Classification

ACTL3143 & ACTL5111 Deep Learning for Actuaries

Patrick Laub

Lecture Outline

- **Classification**
- Stroke Prediction

Iris dataset

```python
from sklearn.datasets import load_iris
iris = load_iris()
names = ["SepalLength", "SepalWidth", "PetalLength", "PetalWidth"]
features = pd.DataFrame(iris.data, columns = names)
features
```

	SepalLength	SepalWidth	PetalLength	PetalWidth
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
…	…	…	…	…
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

Target variable

```python
iris.target_names
```

```
array(['setosa', 'versicolor', 'virginica'],
dtype='<U10')
```

```python
iris.target[:8]
```

```
array([0, 0, 0, 0, 0, 0, 0, 0])
```

```python
target = iris.target
target = target.reshape(-1, 1)
target[:8]
```

```
array([[0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0]])
```

```python
classes, counts = np.unique(
        target,
        return_counts=True
)
print(classes)
print(counts)
```

```
[0 1 2]
[50 50 50]
```

```python
iris.target_names[
  target[[0, 30, 60]]
]
```

```
array([['setosa'],
       ['setosa'],
       ['versicolor']], dtype='<U10')
```

Split the data into train and test

```
X_train, X_test, y_train, y_test = train_test_split(features, target, random_state=24)
X_train
```

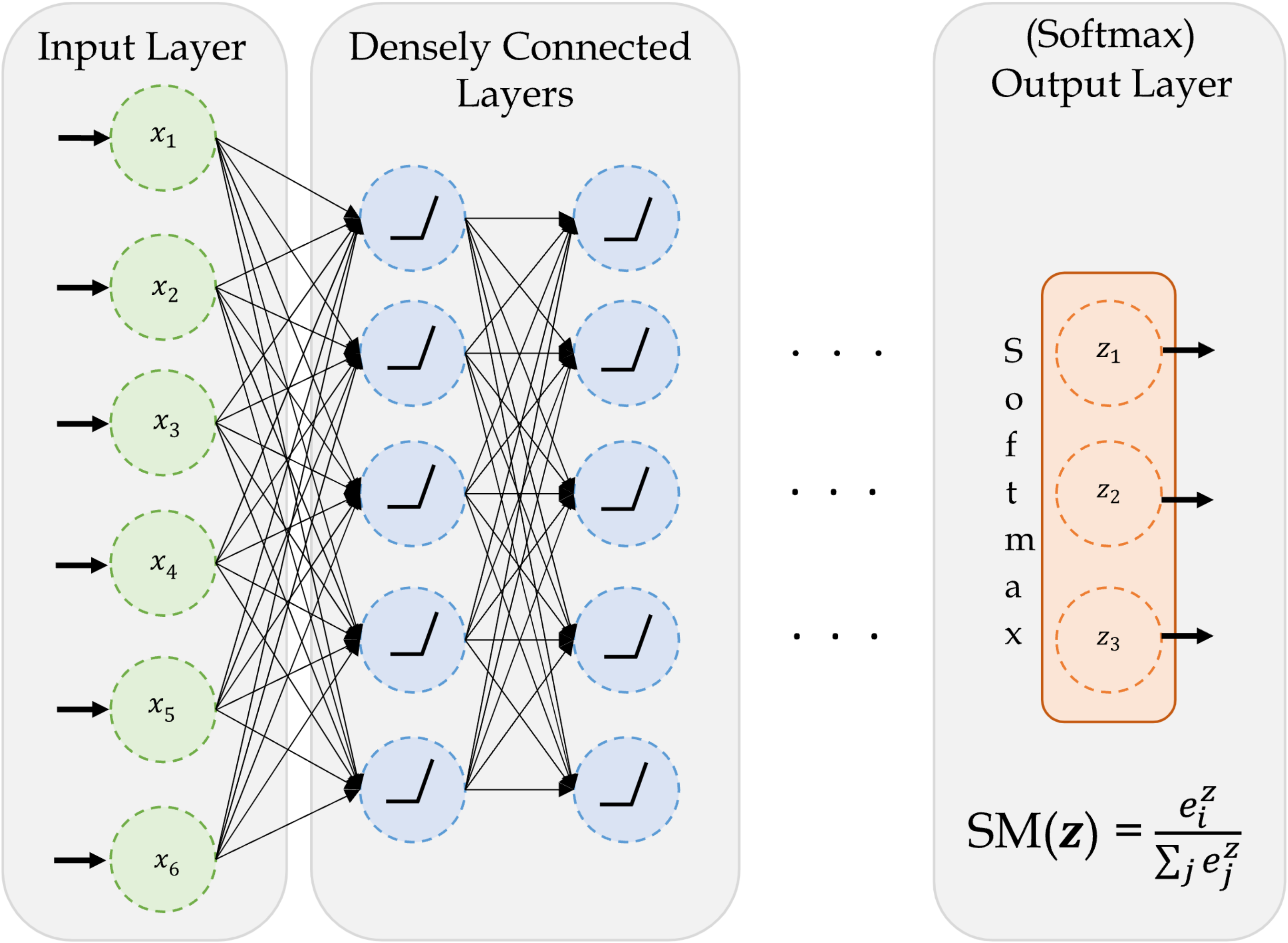
	SepalLength	SepalWidth	PetalLength	PetalWidth
53	5.5	2.3	4.0	1.3
58	6.6	2.9	4.6	1.3
95	5.7	3.0	4.2	1.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
87	6.3	2.3	4.4	1.3
131	7.9	3.8	6.4	2.0

