0.149788    -1.491366
1346     0.154795     -0.222647    -0.025381  1.168244     0.663680
1406    -1.063173     -0.222647     0.022331 -0.203265    -0.773017
1389     0.154795     -0.222647    -0.454788 -0.475099    -0.054669

      OverallCond  YearBuilt  YearRemodAdd  MasVnrArea  BsmtFinSF1  …  \
302     -0.512408   0.993969      0.840492    0.269972   -0.961498  …
767      1.285467  -1.026793      0.016525   -0.573129    0.476092  …
429     -0.512408   0.563314      0.161931   -0.573129    1.227559  …
1139    -0.512408  -1.689338     -1.679877   -0.573129    0.443419  …
558     -0.512408   0.828332      0.598149   -0.573129    0.354114  …
…            …          …             …           …           …  …
1041     2.184405  -0.165485      0.743555    0.843281   -0.090231  …
1122    -2.310284  -0.496757     -1.389065   -0.573129   -0.961498  …
1346     1.285467  -0.099230      0.888961   -0.573129   -0.314582  …
1406     1.285467   0.033279      1.082835   -0.573129    0.467379  …
1389     0.386530  -0.993666     -1.679877   -0.573129   -0.144686  …

      GarageType_2Types  GarageType_Attchd  GarageType_Basment  \
302                 0.0                1.0                 0.0
767                 0.0                1.0                 0.0
429                 0.0                1.0                 0.0
1139                0.0                0.0                 0.0
558                 0.0                1.0                 0.0
…                   …                  …                  …
1041                0.0                1.0                 0.0
1122                0.0                0.0                 1.0
1346                0.0                1.0                 0.0
1406                0.0                0.0                 0.0
1389                0.0                0.0                 0.0

      GarageType_BuiltIn  GarageType_CarPort  GarageType_Detchd  \
302                  0.0                 0.0                0.0
767                  0.0                 0.0                0.0
429                  0.0                 0.0                0.0
1139                 0.0                 0.0                1.0
558                  0.0                 0.0                0.0
…                    …                  …                 …
1041                 0.0                 0.0                0.0
1122                 0.0                 0.0                0.0
1346                 0.0                 0.0                0.0
1406                 0.0                 0.0                1.0
1389                 0.0                 0.0                1.0
```

```
      GarageType_missing  PavedDrive_N  PavedDrive_P  PavedDrive_Y
302                  0.0           0.0           0.0           1.0
767                  0.0           0.0           0.0           1.0
429                  0.0           0.0           0.0           1.0
1139                 0.0           0.0           0.0           1.0
558                  0.0           0.0           0.0           1.0
…                    …             …             …             …
1041                 0.0           0.0           0.0           1.0
1122                 0.0           0.0           0.0           1.0
1346                 0.0           0.0           0.0           1.0
1406                 0.0           0.0           0.0           1.0
1389                 0.0           0.0           0.0           1.0

[1314 rows x 263 columns]
```

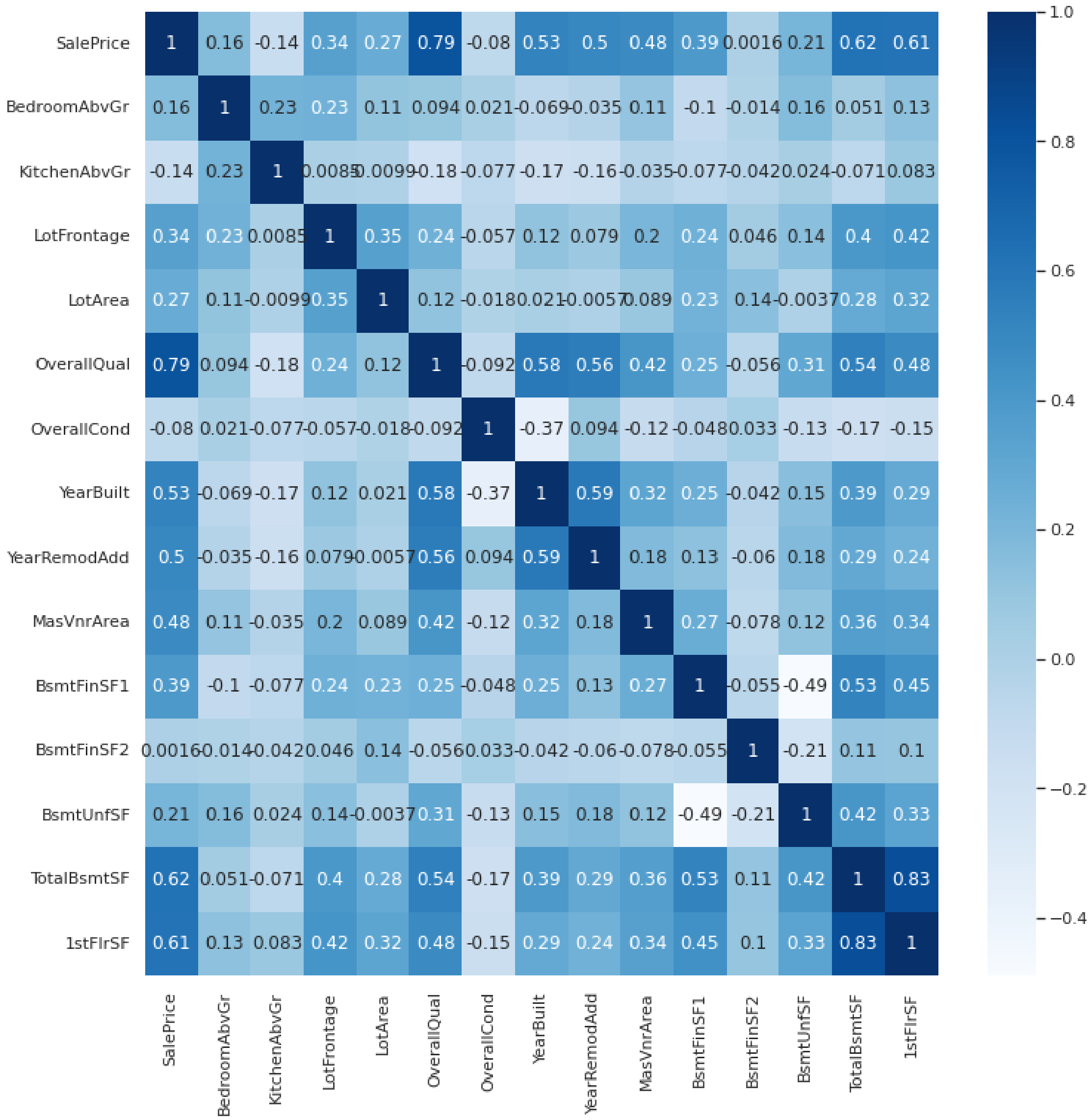
[13]:
```python
X_train_enc.shape
```

[13]:
```
(1314, 263)
```

1.3.3 Feature correlations

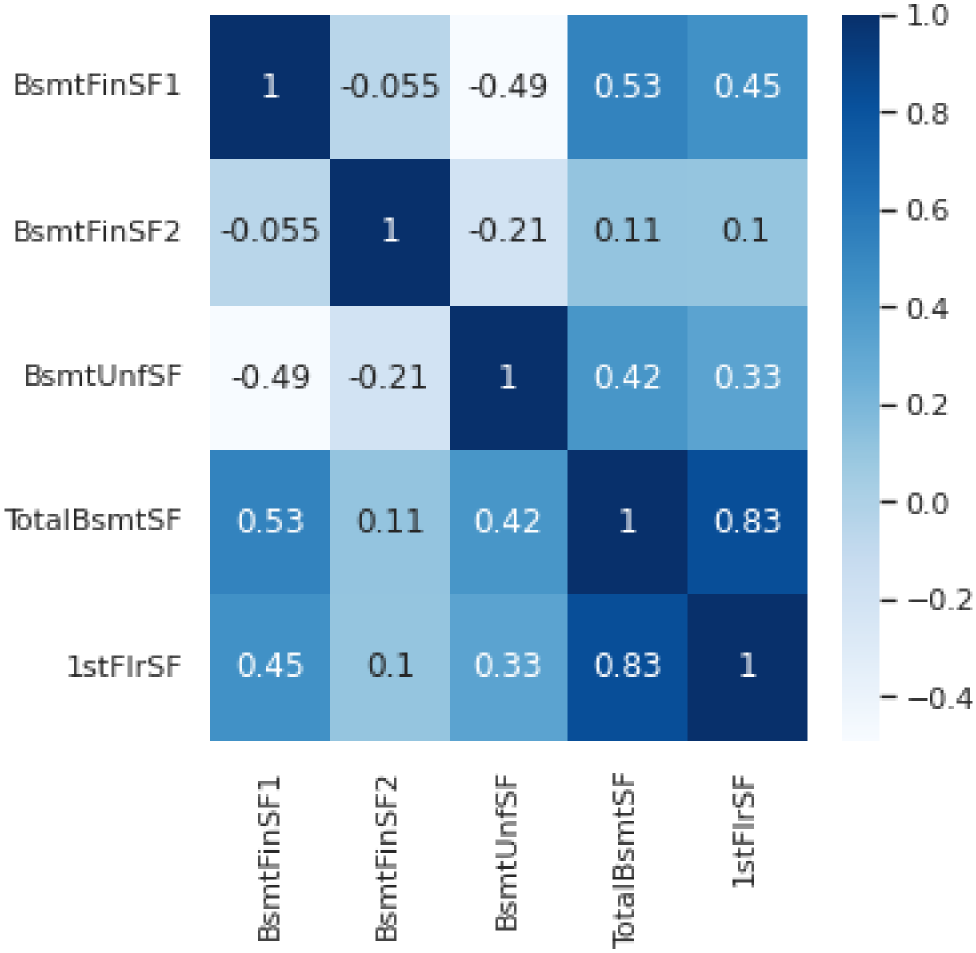
- Let's look at the correlations between various features with other features and the target in our encoded data (first row/column).
- In simple terms here is how you can interpret correlations between two variables X and Y:
  - If Y goes up when X goes up, we say X and Y are **positively correlated**.
  - If Y goes down when X goes up, we say X and Y are **negatively correlated**.
  - If Y is unchanged when X changes, we say X and Y are **uncorrelated**.

[14]:
```python
# Get the pairwise correlations between the first 15 columns (including y_train)
cor = pd.concat((y_train, X_train_enc), axis=1).iloc[:, :15].corr()
plt.figure(figsize=(12, 12))
sns.set(font_scale=1)
sns.heatmap(cor, annot=True, cmap=plt.cm.Blues);
```



- We can immediately see that `SalePrice` is highly correlated with `OverallQual`.
- This is an early hint that `OverallQual` is a useful feature in predicting `SalePrice`.
- However, this approach is **extremely simplistic**.
  - It only looks at **each feature in isolation**.
  - It only looks at **linear associations**:
    * What if `SalePrice` is high when `BsmtFullBath` is 2 or 3, but low when it's 0, 1, or 4? They might seem uncorrelated.

[15]:
```
cor = pd.concat((y_train, X_train_enc), axis=1).iloc[:, 10:15].corr()
plt.figure(figsize=(5, 5))
sns.set(font_scale=1)
sns.heatmap(cor, annot=True, cmap=plt.cm.Blues);
```



- Looking at this diagram also tells us the relationship between features.
  - For example, `1stFlrSF` and `TotalBsmtSF` are highly correlated.
  - Do we need both of them?
  - If our model says `1stFlrSF` is very important and `TotalBsmtSF` is very unimportant, do we trust those values?
  - Maybe `TotalBsmtSF` only "becomes important" if `1stFlrSF` is removed.
  - Sometimes the opposite happens: a feature only becomes important if another feature is *added*.

1.4 Feature importance in linear models

- Like logistic regression, with linear regression we can look at the *coefficients* for each feature.
- Overall idea: predicted price = intercept + $\sum_i$ coefficient i $\times$ feature i

[16]:
```
lr = make_pipeline(preprocessor, Ridge())
lr.fit(X_train, y_train);
```

Let's look at the coefficients.

[17]:
```python
lr_coefs = pd.DataFrame(data=lr[1].coef_, index=new_columns,␣
    ↪columns=["Coefficient"])
lr_coefs.head(20)
```

[17]:

	Coefficient
BedroomAbvGr	-3723.741570
KitchenAbvGr	-4580.204576
LotFrontage	-1578.664421
LotArea	5109.356718
OverallQual	12487.561839
OverallCond	4855.535334
YearBuilt	4226.684842
YearRemodAdd	324.664715
MasVnrArea	5251.325210
BsmtFinSF1	3667.172851
BsmtFinSF2	583.114880
BsmtUnfSF	-1266.614671
TotalBsmtSF	2751.084018
1stFlrSF	6736.788904
2ndFlrSF	13409.901084
LowQualFinSF	-448.424132
GrLivArea	15988.182407
BsmtFullBath	2299.227266
BsmtHalfBath	500.169112
FullBath	2831.811467

1.4.1 Interpreting coefficients of different types of features.

1.4.2 Ordinal features

- The ordinal features are easiest to interpret.

[18]:
```python
print(ordinal_features_reg)
```

