Categorical Variables

ACTL3143 & ACTL5111 Deep Learning for Actuaries

Patrick Laub

Lecture Outline

- **Preprocessing**
- French Motor Claims Dataset
- Ordinal Variables

Keras model methods

- `compile`: specify the loss function and optimiser
- `fit`: learn the parameters of the model
- `predict`: apply the model
- `evaluate`: apply the model and calculate a metric

```python
random.seed(12)
model = Sequential()
model.add(Dense(1, activation="relu"))
model.compile("adam", "poisson")
model.fit(X_train, y_train, verbose=0)
y_pred = model.predict(X_val, verbose=0)
print(model.evaluate(X_val, y_val, verbose=0)
```

```
4.944334506988525
```

Scikit-learn model methods

- `fit`: learn the parameters of the model
- `predict`: apply the model
- `score`: apply the model and calculate a metric

```python
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_val)
print(model.score(X_val, y_val))
```

```
-0.666850597951445
```

UNSW SYDNEY

Scikit-learn preprocessing methods

- `fit`: learn the parameters of the transformation
- `transform`: apply the transformation
- `fit_transform`: learn the parameters and apply the transformation

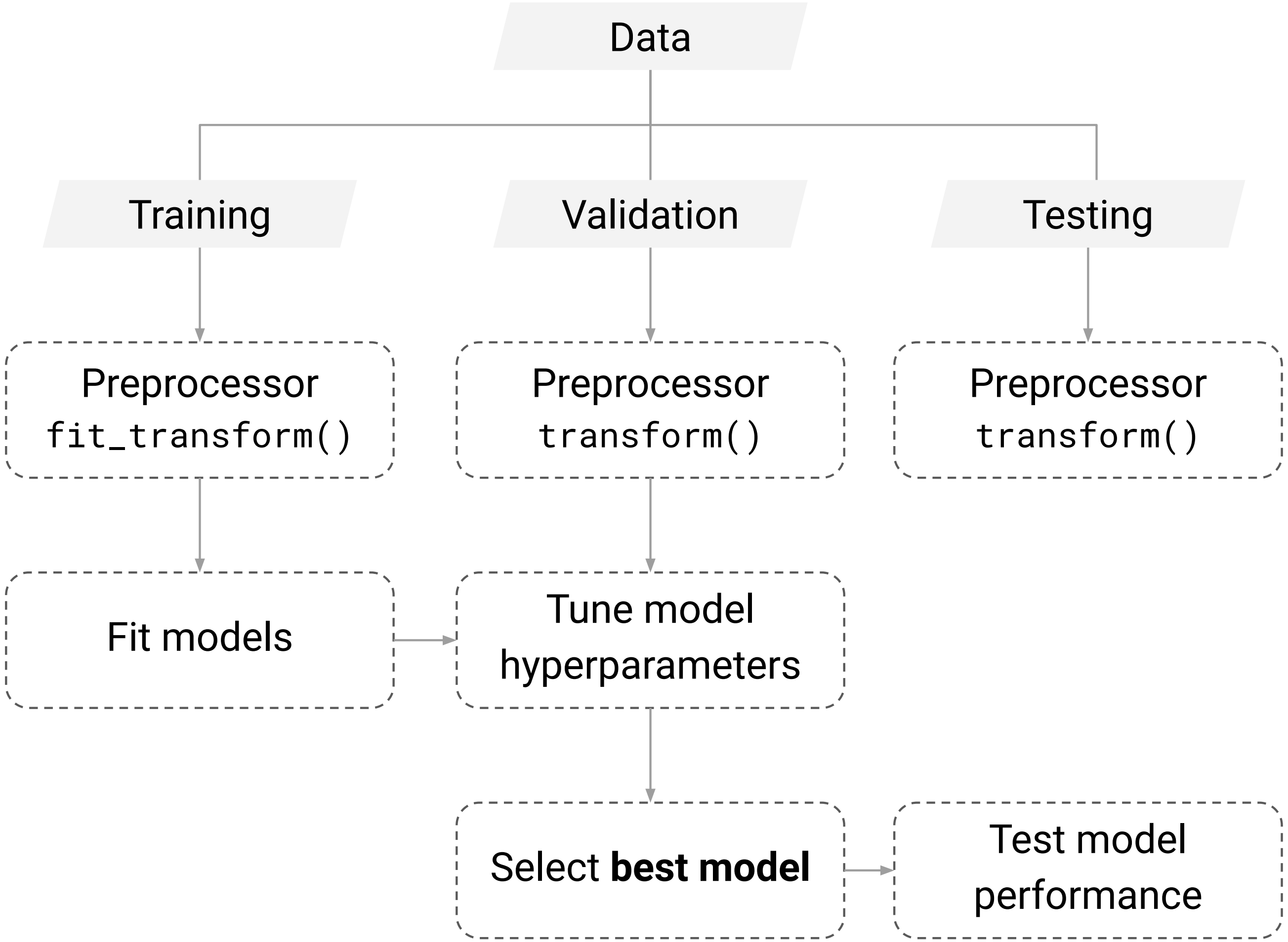
`fit` `fit_transform`

```python
scaler = StandardScaler()
scaler.fit(X_train)
X_train_sc = scaler.transform(X_train)
X_val_sc = scaler.transform(X_val)
X_test_sc = scaler.transform(X_test)

print(X_train_sc.mean(axis=0))
print(X_train_sc.std(axis=0))
print(X_val_sc.mean(axis=0))
print(X_val_sc.std(axis=0))
```

```
[ 2.97e-17 -2.18e-17  1.98e-17 -5.65e-17]
[1. 1. 1. 1.]
[-0.34  0.07 -0.27 -0.82]
[1.01 0.66 1.26 0.89]
```