112 rows × 4 columns

```
X_test.shape, y_test.shape
```

```
((38, 4), (38, 1))
```

A basic classifier network



A basic network for classifying into three categories.

Source: Marcus Lautier (2022).

Create a classifier model

```python
NUM_FEATURES = len(features.columns)
NUM_CATS = len(np.unique(target))

print("Number of features:", NUM_FEATURES)
print("Number of categories:", NUM_CATS)
```

```
Number of features: 4
Number of categories: 3
```

Make a function to return a Keras model:

```python
def build_model(seed=42):
    random.seed(seed)
    return Sequential([
        Dense(30, activation="relu"),
        Dense(NUM_CATS, activation="softmax")
    ])
```

Fit the model

```
model = build_model()
model.compile("adam", "sparse_categorical_crossentropy")

model.fit(X_train, y_train, epochs=5, verbose=2);
```

```
Epoch 1/5
4/4 - 0s - 1ms/step - loss: 1.3870
Epoch 2/5
4/4 - 0s - 1ms/step - loss: 1.2943
Epoch 3/5
4/4 - 0s - 1ms/step - loss: 1.2189
Epoch 4/5
4/4 - 0s - 988us/step - loss: 1.1581
Epoch 5/5
4/4 - 0s - 975us/step - loss: 1.1079
```

Track accuracy as the model trains

```python
model = build_model()
model.compile("adam", "sparse_categorical_crossentropy", \
        metrics=["accuracy"])
model.fit(X_train, y_train, epochs=5, verbose=2);
```

```
Epoch 1/5
4/4 - 0s - 1ms/step - accuracy: 0.2857 - loss: 1.3923
Epoch 2/5
4/4 - 0s - 1ms/step - accuracy: 0.2857 - loss: 1.2910
Epoch 3/5
4/4 - 0s - 1ms/step - accuracy: 0.2857 - loss: 1.2179
Epoch 4/5
4/4 - 0s - 1ms/step - accuracy: 0.3036 - loss: 1.1603
Epoch 5/5
4/4 - 0s - 1ms/step - accuracy: 0.4286 - loss: 1.1095
```

Run a long fit

```python
model = build_model()
model.compile("adam", "sparse_categorical_crossentropy", \
        metrics=["accuracy"])
%time hist = model.fit(X_train, y_train, epochs=500, \
        validation_split=0.25, verbose=False)
```

```
CPU times: user 1.9 s, sys: 7.11 ms, total: 1.9 s
Wall time: 1.94 s
```