```
['ExterQual', 'ExterCond', 'BsmtQual', 'BsmtCond', 'HeatingQC', 'KitchenQual',
'FireplaceQu', 'GarageQual', 'GarageCond', 'PoolQC']
```

[19]:
```python
lr_coefs.loc["ExterQual", "Coefficient"]
```

[19]: 4195.671512467826

- Increasing by one category of exterior quality (e.g. good -> excellent) increases the predicted price by ∼ $4195.
  - Wow, that's a lot!
  - Remember this is just what the model has learned. It doesn't tell us how the world works.

[20]:
```python
one_example = X_test[:1]
```

[21]:
```
one_example["ExterQual"]
```

[21]:
```
147    Gd
Name: ExterQual, dtype: object
```

Let's perturb the example and change `ExterQual` to `Ex`.

[22]:
```
one_example_perturbed = one_example.copy()
one_example_perturbed["ExterQual"] = "Ex"  # Change Gd to Ex
```

[23]:
```
one_example_perturbed
```

[23]:
```
       Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
147  148          60       RL          NaN     9505   Pave   NaN      IR1

    LandContour Utilities  … ScreenPorch PoolArea PoolQC Fence MiscFeature  \
147         Lvl    AllPub  …           0        0    NaN   NaN         NaN

    MiscVal MoSold  YrSold  SaleType  SaleCondition
147       0      5    2010        WD         Normal

[1 rows x 80 columns]
```

[24]:
```
one_example_perturbed["ExterQual"]
```

[24]:
```
147    Ex
Name: ExterQual, dtype: object
```

How does the prediction change after changing `ExterQual` from `Gd` to `Ex`?

[25]:
```
print("Prediction on the original example: ", lr.predict(one_example))
print("Prediction on the perturbed example: ", lr.
 ↪predict(one_example_perturbed))
print(
    "After changing ExterQual from Gd to Ex increased the prediction by: ",
    lr.predict(one_example_perturbed) - lr.predict(one_example),
)
```

```
Prediction on the original example:  [224795.63596802]
Prediction on the perturbed example:  [228991.30748049]
After changing ExterQual from Gd to Ex increased the prediction by:
[4195.67151247]
```

That's exactly the learned coefficient for `ExterQual`!

[26]:
```
lr_coefs.loc["ExterQual", "Coefficient"]
```

[26]:
```
4195.671512467826
```

So our interpretation is correct! - Increasing by one category of exterior quality (e.g. good -> excellent) increases the predicted price by ∼ $4195.

1.4.3 Categorical features

- What about the categorical features?
- We have created a number of columns for each category with OHE and each category gets it's own coefficient.

[27]:
```
print(categorical_features)
```

```
['MasVnrType', 'Exterior2nd', 'LandSlope', 'RoofStyle', 'Neighborhood',
'SaleType', 'Condition2', 'Exterior1st', 'SaleCondition', 'MSZoning',
'BldgType', 'RoofMatl', 'HouseStyle', 'MiscFeature', 'MoSold', 'LotConfig',
'MSSubClass', 'Utilities', 'CentralAir', 'LotShape', 'Street', 'Foundation',
'Electrical', 'GarageFinish', 'Condition1', 'Alley', 'LandContour', 'Heating',
'GarageType', 'PavedDrive']
```

[28]:
```
lr_coefs_landslope = lr_coefs[lr_coefs.index.str.startswith("LandSlope")]
lr_coefs_landslope
```

[28]:
```
               Coefficient
LandSlope_Gtl   457.197456
LandSlope_Mod  7420.208381
LandSlope_Sev -7877.405837
```

- We can talk about switching from one of these categories to another by picking a "reference" category:

[29]:
```
lr_coefs_landslope.loc["LandSlope_Gtl"]
```

[29]:
```
Coefficient    457.197456
Name: LandSlope_Gtl, dtype: float64
```

[30]:
```
lr_coefs_landslope - lr_coefs_landslope.loc["LandSlope_Gtl"]
```

[30]:
```
               Coefficient
LandSlope_Gtl     0.000000
LandSlope_Mod  6963.010925
LandSlope_Sev -8334.603292
```

- If you change the category from `LandSlope_Gtl` to `LandSlope_Mod` the prediction price goes up by ∼ $6963
- If you change the category from `LandSlope_Gtl` to `LandSlope_Sev` the prediction price goes down by ∼ $8334

Note that this might not make sense in the real world but this is what our model decided to learn given this small amount of data.

[31]:
```
lr_coefs.sort_values(by="Coefficient")
```

[31]:

```
                           Coefficient
RoofMatl_ClyTile     -191129.774314
Condition2_PosN      -105552.840565
Heating_OthW          -27260.681308
MSZoning_C (all)      -21990.746193
Exterior1st_ImStucc   -19393.964621
…                                …
PoolQC                 34217.656047
RoofMatl_CompShg       36525.980874
Neighborhood_NridgHt   37532.643270
Neighborhood_StoneBr   39993.978324
RoofMatl_WdShngl       83646.711008

[263 rows x 1 columns]
```

- For example, the above coefficient says that "If the roof is made of clay tile, the predicted price is \$191K less"?
- Do we believe these interpretations??
  - Do we believe this is how the predictions are being **computed**? Yes.
  - Do we believe that this is how the **world works**? No.

[32]:

```python
# We can see all RoofMatl one hot columns:
lr_coefs[lr_coefs.index.str.startswith("RoofMatl")]
```

[32]:

```
                    Coefficient
RoofMatl_ClyTile -191129.774314
RoofMatl_CompShg   36525.980874
RoofMatl_Membran   24537.788381
RoofMatl_Metal     16788.514414
RoofMatl_Roll       8868.963092
RoofMatl_Tar&Grv    7477.664157
RoofMatl_WdShake   13284.152389
RoofMatl_WdShngl   83646.711008
```

Note If you did `drop='first'` (we didn't) then you already have a reference class, and all the values are with respect to that one. The interpretation depends on whether we did `drop='first'`, hence the hassle.

1.4.4 Interpreting coefficients of numeric features

Let's look at coefficients of `PoolArea` and `LotFrontage`.

[33]:

```python
lr_coefs.loc[["PoolArea", "LotArea", "LotFrontage"]]
```

[33]:

```
              Coefficient
PoolArea      2822.370476
LotArea       5109.356718
LotFrontage -1578.664421
```

Intuition:

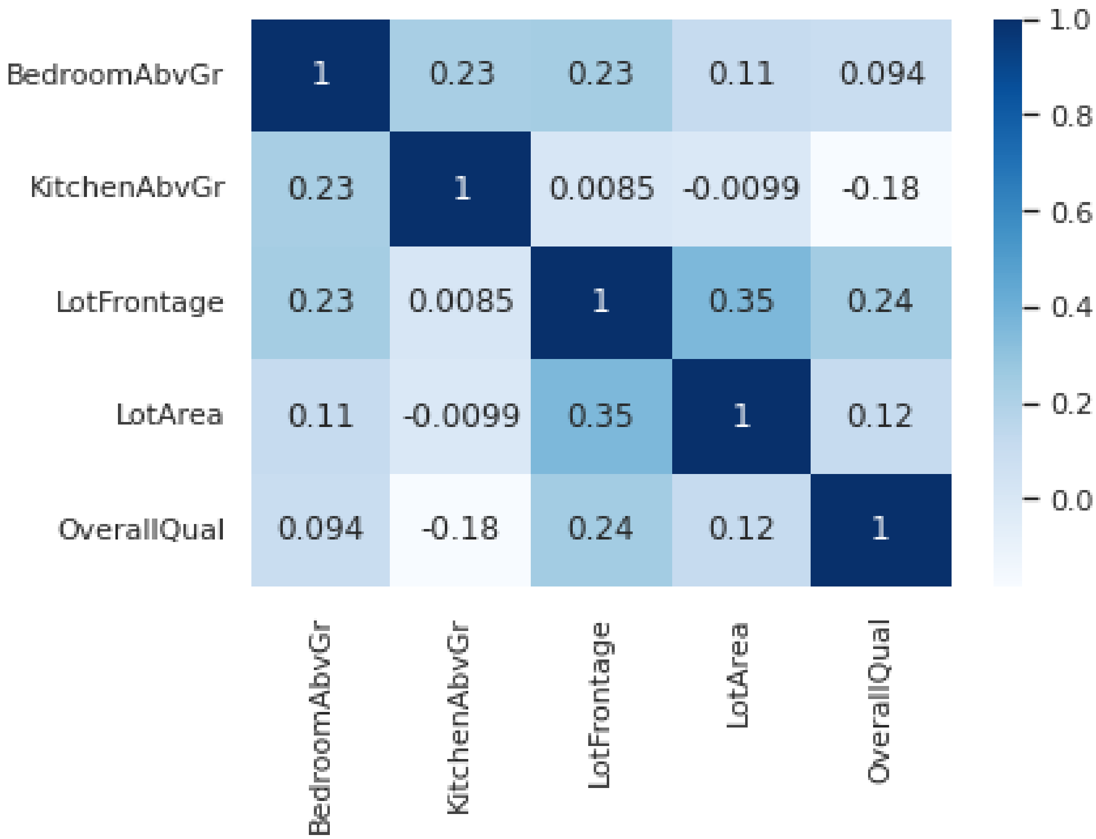
- **Tricky** because numeric features are **scaled**!
- **Increasing** `PoolArea` by ***1 scaled unit*** **increases** the predicted price by ∼ $2822.
- **Increasing** `LotFrontage` by ***1 scaled unit*** **decreases** the predicted price by ∼ $1578.

Does that sound reasonable?

- For `PoolArea`, yes.
- For `LotFrontage`, that's surprising. Something positive would have made more sense?

It's not the case here but maybe the problem is that `LotFrontage` and `LotArea` are very correlated. `LotArea` has a larger positive coefficient.

[34]:
```
cor = X_train_enc[numeric_features[:5]].corr()
sns.heatmap(cor, annot=True, cmap=plt.cm.Blues);
```



First, let's make sure the predictions behave as expected:

[35]:
```
one_example = X_test[:1]
```

[36]:
```
one_example
```

[36]:
```
      Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
147  148          60       RL          NaN     9505   Pave   NaN      IR1

