

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

```
1 random.seed(12)
2 model = Sequential()
3 model.add(Dense(1, activation="relu"))
4 model.compile("adam", "poisson")
5 model.fit(X_train, y_train, verbose=0)
6 y_pred = model.predict(X_val, verbose=0)
7 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

```
1 model = LinearRegression()  
2 model.fit(X_train, y_train)  
3 y_pred = model.predict(X_val)  
4 print(model.score(X_val, y_val))
```

-0.666850597951445



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
-----	---------------

```

1 scaler = StandardScaler()
2 scaler.fit(X_train)
3 X_train_sc = scaler.transform(X_train)
4 X_val_sc = scaler.transform(X_val)
5 X_test_sc = scaler.transform(X_test)
6
7 print(X_train_sc.mean(axis=0))
8 print(X_train_sc.std(axis=0))
9 print(X_val_sc.mean(axis=0))
10 print(X_val_sc.std(axis=0))

```

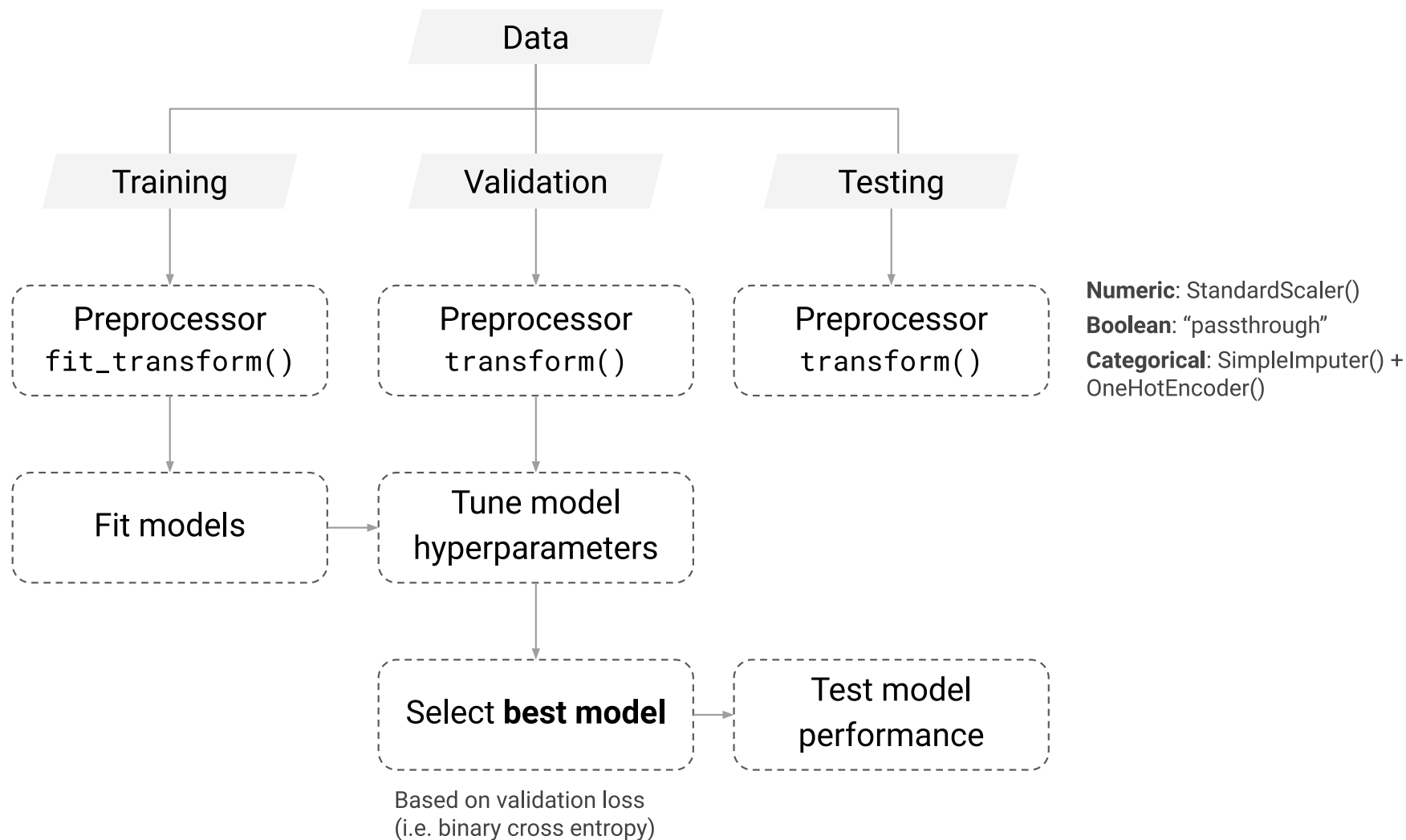
```

[ 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



Source: Melantha Wang (2022), ACTL3143 Project.