Evaluation now returns both *loss* and *accuracy*.

```python
model.evaluate(X_test, y_test, verbose=False)
```

```
[0.08759287744760513, 0.9736841917037964]
```

Add early stopping

```
model = build_model()
model.compile("adam", "sparse_categorical_crossentropy", \
        metrics=["accuracy"])

es = EarlyStopping(restore_best_weights=True, patience=50,
        monitor="val_accuracy")
%time hist_es = model.fit(X_train, y_train, epochs=500, \
        validation_split=0.25, callbacks=[es], verbose=False);

print(f"Stopped after {len(hist_es.history['loss'])} epochs.")
```

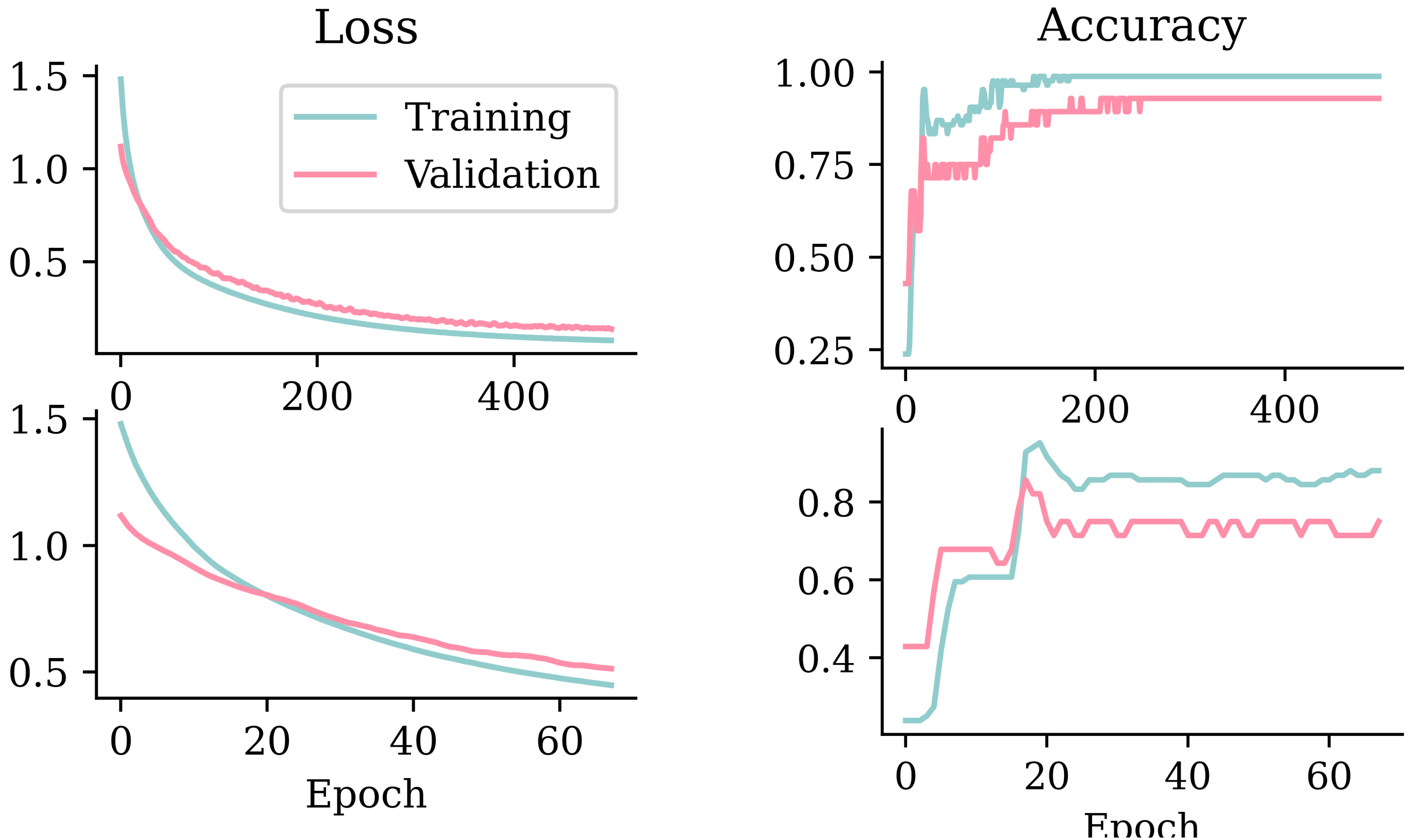
```
CPU times: user 265 ms, sys: 857 µs, total: 266 ms
Wall time: 267 ms
Stopped after 68 epochs.
```