    LandContour Utilities  … ScreenPorch PoolArea PoolQC Fence MiscFeature  \
147         Lvl    AllPub  …           0        0    NaN   NaN         NaN

    MiscVal MoSold  YrSold  SaleType  SaleCondition
147       0      5    2010        WD         Normal

[1 rows x 80 columns]
```

Let's **perturb** the example and add 1 to the `LotArea`.

[37]:
```
one_example_perturbed = one_example.copy()
one_example_perturbed["LotArea"] += 1  # add 1 to the LotArea
```

[38]:
```
one_example_perturbed
```

[38]:
```
      Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
147  148          60       RL          NaN     9506   Pave   NaN      IR1

    LandContour Utilities  … ScreenPorch PoolArea PoolQC Fence MiscFeature  \
147         Lvl    AllPub  …           0        0    NaN   NaN         NaN

    MiscVal MoSold  YrSold  SaleType  SaleCondition
147       0      5    2010        WD         Normal

[1 rows x 80 columns]
```

Prediction on the original example.

[39]:
```
lr.predict(one_example)
```

[39]:
```
array([224795.63596802])
```

Prediction on the perturbed example.

[40]:
```
lr.predict(one_example_perturbed)
```

[40]:
```
array([224796.2040233])
```

- What's the difference between prediction?
- Does the difference make sense given the coefficient of the feature?

[41]:
```
lr.predict(one_example_perturbed) - lr.predict(one_example)
```

[41]:
```
array([0.56805528])
```

[42]:
```
lr_coefs.loc[["LotArea"]]
```

[42]:

	Coefficient
LotArea	5109.356718

- Why did the prediction only go up by \$0.57 instead of \$5109?
- This is an issue of **units**.
  - LotArea is in sqft, but the coefficient is **not** \$5109/sqft **because we scaled the features**.

1.4.5 Example showing how to interpret coefficients of scaled features

- The scaler subtracted the mean and divided by the standard deviation.
- The division actually changed the scale!
- For the unit conversion, we don't care about the subtraction, but only the scaling.

[43]:
```
scaler = preprocessor.named_transformers_["pipeline-1"]["standardscaler"]
```

[44]:
```
scaler.scale_
```

[44]:
```
array([8.21039683e-01, 2.18760172e-01, 2.09591390e+01, 8.99447103e+03,
       1.39208177e+00, 1.11242416e+00, 3.01866337e+01, 2.06318985e+01,
       1.77914527e+02, 4.59101890e+02, 1.63890010e+02, 4.42869860e+02,
       4.42817167e+02, 3.92172897e+02, 4.35820743e+02, 4.69800920e+01,
       5.29468070e+02, 5.18276015e-01, 2.33809970e-01, 5.49298599e-01,
       5.02279069e-01, 1.62604030e+00, 6.34398801e-01, 2.40531598e+01,
       7.40269201e-01, 2.10560601e+02, 1.25388753e+02, 6.57325181e+01,
       6.07432962e+01, 3.03088902e+01, 5.38336322e+01, 4.23249944e+01,
       5.22084645e+02, 1.33231649e+00])
```

[45]:
```
lr_scales = pd.DataFrame(
    data=scaler.scale_, index=numeric_features, columns=["Scale"]
)
lr_scales.head()
```

[45]:

	Scale
BedroomAbvGr	0.821040
KitchenAbvGr	0.218760
LotFrontage	20.959139
LotArea	8994.471032
OverallQual	1.392082

- It seems like LotArea was divided by 8994.471032 sqft.

[46]:
```
lr_coefs.loc["LotArea", "Coefficient"]
```

[46]: 5109.356718094088

- The coefficient tells us that if we increase the **scaled** LotArea by one unit the price would go up by $\approx$ \$5109.
- One scaled unit represents $\sim$ 8994 sq feet (`lr_scales.loc["LotArea", "Scale"]`)

- So if I increase original `LotArea` by one square foot then the predicted price would go up by this amount:

[47]:
```
lr_coefs.loc["LotArea", "Coefficient"] / lr_scales.loc["LotArea", "Scale"]
```

[47]:
```
0.5680552752646643
```

[48]:
```
5109.356718094072 / 8994.471032
```

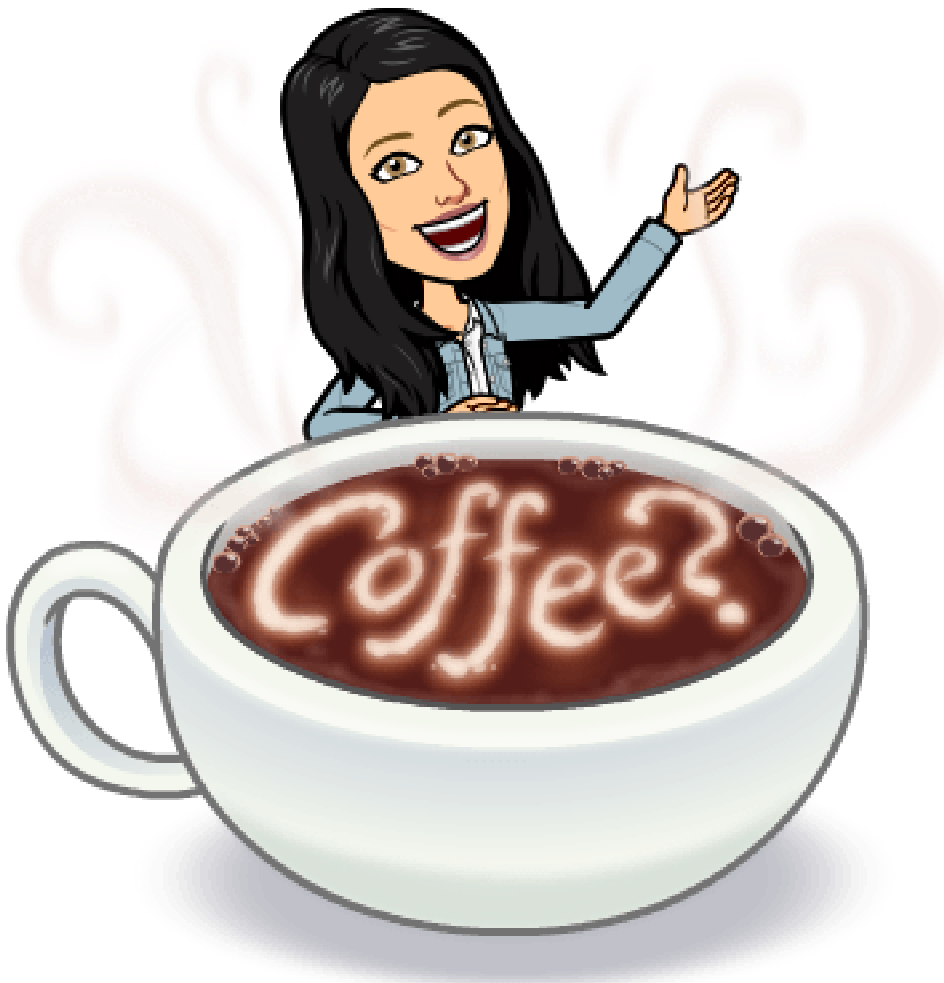
[48]:
```
0.5680552752814816
```

- This makes much more sense. Now we get the number we got before.
- That said don't read too much into these coefficients without statistical training.

1.4.6 Interim summary

- **Correlation** among features might make coefficients completely **uninterpretable**.
- Fairly **straightforward** to interpret coefficients of **ordinal** features.
- In **categorical** features, it's often helpful to consider **one category as a reference** point and think about relative importance.
- For **numeric** features, relative importance is meaningful **after scaling**.
  - You have to be careful about the scale of the feature when interpreting the coefficients.
- Remember that explaining the model $\neq$ explaining the data.
  - the **coefficients** tell us only about the **model** and they might **not** accurately reflect the **data**.

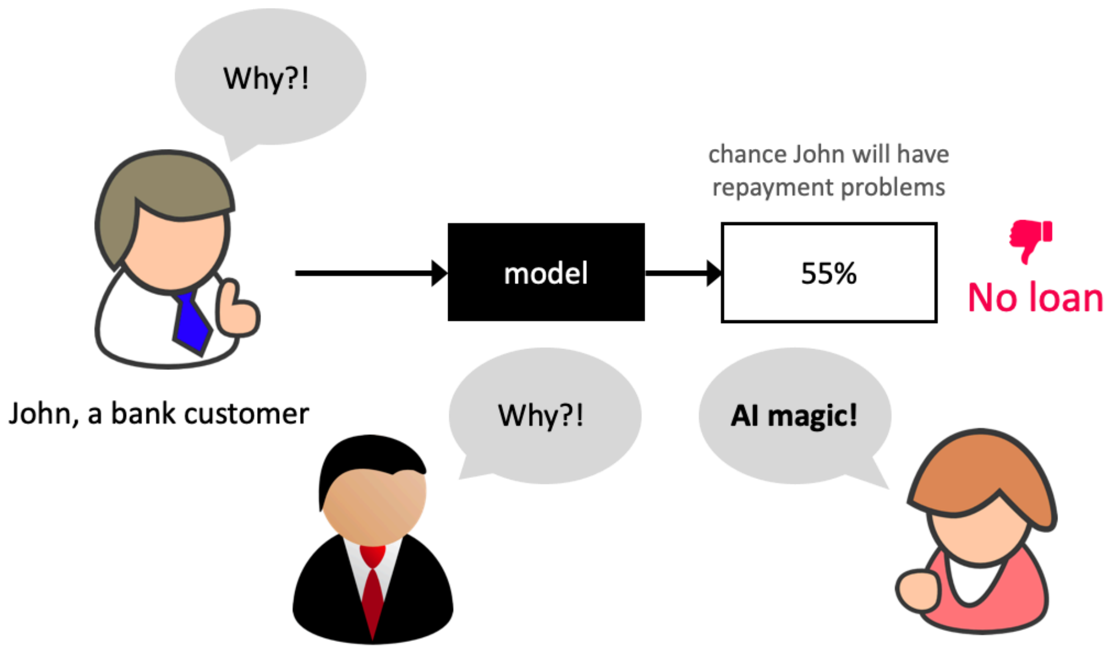
1.5 Break (5 min)



1.6 Interpretability of ML models: Motivations

1.6.1 Why model interpretability?

- Ability to interpret ML models is crucial in many applications such as banking, healthcare, and criminal justice.
- It can be leveraged by domain experts to diagnose systematic errors and underlying biases of complex ML systems.



Source

1.6.2 What is model interpretability?

- In this course, our definition of model iterpretability will be looking at **feature importances**.
- There is more to interpretability than feature importances, but it's a good start!
- Resource:
  - Interpretable Machine Learning
  - Yann LeCun, Kilian Weinberger, Patrice Simard, and Rich Caruana: Panel debate on interpretability

1.6.3 Data

- Let's work with the adult census data set from last lecture and hw3.

[49]:
```
adult_df_large = pd.read_csv("data/adult.csv")
train_df, test_df = train_test_split(adult_df_large, test_size=0.2,␣
↪random_state=42)
train_df_nan = train_df.replace("?", np.NaN)
test_df_nan = test_df.replace("?", np.NaN)
train_df_nan.head()
```

[49]:
```
       age  workclass  fnlwgt     education  education.num  \
5514    26    Private  256263       HS-grad              9
19777   24    Private  170277       HS-grad              9
10781   36    Private   75826     Bachelors             13
32240   22  State-gov   24395  Some-college             10
9876    31  Local-gov  356689     Bachelors             13