Summary of the splitting



Numeric: StandardScaler()
Boolean: “passthrough”
Categorical: SimpleImputer() + OneHotEncoder()

Based on validation loss
(i.e. binary cross entropy)

Source: Melantha Wang (2022), ACTL3143 Project.

Dataframes & arrays

```
X_test.head(3)
```

	X1	X2	
83	0.075805	-0.677162	0.97512
53	0.954002	0.651391	-0.3152
70	0.113517	0.662131	1.58601

```
X_test_sc
```

```
array([[ 0.13, -0.64,  0.89, -0.4 ],
       [ 1.15,  0.67, -0.44,  0.62],
       [ 0.18,  0.68,  1.52, -1.62],
       [ 0.77, -0.82, -1.22,  0.31],
       [ 0.06,  1.46, -0.39,  2.83],
       [ 2.21,  0.49, -1.34,  0.51],
       [-0.57,  0.53, -0.02,  0.86],
       [ 0.16,  0.61, -0.96,  2.12],
       [ 0.9 ,  0.2 , -0.23, -0.57],
       [ 0.62, -0.11,  0.55,  1.48],
       [ 0.  ,  1.57, -2.81,  0.69],
       [ 0.96, -0.87,  1.33, -1.81],
       [-0.64,  0.87,  0.25, -1.01],
       [-1.19,  0.49, -1.06,  1.51],
       [ 0.65,  1.54, -0.23,  0.22],
       [-1.13,  0.34, -1.05, -1.82],
       [ 0.02,  0.14,  1.2 , -0.9 ],
       [ 0.68, -0.17, -0.34,  1.  ],
```

Note

By default, when you pass `sklearn` a DataFrame it returns a `numpy` array.

Keep as a DataFrame

From scikit-learn 1.2:

```python
from sklearn import set_config
set_config(transform_output="pandas")

imp = SimpleImputer()
imp.fit(X_train)
X_train_imp = imp.fit_transform(X_train)
X_val_imp = imp.transform(X_val)
X_test_imp = imp.transform(X_test)
```

```python
X_test_imp
```

	X1	X2	
83	0.075805	-0.677162	0.
53	0.954002	0.651391	-(
…	…	…	…
42	-0.245388	-0.753736	-(
69	0.199060	-0.600217	0.

25 rows × 4 columns

Lecture Outline

- Preprocessing
- **French Motor Claims Dataset**
- Ordinal Variables

French motor dataset

Download the dataset if we don’t have it already.

```python
from pathlib import Path
from sklearn.datasets import fetch_openml

if not Path("french-motor.csv").exists():
    freq = fetch_openml(data_id=41214, as_frame=True).frame
    freq.to_csv("french-motor.csv", index=False)
else:
    freq = pd.read_csv("french-motor.csv")

freq
```

French motor dataset

	IDpol	ClaimNb	Exposure	Area	VehPower	VehA
0	1.0	1.0	0.10000	D	5.0	0.0
1	3.0	1.0	0.77000	D	5.0	0.0
2	5.0	1.0	0.75000	B	6.0	2.0
...	...	...	...	...	...	...
678010	6114328.0	0.0	0.00274	D	6.0	2.0
678011	6114329.0	0.0	0.00274	B	4.0	0.0
678012	6114330.0	0.0	0.00274	B	7.0	6.0

678013 rows × 12 columns

Data dictionary

- `IDpol`: policy number (unique identifier)
- `ClaimNb`: number of claims on the given policy
- `Exposure`: total exposure in yearly units
- `Area`: area code (categorical, ordinal)
- `VehPower`: power of the car (categorical, ordinal)
- `VehAge`: age of the car in years
- `DrivAge`: age of the (most common) driver in years
- `BonusMalus`: bonus-malus level between 50 and 230 (with reference level 100)
- `VehBrand`: car brand (categorical, nominal)
- `VehGas`: diesel or regular fuel car (binary)
- `Density`: density of inhabitants per km^2 in the city of the living place of the driver
- `Region`: regions in France (prior to 2016)

Source: Nell et al. (2020), Case Study: French Motor Third-Party Liability Claims, SSRN.

The model

Have $\{(\mathbf{x}_i, y_i)\}_{i=1,\dots,n}$ for $\mathbf{x}_i \in \mathbb{R}^{47}$ and $y_i \in \mathbb{N}_0$.