Evaluation on test set:

```
model.evaluate(X_test, y_test, verbose=False)
```

```
[0.8151808977127075, 0.9473684430122375]
```

Fitting metrics



What is the softmax activation?

It creates a "probability" vector: $\text{Softmax}(\boldsymbol{x}) = \frac{\mathrm{e}^{x}_{i}}{\sum_j \mathrm{e}^{x}_{j}}$.

In NumPy:

```python
out = np.array([5, -1, 6])
(np.exp(out) / np.exp(out).sum()).round(3)
```

```
array([0.269, 0.001, 0.731])
```

In Keras:

```python
out = keras.ops.convert_to_tensor([[5.0, -1.0, 6.0]])
keras.ops.round(keras.ops.softmax(out), 3)
```

```
tensor([[0.2690, 0.0010, 0.7310]])
```

Prediction using classifiers

```python
y_test[:4]
```

```
array([[2],
       [2],
       [1],
       [1]])
```

```python
y_pred = model.predict(X_test.head(4), verbose=0)
y_pred
```

```
array([[0.16517346, 0.41306466, 0.4217618 ],
       [0.1688867 , 0.3096904 , 0.5214229 ],
       [0.279566  , 0.4134798 , 0.30695418],
       [0.2462524 , 0.38838378, 0.3653639 ]], dtype=float32)
```

```python
# Add 'keepdims=True' to get a column vector.
np.argmax(y_pred, axis=1)
```

```
array([2, 2, 1, 1])
```

```python
iris.target_names[np.argmax(y_pred, axis=1)]
```

```
array(['virginica', 'virginica', 'versicolor', 'versicolor'], dtype='<U10')
```