           marital.status      occupation   relationship   race     sex  \
```

```
5514        Never-married    Craft-repair  Not-in-family  White    Male
19777       Never-married   Other-service  Not-in-family  White  Female
10781            Divorced    Adm-clerical      Unmarried  White  Female
32240  Married-civ-spouse    Adm-clerical           Wife  White  Female
9876   Married-civ-spouse  Prof-specialty        Husband  White    Male

       capital.gain  capital.loss  hours.per.week native.country income
5514              0             0              25  United-States  <=50K
19777             0             0              35  United-States  <=50K
10781             0             0              40  United-States  <=50K
32240             0             0              20  United-States  <=50K
9876              0             0              40  United-States  <=50K
```

[50]:
```python
numeric_features = ["age", "fnlwgt", "capital.gain", "capital.loss", "hours.per.
 ↪week"]
categorical_features = [
    "workclass",
    "marital.status",
    "occupation",
    "relationship",
    "native.country",
]
ordinal_features = ["education"]
binary_features = ["sex"]
drop_features = ["race", "education.num"]
target_column = "income"
```

[51]:
```python
education_levels = [
    "Preschool",
    "1st-4th",
    "5th-6th",
    "7th-8th",
    "9th",
    "10th",
    "11th",
    "12th",
    "HS-grad",
    "Prof-school",
    "Assoc-voc",
    "Assoc-acdm",
    "Some-college",
    "Bachelors",
    "Masters",
    "Doctorate",
]
```

[52]:
```python
assert set(education_levels) == set(train_df["education"].unique())
```

[53]:
```
numeric_transformer = make_pipeline(SimpleImputer(strategy="median"),␣
 ↪StandardScaler())
tree_numeric_transformer = make_pipeline(SimpleImputer(strategy="median"))

categorical_transformer = make_pipeline(
    SimpleImputer(strategy="constant", fill_value="missing"),
    OneHotEncoder(handle_unknown="ignore"),
)

ordinal_transformer = make_pipeline(
    SimpleImputer(strategy="constant", fill_value="missing"),
    OrdinalEncoder(categories=[education_levels], dtype=int),
)

binary_transformer = make_pipeline(
    SimpleImputer(strategy="constant", fill_value="missing"),
    OneHotEncoder(drop="if_binary", dtype=int),
)

preprocessor = make_column_transformer(
    ("drop", drop_features),
    (numeric_transformer, numeric_features),
    (ordinal_transformer, ordinal_features),
    (binary_transformer, binary_features),
    (categorical_transformer, categorical_features),
)
```

[54]:
```
X_train = train_df_nan.drop(columns=[target_column])
y_train = train_df_nan[target_column]

X_test = test_df_nan.drop(columns=[target_column])
y_test = test_df_nan[target_column]
```

1.6.4 Do we have class imbalance?

- There is class imbalance. But without any context, **both classes seem equally important**.
- Let's use accuracy as our metric.

[55]:
```
train_df_nan["income"].value_counts(normalize=True)
```

[55]:
```
<=50K    0.757985
>50K     0.242015
Name: income, dtype: float64
```

[56]:
```
scoring_metric = "accuracy"
```

[57]:
```python
import warnings

warnings.simplefilter(action="ignore", category=FutureWarning)
warnings.simplefilter(action="ignore", category=UserWarning)
```

Let's store all the results in a dictionary called `results`.

[58]:
```python
results = {}
```

[59]:
```python
from lightgbm.sklearn import LGBMClassifier
from xgboost import XGBClassifier

pipe_lr = make_pipeline(
    preprocessor, LogisticRegression(max_iter=2000, random_state=123)
)
pipe_rf = make_pipeline(preprocessor, RandomForestClassifier(random_state=123))
pipe_xgb = make_pipeline(
    preprocessor, XGBClassifier(random_state=123, eval_metric="logloss",␣
 ↪verbosity=0)
)
pipe_lgbm = make_pipeline(preprocessor, LGBMClassifier(random_state=123))
classifiers = {
    "logistic regression": pipe_lr,
    "random forest": pipe_rf,
    "XGBoost": pipe_xgb,
    "LightGBM": pipe_lgbm,
}
```

[60]:
```python
dummy = DummyClassifier(strategy="most_frequent")
results["Dummy"] = mean_std_cross_val_scores(
    dummy, X_train, y_train, return_train_score=True, scoring=scoring_metric
)
```

[61]:
```python
for (name, model) in classifiers.items():
    results[name] = mean_std_cross_val_scores(
        model, X_train, y_train, return_train_score=True, scoring=scoring_metric
    )
```

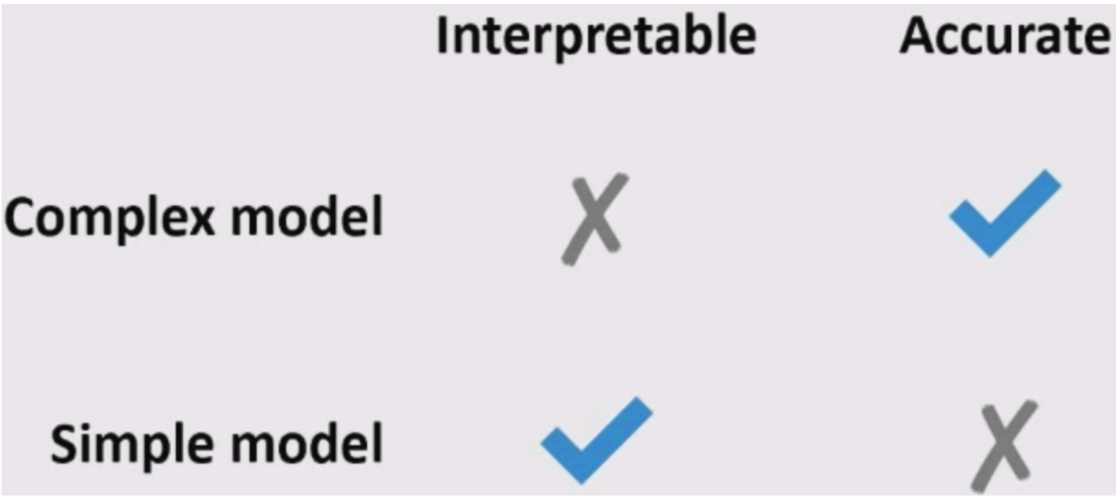
[62]:
```python
pd.DataFrame(results).T
```

[62]:
```
                               fit_time          score_time          test_score  \
Dummy                  0.018 (+/- 0.004)  0.011 (+/- 0.001)  0.758 (+/- 0.000)
logistic regression    1.449 (+/- 0.057)  0.032 (+/- 0.006)  0.850 (+/- 0.006)
random forest         12.519 (+/- 0.380)  0.149 (+/- 0.012)  0.857 (+/- 0.004)
XGBoost                3.407 (+/- 2.349)  0.091 (+/- 0.027)  0.871 (+/- 0.004)
LightGBM               0.405 (+/- 0.061)  0.078 (+/- 0.003)  0.871 (+/- 0.004)

                           train_score
```

```
Dummy                  0.758 (+/- 0.000)
logistic regression    0.851 (+/- 0.001)
random forest          1.000 (+/- 0.000)
XGBoost                0.908 (+/- 0.001)
LightGBM               0.892 (+/- 0.000)
```

- One problem is that often simple models are interpretable but not accurate.
- But more complex models (e.g., LightGBM) are less interpretable.



Source

[63]:
```
# Reactivate default warning settings.
# As a best practice, try not to suppress warnings without a good reason
# and be sure to reactivate them when suppression is not needed anymore.

warnings.simplefilter(action="default", category=FutureWarning)
warnings.simplefilter(action="default", category=UserWarning)
```

1.6.5 Feature importances in linear models

- **Simpler** models are often **more interpretable** but **less accurate**.

Let's create and fit a pipeline with preprocessor and **logistic regression**.

[64]:
```
pipe_lr = make_pipeline(preprocessor, LogisticRegression(max_iter=2000,␣
 ↪random_state=2))
pipe_lr.fit(X_train, y_train);
```

[65]:
```
ohe_feature_names = (
    pipe_rf.named_steps["columntransformer"]
    .named_transformers_["pipeline-4"]
    .named_steps["onehotencoder"]
    .get_feature_names_out(categorical_features)
    .tolist()
)
```

```
feature_names = (
    numeric_features + ordinal_features + binary_features + ohe_feature_names
)
pd.DataFrame(columns=feature_names)
```

[65]:
```
Empty DataFrame
Columns: [age, fnlwgt, capital.gain, capital.loss, hours.per.week, education,
sex, workclass_Federal-gov, workclass_Local-gov, workclass_Never-worked,
workclass_Private, workclass_Self-emp-inc, workclass_Self-emp-not-inc,
workclass_State-gov, workclass_Without-pay, workclass_missing,
marital.status_Divorced, marital.status_Married-AF-spouse,
marital.status_Married-civ-spouse, marital.status_Married-spouse-absent,
marital.status_Never-married, marital.status_Separated, marital.status_Widowed,
occupation_Adm-clerical, occupation_Armed-Forces, occupation_Craft-repair,
occupation_Exec-managerial, occupation_Farming-fishing, occupation_Handlers-
cleaners, occupation_Machine-op-inspct, occupation_Other-service,
occupation_Priv-house-serv, occupation_Prof-specialty, occupation_Protective-
serv, occupation_Sales, occupation_Tech-support, occupation_Transport-moving,
occupation_missing, relationship_Husband, relationship_Not-in-family,
relationship_Other-relative, relationship_Own-child, relationship_Unmarried,
relationship_Wife, native.country_Cambodia, native.country_Canada,
native.country_China, native.country_Columbia, native.country_Cuba,
native.country_Dominican-Republic, native.country_Ecuador, native.country_El-
Salvador, native.country_England, native.country_France, native.country_Germany,
native.country_Greece, native.country_Guatemala, native.country_Haiti,
native.country_Holand-Netherlands, native.country_Honduras, native.country_Hong,
native.country_Hungary, native.country_India, native.country_Iran,
native.country_Ireland, native.country_Italy, native.country_Jamaica,
native.country_Japan, native.country_Laos, native.country_Mexico,
native.country_Nicaragua, native.country_Outlying-US(Guam-USVI-etc),
native.country_Peru, native.country_Philippines, native.country_Poland,
native.country_Portugal, native.country_Puerto-Rico, native.country_Scotland,
native.country_South, native.country_Taiwan, native.country_Thailand,
native.country_Trinadad&Tobago, native.country_United-States,
native.country_Vietnam, native.country_Yugoslavia, native.country_missing]
Index: []