Dataframes & arrays

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

	X1	X2	
83	0.075805	-0.677162	0.9751
53	0.954002	0.651391	-0.315
70	0.113517	0.662131	1.5860

```
1 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. ],
       [ 0.44, -1.72,  0.22, -0.66],
```

Note

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



Keep as a DataFrame

From **scikit-learn 1.2**:

```
1 from sklearn import set_config
2 set_config(transform_output="pandas")
3
4 imp = SimpleImputer()
5 imp.fit(X_train)
6 X_train_imp = imp.fit_transform(X_train)
7 X_val_imp = imp.transform(X_val)
8 X_test_imp = imp.transform(X_test)
```

```
1 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.

```
1 from pathlib import Path
2 from sklearn.datasets import fetch_openml
3
4 if not Path("french-motor.csv").exists():
5     freq = fetch_openml(data_id=41214, as_frame=True).frame
6     freq.to_csv("french-motor.csv", index=False)
7 else:
8     freq = pd.read_csv("french-motor.csv")
9
10 freq
```



French motor dataset

	IDpol	ClaimNb	Exposure	Area	VehPower	VehAge
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 km² in the city of the living place of the driver
- **Region**: regions in France (prior to 2016)



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

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

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



Lecture Outline

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



Subsample and split

```
1 freq = freq.drop("IDpol", axis=1).head(25_000)
2
3 X_train, X_test, y_train, y_test = train_test_split(
4     freq.drop("ClaimNb", axis=1), freq["ClaimNb"], random_state=2023)
5
6 # Reset each index to start at 0 again.
7 X_train = X_train.reset_index(drop=True)
8 X_test = X_test.reset_index(drop=True)
```



What values do we see in the data?

```
1 X_train["Area"].value_counts()
2 X_train["VehBrand"].value_counts()
3 X_train["VehGas"].value_counts()
4 X_train["Region"].value_counts()
```

Area

```
C    5507
D    4113
A    3527
E    2769
B    2359
F     475
```

Name: count, dtype: int64

VehGas

```
Regular    10773
Diesel      7977
```

Name: count, dtype: int64

VehBrand

```
B1    5069
B2    4838
B12   3708
...
B13    336
B11    284
B14    136
```

Name: count, Length: 11, 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

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

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

```
1 for i, area in enumerate(oe.categories_[0]):
2     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.

```
1 for i, gas in enumerate(oe.categories_[1]):
2     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

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

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

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

```
1 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

```
1 random.seed(12)
2 model = Sequential([
3     Dense(1, activation="exponential")
4 ])
5
6 model.compile(optimizer="adam", loss="poisson")
7
8 es = EarlyStopping(verbose=True)
9 hist = model.fit(X_train_ord, y_train, epochs=100, verbose=0,
10                 validation_split=0.2, callbacks=[es])
11 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

```

1 from sklearn.compose import make_column_transformer
2
3 ct = make_column_transformer(
4     (OrdinalEncoder(), ["Area", "VehGas"]),
5     ("drop", ["VehBrand", "Region"]),
6     remainder=StandardScaler()
7 )
8
9 X_train_ct = ct.fit_transform(X_train)

```

```
1 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

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

	ordinalencoder__Area	ord
0	2.0	0.0
1	2.0	1.0
2	4.0	1.0



Preprocess ordinal & continuous II

```

1 from sklearn.compose import make_column_transformer
2
3 ct = make_column_transformer(
4     (OrdinalEncoder(), ["Area", "VehGas"]),
5     ("drop", ["VehBrand", "Region"]),
6     remainder=StandardScaler(),
7     verbose_feature_names_out=False
8 )
9 X_train_ct = ct.fit_transform(X_train)

```

```
1 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

```
1 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