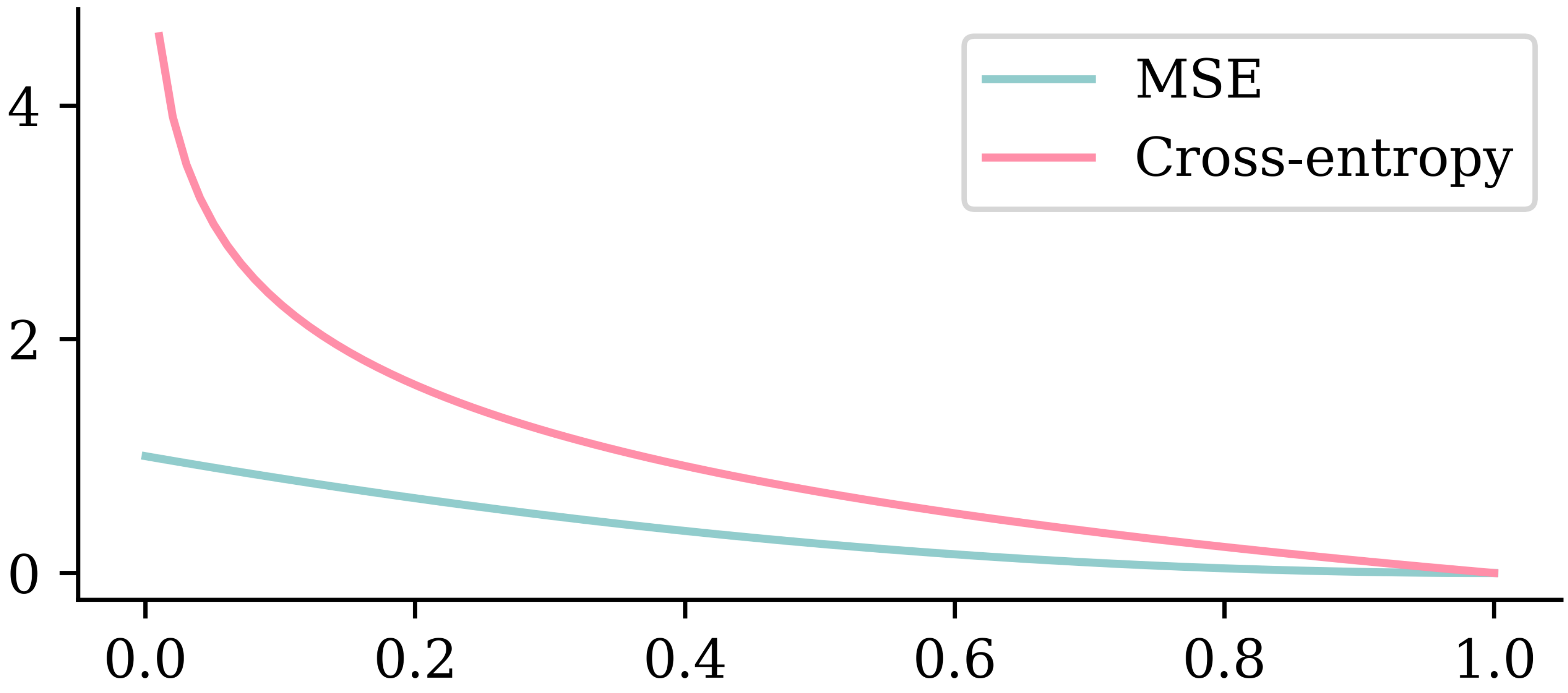
Cross-entropy loss: ELI5

Neural Networks Part 6: Cross Entropy

Neural Networks Part 7: Cross Entropy Derivativ…

Why use cross-entropy loss?

```python
p = np.linspace(0, 1, 100)
plt.plot(p, (1-p)**2)
plt.plot(p, -np.log(p))
plt.legend(["MSE", "Cross-entropy"]);
```



One-hot encoding

```python
from sklearn.preprocessing import OneHotEncoder

enc = OneHotEncoder(sparse_output=False)

y_train_oh = enc.fit_transform(y_train)
y_test_oh = enc.transform(y_test)
```

```python
y_train[:5]
```

```
array([[1],
       [1],
       [1],
       [0],
       [0]])
```

```python
y_train_oh[:5]
```

	x0_0	x0_1	x0_2
0	0.0	1.0	0.0
1	0.0	1.0	0.0
2	0.0	1.0	0.0
3	1.0	0.0	0.0
4	1.0	0.0	0.0

Classifier given one-hot outputs

Create the model (*new loss function*):

```
model = build_model()
model.compile("adam", "categorical_crossentropy", \
    metrics=["accuracy"])
```

Fit the model (*new target variables*):

```
model.fit(X_train, y_train_oh, epochs=100, verbose=False);
```

Evaluate the model (*new target variables*):

```
model.evaluate(X_test, y_test_oh, verbose=False)
```

```
[0.29025211930274963, 0.9473684430122375]
```

Lecture Outline

- Classification
- **Stroke Prediction**

The data

Dataset source: Kaggle Stroke Prediction Dataset.

```
data = pd.read_csv("stroke.csv")
data.head()
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residenc
0	9046	Male	67.0	0	1	Yes	Private	Urban
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural
2	31112	Male	80.0	0	1	Yes	Private	Rural
3	60182	Female	49.0	0	0	Yes	Private	Urban
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural

Data description

1. `id`: unique identifier
2. `gender`: “Male”, “Female” or “Other”
3. `age`: age of the patient
4. `hypertension`: 0 or 1 if the patient has hypertension
5. `heart_disease`: 0 or 1 if the patient has any heart disease
6. `ever_married`: “No” or “Yes”
7. `work_type`: “children”, “Govt_jov”, “Never_worked”, “Private” or “Self-employed”
8. `Residence_type`: “Rural” or “Urban”
9. `avg_glucose_level`: average glucose level in blood
10. `bmi`: body mass index
11. `smoking_status`: “formerly smoked”, “never smoked”, “smokes” or “Unknown”
12. `stroke`: 0 or 1 if the patient had a stroke

Source: Kaggle, Stroke Prediction Dataset.

Split the data

First, look for missing values.