[0 rows x 86 columns]
```

[66]:
```
data = {
    "coefficient": pipe_lr.named_steps["logisticregression"].coef_[0].tolist(),
    "magnitude": np.absolute(
        pipe_lr.named_steps["logisticregression"].coef_[0].tolist()
    ),
}
coef_df = pd.DataFrame(data, index=feature_names).sort_values(
    "magnitude", ascending=False
```

```
)
coef_df.head()
```

[66]:

	coefficient	magnitude
capital.gain	2.355927	2.355927
marital.status_Married-AF-spouse	1.754646	1.754646
occupation_Priv-house-serv	-1.436944	1.436944
marital.status_Married-civ-spouse	1.341062	1.341062
relationship_Wife	1.274917	1.274917

- Increasing `capital.gain` is
  - likely to push the prediction towards ">50k" income class.
- Whereas `occupation_Priv-house-serv` is
  - likely to push the prediction towards "<=50K" income.

Can we get feature importances for non-linear models?

1.7 Model interpretability beyond linear models

We will be looking at three ways for model interpretability.

- `sklearn feature_importances_`

- eli5 (stands for "explain like I'm 5")
- SHAP

1.7.1 `sklearn feature_importances_`

- Many `sklearn` models have `feature_importances_` attribute.
- For **tree-based models**, it's calculated based on **impurity** (gini index or information gain).
- For example, let's look at `feature_importances_` of `RandomForestClassifier`.

Let's create and fit a pipeline with preprocessor and random forest.

[67]:
```
pipe_rf = make_pipeline(preprocessor, RandomForestClassifier(random_state=2))
pipe_rf.fit(X_train, y_train);
```

Which features are driving the predictions the most?

[68]:
```
data = {
    "Importance": pipe_rf.named_steps["randomforestclassifier"].
 ↪feature_importances_,
}
imps = pd.DataFrame(data=data, index=feature_names,).sort_values(
    by="Importance", ascending=False
)
imps.head(10)
```

[68]:

	Importance
fnlwgt	0.169580

```
age                                    0.153339
education                              0.102953
capital.gain                           0.097686
hours.per.week                         0.085583
marital.status_Married-civ-spouse      0.064646
relationship_Husband                   0.048896
capital.loss                           0.033387
marital.status_Never-married           0.028629
occupation_Exec-managerial             0.020458
```

1.7.2 Key point

- Unlike the linear model coefficients, `feature_importances_` **do not have a sign**!
  - They tell us about **importance**, but ***not*** an "**up** or **down**".
  - Indeed, increasing a feature may cause the prediction to first go up, and then go down.
  - This cannot happen in linear models, because they are linear.

Do these importances match with importances identified by logistic regression?

[69]:
```
data = {
    "random forest importance":
        pipe_rf.named_steps["randomforestclassifier"].feature_importances_,
    "logistic regression importance":
        pipe_lr.named_steps["logisticregression"].coef_[0],
}
imps = pd.DataFrame(data=data, index=feature_names)
```

[70]:
```
imps.sort_values(by="random forest importance", ascending=False).head()
```

[70]:

	random forest importance	logistic regression importance
fnlwgt	0.169580	0.078255
age	0.153339	0.359699
education	0.102953	0.184117
capital.gain	0.097686	2.355927
hours.per.week	0.085583	0.370219

Let's compare their top ten important feature lists:

[71]:
```
col_rf = "random forest importance"
col_lr = "logistic regression importance"

ranking = pd.DataFrame({
    col_rf: imps[col_rf].sort_values(ascending=False).index,
    col_lr: imps[col_lr].sort_values(ascending=False, key=np.abs).index
}).rename_axis('ranking')

ranking.head(10)
```

[71]:

```
                    random forest importance     logistic regression importance
ranking
0                                     fnlwgt                       capital.gain
1                                        age    marital.status_Married-AF-spouse
2                                  education          occupation_Priv-house-serv
3                               capital.gain   marital.status_Married-civ-spouse
4                             hours.per.week                   relationship_Wife
5          marital.status_Married-civ-spouse            native.country_Columbia
6                       relationship_Husband           occupation_Prof-specialty
7                               capital.loss          occupation_Exec-managerial
8               marital.status_Never-married  native.country_Dominican-Republic
9                 occupation_Exec-managerial             relationship_Own-child
```

- In their top 10 lists, both models agree on:
  - `capital.gain`
  - `marital.status_Married-civ-spouse`
  - `occupation_Exec-managerial`
- The actual numbers for random forests and logistic regression are not really comparable.

1.7.3 How can we get feature importances for non sklearn models?

- One way to do it is by using a tool called `eli5`.

You'll have to install it

`conda install -n cpsc330 -c conda-forge eli5`

Let's look at feature importances for `XGBClassifier`.

[72]:

```python
import eli5

pipe_xgb = make_pipeline(
    preprocessor,
    XGBClassifier(random_state=123, eval_metric="logloss", verbosity=0))

warnings.simplefilter(action="ignore", category=UserWarning)  # ignore warnings
pipe_xgb.fit(X_train, y_train);
warnings.simplefilter(action="default", category=UserWarning)  # reactivate
 ↪warnings

eli5_xgb = eli5.explain_weights(pipe_xgb.named_steps["xgbclassifier"],
 ↪feature_names=feature_names)
eli5_xgb
```

[72]:

```
Explanation(estimator="XGBClassifier(base_score=0.5, booster='gbtree',
colsample_bylevel=1,\n              colsample_bynode=1, colsample_bytree=1,
enable_categorical=False,\n           eval_metric='logloss', gamma=0,
gpu_id=-1, importance_type=None,\n              interaction_constraints='',
learning_rate=0.300000012,\n          max_delta_step=0, max_depth=6,
```

```
min_child_weight=1, missing=nan,\n              monotone_constraints='()',
n_estimators=100, n_jobs=8,\n              num_parallel_tree=1,
predictor='auto', random_state=123,\n              reg_alpha=0, reg_lambda=1,
scale_pos_weight=1, subsample=1,\n              tree_method='exact',
validate_parameters=1, verbosity=0)", description='\nXGBoost feature
importances; values are numbers 0 <= x <= 1;\nall values sum to 1.\n',
error=None, method='feature importances', is_regression=False, targets=None, fea
ture_importances=FeatureImportances(importances=[FeatureWeight(feature='marital.
status_Married-civ-spouse', weight=0.40608603, std=None, value=None),
FeatureWeight(feature='capital.gain', weight=0.054722264, std=None, value=None),
FeatureWeight(feature='relationship_Own-child', weight=0.044089068, std=None,
value=None), FeatureWeight(feature='education', weight=0.034879886, std=None,
value=None), FeatureWeight(feature='occupation_Other-service',
weight=0.032538496, std=None, value=None), FeatureWeight(feature='capital.loss',
weight=0.026842514, std=None, value=None),
FeatureWeight(feature='occupation_Prof-specialty', weight=0.024732435, std=None,
value=None), FeatureWeight(feature='occupation_Exec-managerial',
weight=0.01791228, std=None, value=None),
FeatureWeight(feature='occupation_Tech-support', weight=0.017841721, std=None,
value=None), FeatureWeight(feature='occupation_Handlers-cleaners',
weight=0.017215686, std=None, value=None),
FeatureWeight(feature='occupation_Machine-op-inspct', weight=0.016416635,
std=None, value=None), FeatureWeight(feature='occupation_Farming-fishing',
weight=0.016381254, std=None, value=None),
FeatureWeight(feature='workclass_Federal-gov', weight=0.015793154, std=None,
value=None), FeatureWeight(feature='age', weight=0.011652168, std=None,
value=None), FeatureWeight(feature='workclass_Self-emp-inc', weight=0.01078908,
std=None, value=None), FeatureWeight(feature='hours.per.week',
weight=0.0107445745, std=None, value=None),
FeatureWeight(feature='relationship_Wife', weight=0.010192984, std=None,
value=None), FeatureWeight(feature='sex', weight=0.010090632, std=None,
value=None), FeatureWeight(feature='relationship_Not-in-family',
weight=0.009422912, std=None, value=None),
FeatureWeight(feature='workclass_Self-emp-not-inc', weight=0.009076657,
std=None, value=None)], remaining=66), decision_tree=None,
highlight_spaces=None, transition_features=None, image=None)
```

Let's look at feature importances for `LGBMClassifier`.

[73]:
```python
pipe_lgbm = make_pipeline(preprocessor, LGBMClassifier(random_state=123))
pipe_lgbm.fit(X_train, y_train)
eli5_lgbm = eli5.explain_weights(
    pipe_lgbm.named_steps["lgbmclassifier"], feature_names=feature_names
)
eli5_lgbm
```

[73]: Explanation(estimator='LGBMClassifier(random_state=123)',
description='\nLightGBM feature importances; values are numbers 0 <= x <=
1;\nall values sum to 1.\n', error=None, method='feature importances',
is_regression=False, targets=None, feature_importances=FeatureImportances(import
ances=[FeatureWeight(feature='marital.status_Married-civ-spouse',
weight=0.35584397468549844, std=None, value=None),
FeatureWeight(feature='capital.gain', weight=0.19098036725150688, std=None,
value=None), FeatureWeight(feature='education', weight=0.13630962171648306,
std=None, value=None), FeatureWeight(feature='age', weight=0.08515574639356348,
std=None, value=None), FeatureWeight(feature='capital.loss',
weight=0.06393026262311322, std=None, value=None),
FeatureWeight(feature='hours.per.week', weight=0.0418456004162135, std=None,
value=None), FeatureWeight(feature='fnlwgt', weight=0.02451337553136395,
std=None, value=None), FeatureWeight(feature='occupation_Exec-managerial',
weight=0.013429664556178146, std=None, value=None),
FeatureWeight(feature='occupation_Prof-specialty', weight=0.012015760716975882,
std=None, value=None), FeatureWeight(feature='occupation_Other-service',
weight=0.0066740803861996744, std=None, value=None),
FeatureWeight(feature='sex', weight=0.006525060558280043, std=None, value=None),
FeatureWeight(feature='relationship_Wife', weight=0.005453787951464236,
std=None, value=None), FeatureWeight(feature='workclass_Self-emp-not-inc',
weight=0.005364964149799726, std=None, value=None),
FeatureWeight(feature='occupation_Farming-fishing', weight=0.005168542153226924,
std=None, value=None), FeatureWeight(feature='relationship_Own-child',
weight=0.0046119894507038505, std=None, value=None),
FeatureWeight(feature='occupation_Tech-support', weight=0.0032505444073185675,
std=None, value=None), FeatureWeight(feature='occupation_Sales',
weight=0.0024604164249372976, std=None, value=None),
FeatureWeight(feature='workclass_Private', weight=0.002385110169021698,
std=None, value=None), FeatureWeight(feature='workclass_Federal-gov',
weight=0.002384806796571508, std=None, value=None),
FeatureWeight(feature='occupation_Handlers-cleaners',
weight=0.00228880851674175, std=None, value=None)], remaining=66),
decision_tree=None, highlight_spaces=None, transition_features=None, image=None)

You can also look at feature importances for RandomForestClassifier, which we have already trained above.

[74]:
```
eli5_rf = eli5.explain_weights(
    pipe_rf.named_steps["randomforestclassifier"], feature_names=feature_names
)
eli5_rf
```

[74]: Explanation(estimator='RandomForestClassifier(random_state=2)',
description='\nRandom forest feature importances; values are numbers 0 <= x <=
1;\nall values sum to 1.\n', error=None, method='feature importances',
is_regression=False, targets=None, feature_importances=FeatureImportances(import
ances=[FeatureWeight(feature='fnlwgt', weight=0.16958005428552844,