Assume the distribution

$$Y_i \sim \mathsf{Poisson}(\lambda(\mathbf{x}_i))$$

We have $\mathbb{E}Y_i = \lambda(\mathbf{x}_i)$. The NN takes $\mathbf{x}_i$ & predicts $\mathbb{E}Y_i$.

UNSW SYDNEY

Lecture Outline

- Preprocessing
- French Motor Claims Dataset
- **Ordinal Variables**

Subsample and split

```python
freq = freq.drop("IDpol", axis=1).head(25_000)

X_train, X_test, y_train, y_test = train_test_split(
  freq.drop("ClaimNb", axis=1), freq["ClaimNb"], random_state=2023)

# Reset each index to start at 0 again.
X_train = X_train.reset_index(drop=True)
X_test = X_test.reset_index(drop=True)
```

What values do we see in the data?

```python
X_train["Area"].value_counts()
X_train["VehBrand"].value_counts()
X_train["VehGas"].value_counts()
X_train["Region"].value_counts()
```

```
Area
C    5507
D    4113
A    3527
E    2769
B    2359
F     475
Name: count, dtype: int64
```

```
VehBrand
B1      5069
B2      4838
B12     3708
        ...
B13      336
B11      284
B14      136
Name: count, Length: 11, dtype: int64
```

```
VehGas
Regular    10773
Diesel      7977
Name: count, dtype: int64
```

```
Region
R24     6498
R82     2119
R11     1909
        ...
R21       90
R42       55
R43       26
Name: count, Length: 22, dtype: int64
```

Ordinal & binary categories are easy

```python
from sklearn.preprocessing import OrdinalEncoder
oe = OrdinalEncoder()
oe.fit(X_train[["Area", "VehGas"]])
oe.categories_
```

```
[array(['A', 'B', 'C', 'D', 'E', 'F'], dtype=object),
 array(['Diesel', 'Regular'], dtype=object)]
```

```python
for i, area in enumerate(oe.categories_[0]):
    print(f"The Area value {area} gets turned into {i}.")
```

```
The Area value A gets turned into 0.
The Area value B gets turned into 1.
The Area value C gets turned into 2.
The Area value D gets turned into 3.
The Area value E gets turned into 4.
The Area value F gets turned into 5.
```

```python
for i, gas in enumerate(oe.categories_[1]):
    print(f"The VehGas value {gas} gets turned into {i}.")
```

```
The VehGas value Diesel gets turned into 0.
The VehGas value Regular gets turned into 1.
```

Ordinal encoded values

```
X_train_ord = oe.transform(X_train[["Area", "VehGas"]])
X_test_ord = oe.transform(X_test[["Area", "VehGas"]])
```

```
X_train[["Area", "VehGas"]].head()
```

	Area	VehGas
0	C	Diesel
1	C	Regular
2	E	Regular
3	D	Diesel
4	A	Regular

```
X_train_ord.head()
```

	Area	VehGas
0	2.0	0.0
1	2.0	1.0
2	4.0	1.0
3	3.0	0.0
4	0.0	1.0

Train on ordinal encoded values

```python
random.seed(12)
model = Sequential([
  Dense(1, activation="exponential")
])

model.compile(optimizer="adam", loss="poisson")

es = EarlyStopping(verbose=True)
hist = model.fit(X_train_ord, y_train, epochs=100, verbose=0,
    validation_split=0.2, callbacks=[es])
hist.history["val_loss"][-1]
```

```
Epoch 22: early stopping
```

```
0.7821308970451355
```

What about adding the continuous variables back in? Use a sklearn *column transformer* for that.

Preprocess ordinal & continuous

```python
from sklearn.compose import make_column_transformer

ct = make_column_transformer(
  (OrdinalEncoder(), ["Area", "VehGas"]),
  ("drop", ["VehBrand", "Region"]),
  remainder=StandardScaler()
)

X_train_ct = ct.fit_transform(X_train)
```

```python
X_train.head(3)
```

	Exposure	Area	VehPower
0	1.00	C	6.0
1	0.36	C	4.0
2	0.02	E	12.0

```python
X_train_ct.head(3)
```

	ordinalencoder__Area	ordi
0	2.0	0.0
1	2.0	1.0
2	4.0	1.0

Preprocess ordinal & continuous II

```
from sklearn.compose import make_column_transformer

ct = make_column_transformer(
  (OrdinalEncoder(), ["Area", "VehGas"]),
  ("drop", ["VehBrand", "Region"]),
  remainder=StandardScaler(),
  verbose_feature_names_out=False
)
X_train_ct = ct.fit_transform(X_train)
```

```
X_train.head(3)
```

	Exposure	Area	VehPower
0	1.00	C	6.0
1	0.36	C	4.0
2	0.02	E	12.0

```
X_train_ct.head(3)
```

	Area	VehGas	Exposure
0	2.0	0.0	1.126979
1	2.0	1.0	-0.590896
2	4.0	1.0	-1.503517

Glossary

- column transformer
- nominal variables
- ordinal variables