```python
number_missing = data.isna().sum()
number_missing[number_missing > 0]
```

```
bmi      201
dtype: int64
```

```python
features = data.drop(["id", "stroke"], axis=1)
target = data["stroke"]

X_main, X_test, y_main, y_test = train_test_split(
    features, target, test_size=0.2, random_state=7)
X_train, X_val, y_train, y_val = train_test_split(
    X_main, y_main, test_size=0.25, random_state=12)

X_train.shape, X_val.shape, X_test.shape
```

```
((3066, 10), (1022, 10), (1022, 10))
```

What values do we see in the data?

```python
X_train["gender"].value_counts()
```

```
gender
Female    1802
Male      1264
Name: count, dtype: int64
```

```python
X_train["ever_married"].value_counts()
```

```
ever_married
Yes    2007
No     1059
Name: count, dtype: int64
```

```python
X_train["Residence_type"].value_counts(
```

```
Residence_type
Urban    1536
Rural    1530
Name: count, dtype: int64
```

```python
X_train["work_type"].value_counts()
```

```
work_type
Private          1754
Self-employed     490
children          419
Govt_job          390
Never_worked       13
Name: count, dtype: int64
```

```python
X_train["smoking_status"].value_counts(
```

```
smoking_status
never smoked       1130
Unknown             944
formerly smoked     522
smokes              470
Name: count, dtype: int64
```

Preprocess columns individually

1. Take categorical columns $\hookrightarrow$ one-hot vectors
2. binary columns $\hookrightarrow$ do nothing
3. continuous columns $\hookrightarrow$ impute NaNs & standardise.

Scikit-learn column transformer

```python
from sklearn.pipeline import make_pipeline

cat_vars =  ["gender", "ever_married", "Residence_type",
    "work_type", "smoking_status"]

ct = make_column_transformer(
  (OneHotEncoder(sparse_output=False, handle_unknown="ignore"), cat_vars),
  ("passthrough", ["hypertension", "heart_disease"]),
  remainder=make_pipeline(SimpleImputer(), StandardScaler()),
  verbose_feature_names_out=False
)

X_train_ct = ct.fit_transform(X_train)
X_val_ct = ct.transform(X_val)
X_test_ct = ct.transform(X_test)

for name, X in zip(("train", "val", "test"), (X_train_ct, X_val_ct, X_test_ct)):
    num_na = X.isna().sum().sum()
    print(f"The {name} set has shape {X_train_ct.shape} & with {num_na} NAs.")
```

```
The train set has shape (3066, 20) & with 0 NAs.
The val set has shape (3066, 20) & with 0 NAs.
The test set has shape (3066, 20) & with 0 NAs.
```

Handling unseen categories

```
X_train["gender"].value_counts()
```

```
gender
Female    1802
Male      1264
Name: count, dtype: int64
```

```
X_val["gender"].value_counts()
```

```
gender
Female    615
Male      406
Other       1
Name: count, dtype: int64
```

```
ind = np.argmax(X_val["gender"] == "Oth
X_val.iloc[ind-1:ind+3][["gender"]]
```

	gender
4970	Male
3116	Other
4140	Male
2505	Female

```
gender_cols = X_val_ct[["gender_Female"
gender_cols.iloc[ind-1:ind+3]
```

	gender_Female	gender_
4970	0.0	1.0
3116	0.0	0.0
4140	0.0	1.0
2505	1.0	0.0

Setup a binary classification model