```
std=0.00562712608917515, value=None), FeatureWeight(feature='age',
weight=0.1533390215043909, std=0.019798601881412622, value=None),
FeatureWeight(feature='education', weight=0.10295283451436564,
std=0.017386002656557657, value=None), FeatureWeight(feature='capital.gain',
weight=0.09768586081972082, std=0.023939780703838818, value=None),
FeatureWeight(feature='hours.per.week', weight=0.08558272511107902,
std=0.012506242582114328, value=None),
FeatureWeight(feature='marital.status_Married-civ-spouse',
weight=0.06464573433266022, std=0.06927178573889378, value=None),
FeatureWeight(feature='relationship_Husband', weight=0.04889639212082628,
std=0.055850610267523165, value=None), FeatureWeight(feature='capital.loss',
weight=0.03338747186463071, std=0.007844378859870025, value=None),
FeatureWeight(feature='marital.status_Never-married',
weight=0.02862861716671859, std=0.03698238581587818, value=None),
FeatureWeight(feature='occupation_Exec-managerial', weight=0.0204579927038537,
std=0.01053650089346291, value=None), FeatureWeight(feature='occupation_Prof-
specialty', weight=0.019333429958487774, std=0.009345251361284262, value=None),
FeatureWeight(feature='sex', weight=0.011773608675662732,
std=0.011047957330415109, value=None),
FeatureWeight(feature='relationship_Wife', weight=0.010990609952383412,
std=0.01122621572509277, value=None), FeatureWeight(feature='workclass_Private',
weight=0.009378492963718118, std=0.0018793227079254462, value=None),
FeatureWeight(feature='relationship_Not-in-family', weight=0.009322842064467672,
std=0.012121966770941614, value=None), FeatureWeight(feature='workclass_Self-
emp-not-inc', weight=0.00797575339268079, std=0.0017752094852695698,
value=None), FeatureWeight(feature='occupation_Other-service',
weight=0.007805796269149016, std=0.0052126895400305875, value=None),
FeatureWeight(feature='workclass_Self-emp-inc', weight=0.006597991594898411,
std=0.0031876281997523723, value=None), FeatureWeight(feature='relationship_Own-
child', weight=0.006592011991669542, std=0.011971256496172504, value=None),
FeatureWeight(feature='native.country_United-States',
weight=0.00637699697609062, std=0.0012210456325934707, value=None)],
remaining=66), decision_tree=None, highlight_spaces=None,
transition_features=None, image=None)
```

Let's compare them with weights what we got with `sklearn` `feature_importances_`

[75]:
```
data = {
    "Importance": pipe_rf.named_steps["randomforestclassifier"].
 ↪feature_importances_,
}
sk_feat_imp_rf = pd.DataFrame(data=data, index=feature_names,).sort_values(
    by="Importance", ascending=False
)
sk_feat_imp_rf.head(10)
```

[75]:

	Importance
fnlwgt	0.169580
age	0.153339
education	0.102953
capital.gain	0.097686
hours.per.week	0.085583
marital.status_Married-civ-spouse	0.064646
relationship_Husband	0.048896
capital.loss	0.033387
marital.status_Never-married	0.028629
occupation_Exec-managerial	0.020458

Let's see them all together

[76]:

```
def eli5_features(explain_weights):
    return [f.feature for f in explain_weights.feature_importances.importances]

eli5_rows = len(eli5_features(eli5_xgb))

pd.DataFrame({
    "XGB eli5": eli5_features(eli5_xgb),
    "LGBM eli5": eli5_features(eli5_lgbm),
    "RandomForest eli5": eli5_features(eli5_xgb),
    "RandomForest sklearn": sk_feat_imp_rf.head(eli5_rows).index
}).rename_axis('ranking')
```

[76]:

ranking	XGB eli5	LGBM eli5 \
0	marital.status_Married-civ-spouse	marital.status_Married-civ-spouse
1	capital.gain	capital.gain
2	relationship_Own-child	education
3	education	age
4	occupation_Other-service	capital.loss
5	capital.loss	hours.per.week
6	occupation_Prof-specialty	fnlwgt
7	occupation_Exec-managerial	occupation_Exec-managerial
8	occupation_Tech-support	occupation_Prof-specialty
9	occupation_Handlers-cleaners	occupation_Other-service
10	occupation_Machine-op-inspct	sex
11	occupation_Farming-fishing	relationship_Wife
12	workclass_Federal-gov	workclass_Self-emp-not-inc
13	age	occupation_Farming-fishing
14	workclass_Self-emp-inc	relationship_Own-child
15	hours.per.week	occupation_Tech-support
16	relationship_Wife	occupation_Sales
17	sex	workclass_Private
18	relationship_Not-in-family	workclass_Federal-gov
19	workclass_Self-emp-not-inc	occupation_Handlers-cleaners

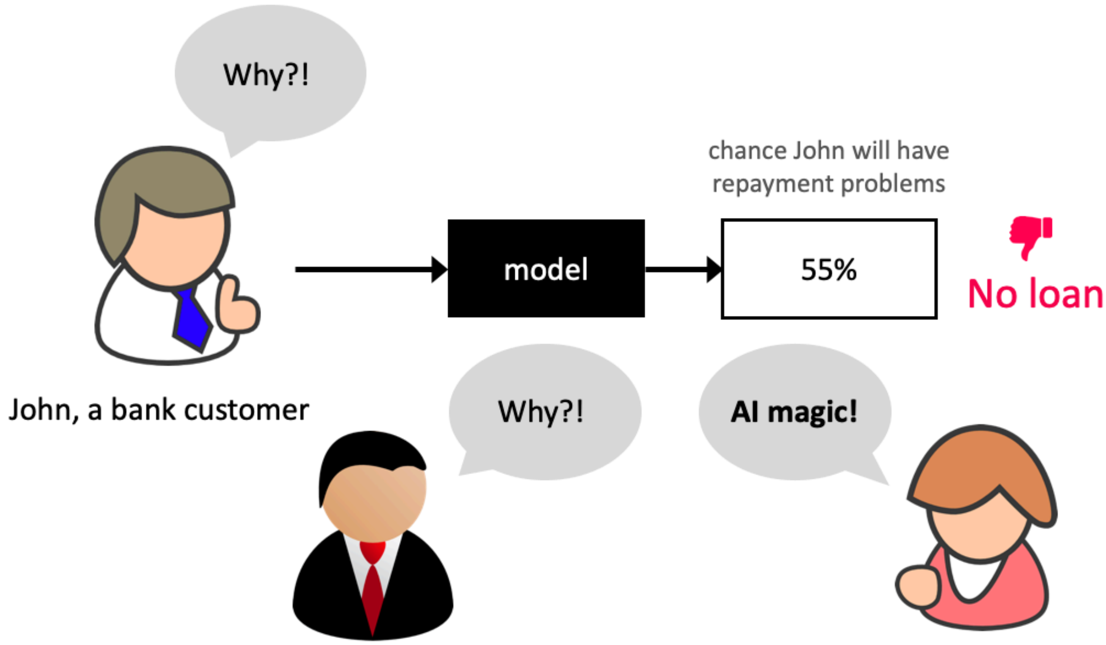
ranking	RandomForest eli5	RandomForest sklearn
0	marital.status_Married-civ-spouse	fnlwgt
1	capital.gain	age
2	relationship_Own-child	education
3	education	capital.gain
4	occupation_Other-service	hours.per.week
5	capital.loss	marital.status_Married-civ-spouse
6	occupation_Prof-specialty	relationship_Husband
7	occupation_Exec-managerial	capital.loss
8	occupation_Tech-support	marital.status_Never-married
9	occupation_Handlers-cleaners	occupation_Exec-managerial
10	occupation_Machine-op-inspct	occupation_Prof-specialty
11	occupation_Farming-fishing	sex
12	workclass_Federal-gov	relationship_Wife
13	age	workclass_Private
14	workclass_Self-emp-inc	relationship_Not-in-family
15	hours.per.week	workclass_Self-emp-not-inc
16	relationship_Wife	occupation_Other-service
17	sex	workclass_Self-emp-inc
18	relationship_Not-in-family	relationship_Own-child
19	workclass_Self-emp-not-inc	native.country_United-States

- These values tell us **globally** about which features are important
- But what if you want to explain a ***specific*** **prediction**?
- Some fancier tools can help us do this

1.8 SHAP (SHapley Additive exPlanations)

- A sophisticated ***measure*** of the **contribution of each feature**.
- Lundberg and Lee, 2017
- We won't go in details. You may refer to Scott Lundberg's GitHub repo if you are interested to know more.

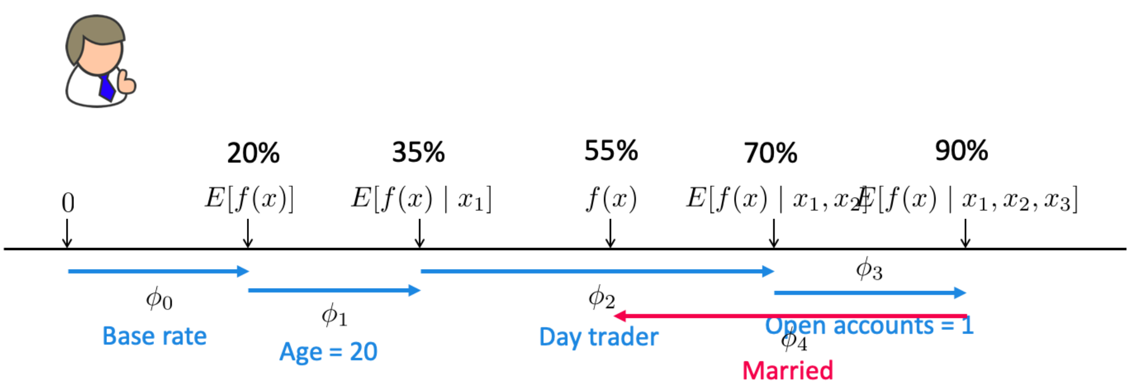
1.8.1 General idea



Source

1.8.2 General idea

- Provides following kind of explanation
  - Start at a **base rate** (e.g., how often people get their loans rejected).
  - **Add one feature** at a time and see **how it impacts** the decision.



Source

Let's try it out on tree-based models.

First you'll have to install it.