```python
def create_model(seed=42):
    random.seed(seed)
    model = Sequential()
    model.add(Input(X_train_ct.shape[1:]))
    model.add(Dense(32, "leaky_relu"))
    model.add(Dense(16, "leaky_relu"))
    model.add(Dense(1, "sigmoid"))
    return model
```

```python
model = create_model()
model.summary()
```

Model: "sequential_5"

Layer (type)	Output Shape	Param #
dense_10 (Dense)	(None, 32)	672
dense_11 (Dense)	(None, 16)	528
dense_12 (Dense)	(None, 1)	17

Total params: 1,217 (4.75 KB)
Trainable params: 1,217 (4.75 KB)
Non-trainable params: 0 (0.00 B)

Add metrics, compile, and fit

```
model = create_model()

pr_auc = keras.metrics.AUC(curve="PR", name="pr_auc")
model.compile(optimizer="adam", loss="binary_crossentropy",
    metrics=[pr_auc, "accuracy", "auc"])

es = EarlyStopping(patience=50, restore_best_weights=True,
    monitor="val_pr_auc", verbose=1)
model.fit(X_train_ct, y_train, callbacks=[es], epochs=1_000, verbose=0,
  validation_data=(X_val_ct, y_val));
```

```
Epoch 69: early stopping
Restoring model weights from the end of the best epoch: 19.
```

```
model.evaluate(X_val_ct, y_val, verbose
```

```
[0.1450638473033905,
 0.13280197978019714,
 0.9589040875434875,
 0.8276846408843994]
```

Overweight the minority class

```
model = create_model()

pr_auc = keras.metrics.AUC(curve="PR", name="pr_auc")
model.compile(optimizer="adam", loss="binary_crossentropy",
    metrics=[pr_auc, "accuracy", "auc"])

es = EarlyStopping(patience=50, restore_best_weights=True,
    monitor="val_pr_auc", verbose=1)
model.fit(X_train_ct, y_train.to_numpy(), callbacks=[es], epochs=1_000, verbose=0,
  validation_data=(X_val_ct, y_val), class_weight={0: 1, 1: 10});
```

```
Epoch 56: early stopping
Restoring model weights from the end of the best epoch: 6.
```

```
model.evaluate(X_val_ct, y_val, verbose
```

```
[0.379545658826828, 0.1360630989074707,
0.767123281955719, 0.8222789168357849]
```

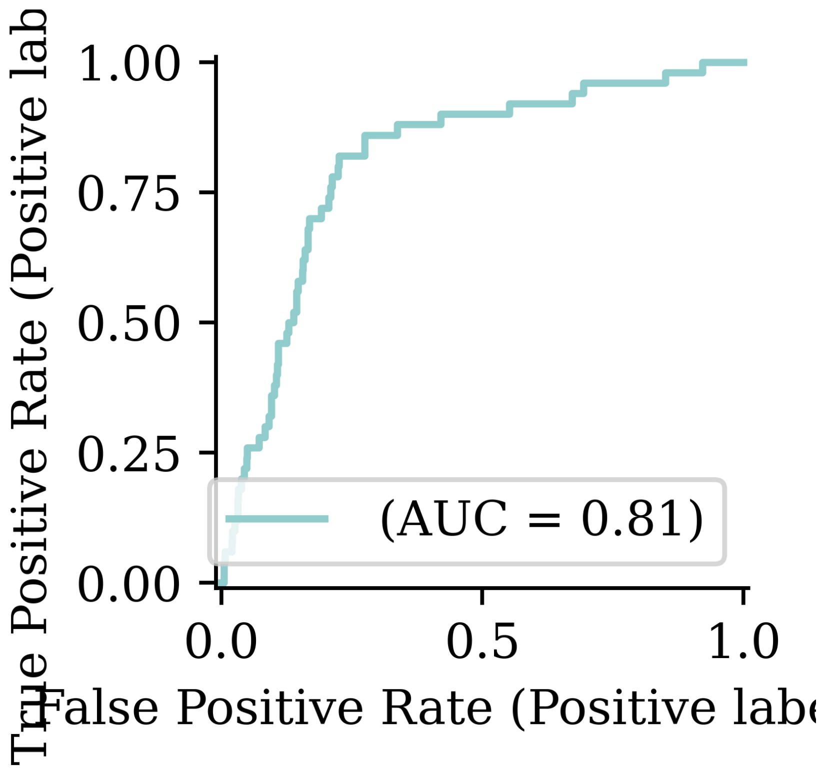
```
model.evaluate(X_test_ct, y_test, verbo
```

```
[0.3905954957008362,
 0.15662002563476562,
 0.7837573289871216,
 0.8110082149505615]
```

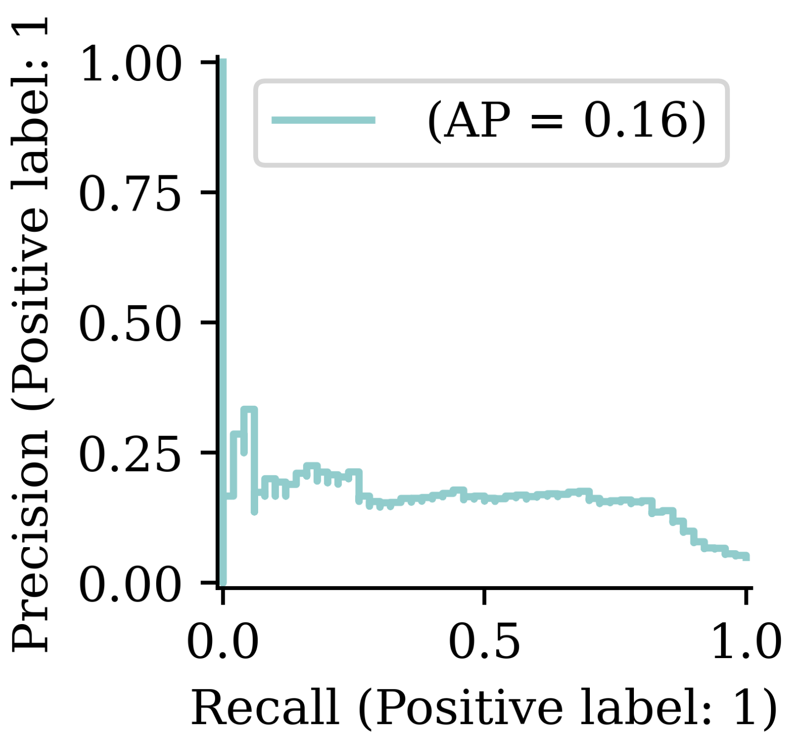
Classification Metrics

```
from sklearn.metrics import confusion_matrix, RocCurveDisplay, PrecisionRecallDisplay
y_pred = model.predict(X_test_ct, verbose=0)
```

```
RocCurveDisplay.from_predictions(y_test, y_pred, name="");
```




```
PrecisionRecallDisplay.from_predictions(y_test, y_pred, na
```




```
y_pred_stroke = y_pred > 0.5
confusion_matrix(y_test, y_pred_stroke)
```

```
array([[762, 210],
       [ 11,  39]])
```

```
y_pred_stroke = y_pred > 0.3
confusion_matrix(y_test, y_pred_stroke)
```

```
array([[629, 343],
       [  6,  44]])
```

Package Versions

```python
from watermark import watermark
print(watermark(python=True, packages="keras,matplotlib,numpy,pandas,seaborn,scipy,torch
```

```
Python implementation: CPython
Python version       : 3.11.8
IPython version      : 8.23.0

keras     : 3.2.0
matplotlib: 3.8.4
numpy     : 1.26.4
pandas    : 2.2.1
seaborn   : 0.13.2
scipy     : 1.11.0
torch     : 2.2.2
tensorflow: 2.16.1
tf_keras  : 2.16.0
```

Glossary

- classification problem
- confusion matrix
- cross-entropy loss
- sigmoid activation function