```
conda install -n cpsc330 -c conda-forge shap
```

Let's create train and test dataframes with our transformed features.

[77]:
```
X_train_enc = pd.DataFrame(
    data=preprocessor.transform(X_train).toarray(),
    columns=feature_names,
    index=X_train.index,
)
X_train_enc.head()
```

[77]:
```
            age    fnlwgt  capital.gain  capital.loss  hours.per.week  \
5514  -0.921955  0.632531     -0.147166      -0.21768       -1.258387
19777 -1.069150 -0.186155     -0.147166      -0.21768       -0.447517
10781 -0.185975 -1.085437     -0.147166      -0.21768       -0.042081
32240 -1.216346 -1.575119     -0.147166      -0.21768       -1.663822
9876  -0.553965  1.588701     -0.147166      -0.21768       -0.042081

       education  sex  workclass_Federal-gov  workclass_Local-gov  \
5514         8.0  1.0                    0.0                  0.0
19777        8.0  0.0                    0.0                  0.0
10781       13.0  0.0                    0.0                  0.0
32240       12.0  0.0                    0.0                  0.0
9876        13.0  1.0                    0.0                  1.0

       workclass_Never-worked  …  native.country_Puerto-Rico  \
5514                      0.0  …                         0.0
19777                     0.0  …                         0.0
10781                     0.0  …                         0.0
32240                     0.0  …                         0.0
9876                      0.0  …                         0.0

       native.country_Scotland  native.country_South  native.country_Taiwan  \
5514                       0.0                   0.0                    0.0
19777                      0.0                   0.0                    0.0
10781                      0.0                   0.0                    0.0
32240                      0.0                   0.0                    0.0
9876                       0.0                   0.0                    0.0

       native.country_Thailand  native.country_Trinadad&Tobago  \
5514                       0.0                             0.0
19777                      0.0                             0.0
10781                      0.0                             0.0
32240                      0.0                             0.0
9876                       0.0                             0.0

       native.country_United-States  native.country_Vietnam  \
5514                            1.0                     0.0
19777                           1.0                     0.0
10781                           1.0                     0.0
32240                           1.0                     0.0
```

```
9876                            1.0                    0.0

       native.country_Yugoslavia  native.country_missing
5514                         0.0                     0.0
19777                        0.0                     0.0
10781                        0.0                     0.0
32240                        0.0                     0.0
9876                         0.0                     0.0

[5 rows x 86 columns]
```

[78]:
```
X_test_enc = pd.DataFrame(
    data=preprocessor.transform(X_test).toarray(),
    columns=feature_names,
    index=X_test.index,
)

X_test_enc.shape
```

[78]:
```
(6513, 86)
```

Let's get SHAP values for train and test data.

[79]:
```
import shap

lgbm_explainer = shap.TreeExplainer(pipe_lgbm.named_steps["lgbmclassifier"])
```

[80]:
```
train_lgbm_shap_values = lgbm_explainer.shap_values(X_train_enc)
train_lgbm_shap_values[1].shape
```

```
LightGBM binary classifier with TreeExplainer shap values output has changed to
a list of ndarray
```

[80]:
```
(26048, 86)
```

[81]:
```
test_lgbm_shap_values = lgbm_explainer.shap_values(X_test_enc)
test_lgbm_shap_values[1].shape
```

[81]:
```
(6513, 86)
```

- For classification, it's a bit confusing. It gives SHAP arrays for both classes.
- Let's stick to shap values for class 1, i.e., income > 50K.

For each example and each feature we have a SHAP value.

[82]:
```
train_lgbm_shap_values[1]
```

[82]:
```
array([[-4.23243013e-01, -5.89878323e-02, -2.65263112e-01, …,
         9.63030623e-04,  0.00000000e+00,  5.74466631e-04],
       [-6.83190014e-01,  1.15708200e-02, -2.72482485e-01, …,
```

```
         8.17274476e-04,  0.00000000e+00,  8.09406158e-04],
       [ 4.49106369e-01, -1.32455245e-01, -2.39454581e-01, …,
         8.27603313e-04,  0.00000000e+00,  4.22023416e-03],
       …,
       [ 1.02714900e+00,  2.38119557e-02, -1.88163464e-01, …,
         1.13580827e-03,  0.00000000e+00,  6.94390861e-04],
       [ 6.37084418e-01,  2.90573592e-02, -3.03429292e-01, …,
         9.70726909e-04,  0.00000000e+00,  2.16856964e-03],
       [-1.24950883e+00,  1.19867799e-01, -2.23378846e-01, …,
         9.70674774e-04,  0.00000000e+00,  9.73838044e-04]])
```

Let's look at the average SHAP values associated with each feature.

[83]:
```python
values = np.abs(train_lgbm_shap_values[1]).mean(0)
pd.DataFrame(data=values, index=feature_names, columns=["SHAP"]).sort_values(
    by="SHAP", ascending=False
).head(10)
```

[83]:

	SHAP
marital.status_Married-civ-spouse	1.086269
age	0.823933
capital.gain	0.572778
education	0.409543
hours.per.week	0.313901
sex	0.188874
capital.loss	0.138607
relationship_Own-child	0.112871
occupation_Exec-managerial	0.107399
occupation_Prof-specialty	0.098181

- You can think of this as **global feature importance**
- But we'll see next that it gives you much more

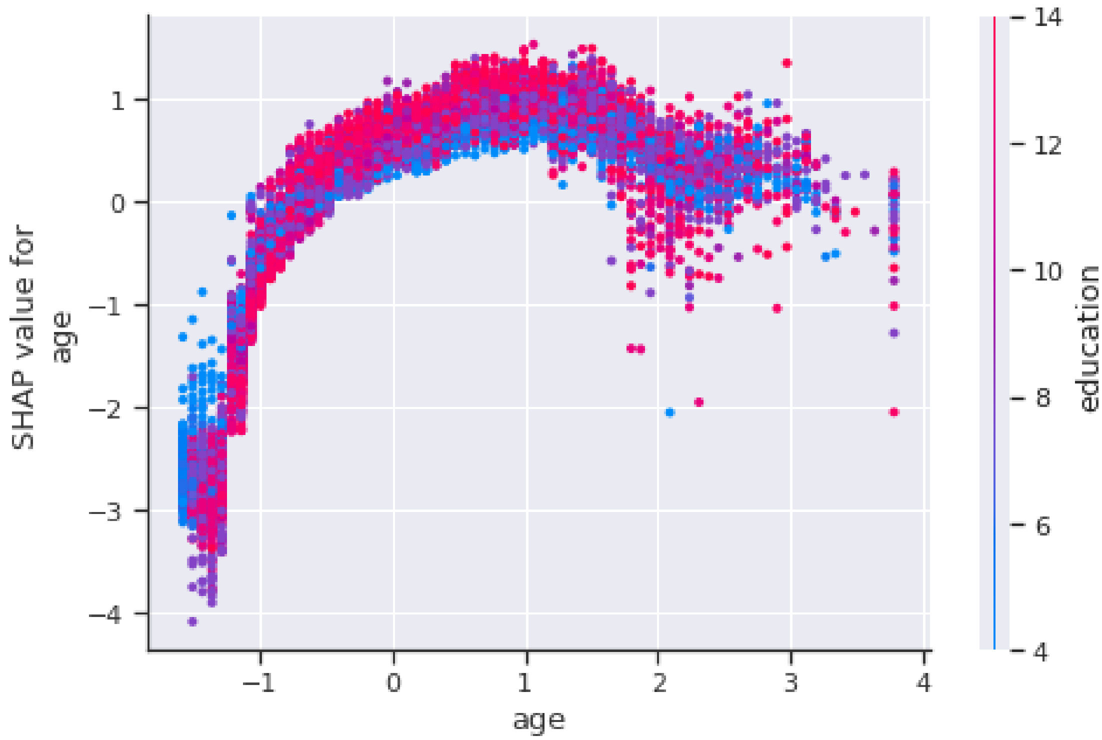
1.9 SHAP plots

[84]:
```python
# load JS visualization code to notebook
shap.initjs()
```

```
<IPython.core.display.HTML object>
```

1.9.1 Dependence plot

[85]:
```python
shap.dependence_plot("age", train_lgbm_shap_values[1], X_train_enc)
```



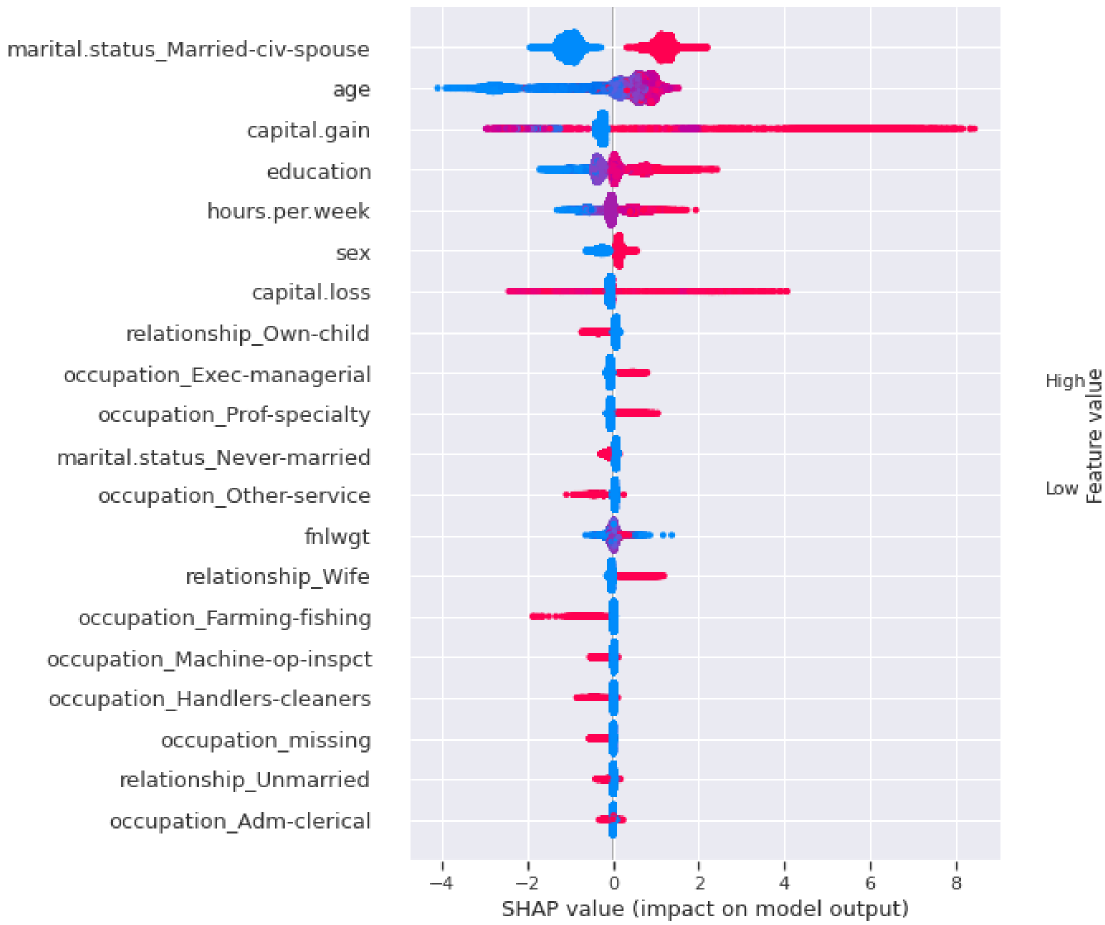
The plot above shows effect of `age` feature on the prediction for class "**>50K**".

- Each **dot** is a **single prediction** for examples above.
- The **x-axis** represents values of the **feature** age (scaled).
- The **y-axis** is the **SHAP value** for that feature
  - This represents how much knowing that **feature's value changes the output** of the model for that example's prediction.
- Lower values of age have smaller SHAP values for class ">50K".
- Similarly, higher values of age also have a bit smaller SHAP values for class ">50K", which makes sense.

- There is some optimal value of age between scaled age of 1 which gives highest SHAP values for for class ">50K".
- Ignore the colour for now.
  - The colour corresponds to a **second feature (education** feature in this case) that may have an **interaction effect** with the feature we are plotting (age).

1.9.2 Summary plot

[86]:
```
shap.summary_plot(train_lgbm_shap_values[1], X_train_enc)
```



The plot shows - (y-axis) The most important features for predicting the class - (x-axis) The direction of how feature values are going to drive the prediction. - (low feature values: blue; high feature values: red)

For example - Presence of the marital status of Married-civ-spouse seems to have bigger SHAP values for class 1 and absence seems to have smaller SHAP values for class 1. - Higher levels of education seem to have bigger SHAP values for class 1 whereas smaller levels of education have smaller SHAP values. - higher education pushes prediction towards >50K - lower education pushes prediction away from >50K

1.9.3 Force plot

- Let's try to explain predictions on a couple of examples from the test data.
- I'm sampling some examples where target is <=50K and some examples where target is >50K.

[87]:
```
l50k_indices, g50k_indices = y_test.reset_index().groupby('income').indices.
 ↪values()
l50k_indices, g50k_indices
```

[87]:
```
(array([   0,    1,    2, …, 6508, 6509, 6511]),
 array([  17,   18,   30, …, 6505, 6510, 6512]))
```

[88]:
```
ex_l50k_index = l50k_indices[10]  # index of the tenth row with <=50K
ex_g50k_index = g50k_indices[10]  # index of the tenth row with >50K
ex_l50k_index, ex_g50k_index
```

[88]:
```
(10, 68)
```

See the rows with these indices:

[89]:
```
y_test.iloc[[ex_l50k_index, ex_g50k_index]]
```

[89]:
```
345      <=50K
23011     >50K
Name: income, dtype: object
```

1.9.4 Example with prediction <=50K

[90]:
```
# pipe_lgbm.named_steps["lgbmclassifier"].
↪predict_proba(X_test_enc)[ex_l50k_index]
```

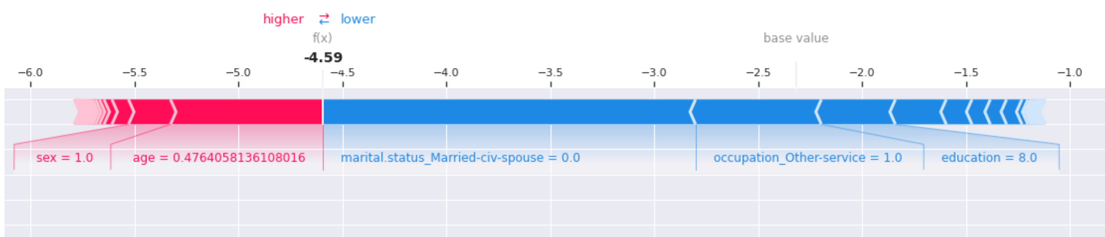
[91]:
```
# pipe_lgbm.named_steps["lgbmclassifier"].predict(X_test_enc)[ex_l50k_index]
```

[92]:
```
# pipe_lgbm.named_steps["lgbmclassifier"].predict(X_test_enc,␣
↪raw_score=True)[ex_l50k_index]  # raw score of the model
```

[93]:
```
# base_value
lgbm_explainer.expected_value[1]
```

[93]:
```
-2.3163172510079377
```

[94]:
```
shap.force_plot(
    base_value=lgbm_explainer.expected_value[1],
    shap_values=test_lgbm_shap_values[1][ex_l50k_index, :],
    features=X_test_enc.iloc[ex_l50k_index, :],
    matplotlib=True,
)
```



1.9.5 Example with prediction >50K

[95]:
```
# pipe_lgbm.named_steps["lgbmclassifier"].
↪predict_proba(X_test_enc)[ex_g50k_index]
```

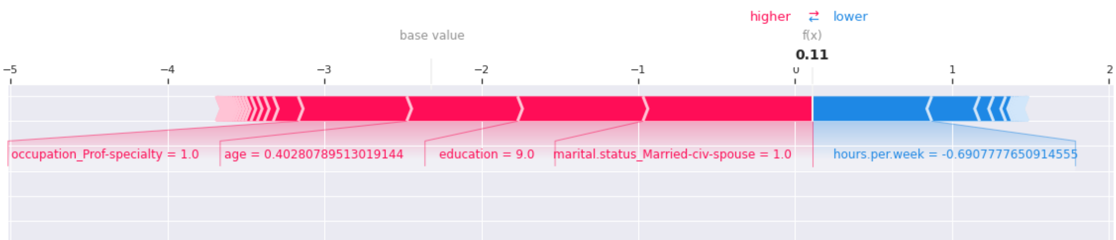
[96]:
```
# pipe_lgbm.named_steps["lgbmclassifier"].predict(X_test_enc,␣
↪raw_score=True)[ex_g50k_index]   # raw model score
```

[97]:
```
# test_lgbm_shap_values[1][ex_g50k_index, :]
```

[98]:
```
# base_value
lgbm_explainer.expected_value[1]
```

[98]: -2.3163172510079377

[99]:
```
shap.force_plot(
    base_value=lgbm_explainer.expected_value[1],
    shap_values=test_lgbm_shap_values[1][ex_g50k_index, :],
    features=X_test_enc.iloc[ex_g50k_index, :],
    matplotlib=True,
)
```



Observations:

- Everything is with **respect to class 1** here.
- The base value for class 1 is -2.316. (You can think of this as the average raw score.)
- We see the forces that drive the prediction.
- That is, we can see the main factors pushing it from the base value (average over the dataset) to this particular prediction.
- Features that **push** the prediction to a **higher** value are shown in **red**.
- Features that **push** the prediction to a **lower** value are shown in **blue**.

Note A nice thing about SHAP values is that the feature importances sum to the prediction:

[100]:
```
test_lgbm_shap_values[1][ex_g50k_index, :].sum() + lgbm_explainer.
↪expected_value[1]
```

[100]: 0.11096043410156309

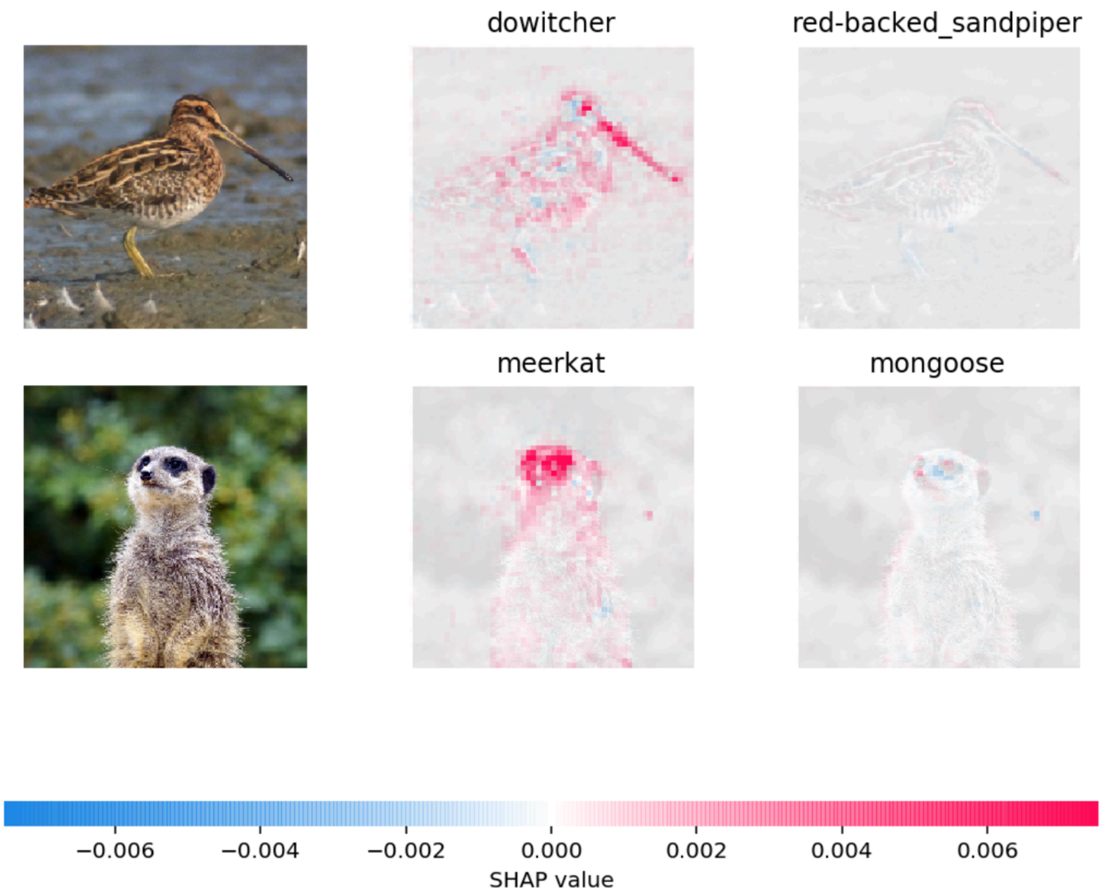
[101]:
```
# recall that
y_test.iloc[ex_g50k_index]
```

[101]: '>50K'

1.9.6 SHAP provides explainer for different kinds of models

- TreeExplainer (supports XGBoost, CatBoost, LightGBM)
- DeepExplainer (supports deep-learning models)
- KernelExplainer (supports kernel-based models)
- GradientExplainer (supports Keras and Tensorflow models)

- SHAP can also be used to explain text classification and image classification
- Example: In the picture below, red pixels represent positive SHAP values that increase the probability of the class, while blue pixels represent negative SHAP values the reduce the probability of the class.



Source

1.9.7 Other tools

- lime is another package.

1.9.8 In summary:

- So far we've only used sklearn models.
- Most sklearn models have some built-in measure of feature importances.
- On many tasks we need to move beyond sklearn, e.g. LightGBM, deep learning.
- These tools work on other models as well, which makes them extremely useful.

1.9.9 Why do we want this information?

Possible reasons:

- Identify features that are not useful and maybe remove them.
- Get guidance on what new data to collect.
  - New features related to useful features -> better results.
  - Don't bother collecting useless features -> save resources.
- Help explain why the model is making certain predictions.
  - Debugging, if the model is behaving strangely.
  - Regulatory requirements.
  - Fairness / bias.
  - Keep in mind this can be used on **deployment** predictions!

1.9.10 Questions for you

1.9.11 True/False

1. You train a random forest on a binary classification problem with two classes [neg, pos]. A value of 0.580 for feat1 given by `feature_importances_` attribute of your model means that increasing the value of feat1 will drive us towards positive class. **FALSE** Feature importance ranges from 0 to 1. Never negative.
2. eli5 can be used to get feature importances for non `sklearn` models. **TRUE**
3. With SHAP you can only explain predictions on the training examples. **FALSE** It can also be used in deployment predictions.

[ ]:

```
```