

Classification

ACTL3143 & ACTL5111 Deep Learning for Actuaries
Patrick Laub



Lecture Outline

- **Classification**
- Stroke Prediction



Iris dataset

```

1 from sklearn.datasets import load_iris
2 iris = load_iris()
3 names = ["SepalLength", "SepalWidth", "PetalLength", "PetalWidth"]
4 features = pd.DataFrame(iris.data, columns = names)
5 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

```
1 iris.target_names
```

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

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

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

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

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

```
1 classes, counts = np.unique(
2     target,
3     return_counts=True
4 )
5 print(classes)
6 print(counts)
```

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

```
1 iris.target_names[
2     target[[0, 30, 60]]
3 ]
```

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



Split the data into train and test

```
1 X_train, X_test, y_train, y_test = train_test_split(features, target, random_state=24)
2 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 |

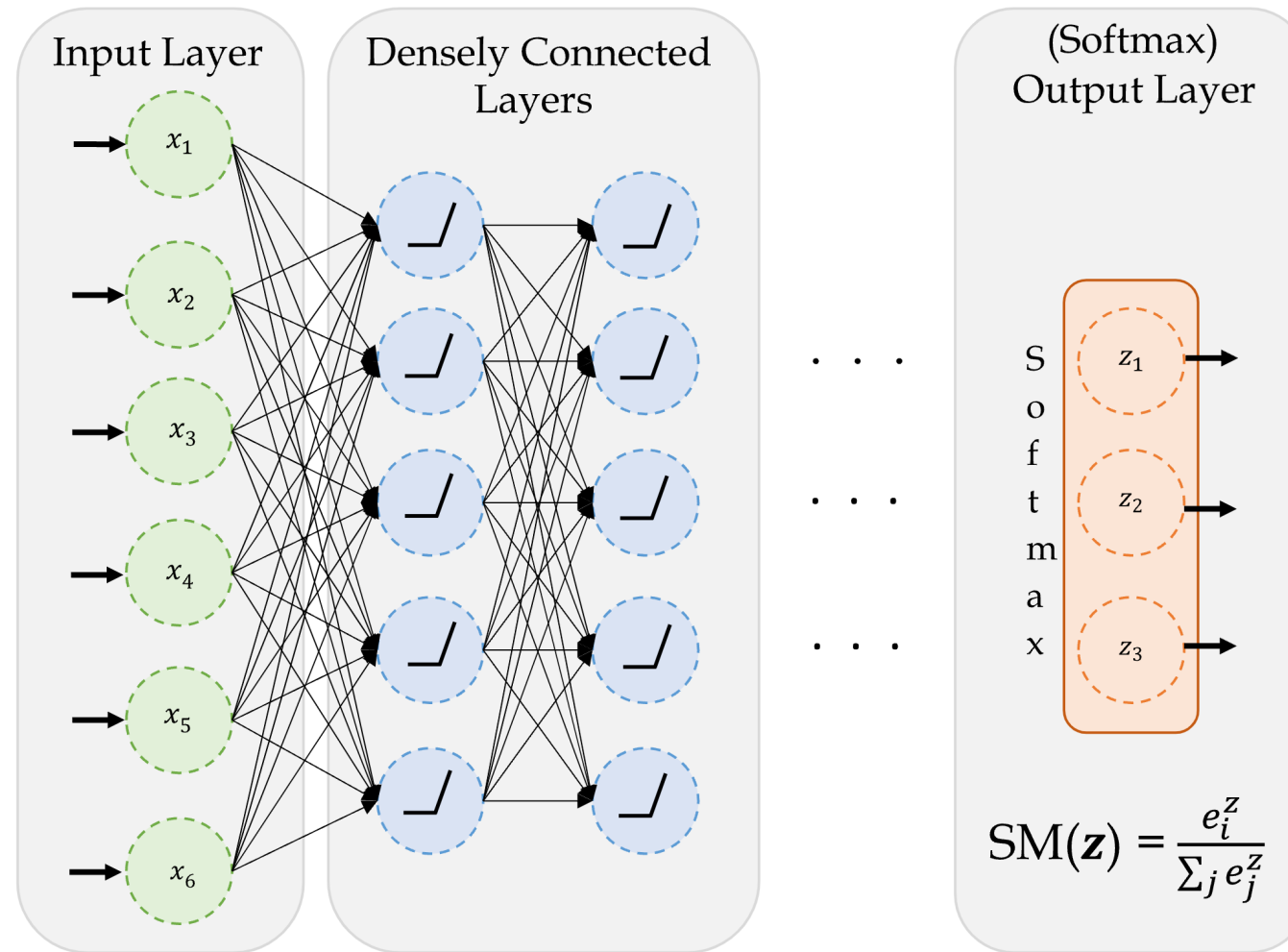
112 rows \times 4 columns

```
1 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

```
1 NUM_FEATURES = len(features.columns)
2 NUM_CATS = len(np.unique(target))
3
4 print("Number of features:", NUM_FEATURES)
5 print("Number of categories:", NUM_CATS)
```

Number of features: 4
Number of categories: 3

Make a function to return a Keras model:

```
1 def build_model(seed=42):
2     random.seed(seed)
3     return Sequential([
4         Dense(30, activation="relu"),
5         Dense(NUM_CATS, activation="softmax")
6     ])
```



Fit the model

```
1 model = build_model()  
2 model.compile("adam", "sparse_categorical_crossentropy")  
3  
4 model.fit(X_train, y_train, epochs=5, verbose=2);
```

```
Epoch 1/5  
4/4 - 0s - 123ms/step - loss: 1.3502  
Epoch 2/5  
4/4 - 0s - 4ms/step - loss: 1.2852  
Epoch 3/5  
4/4 - 0s - 5ms/step - loss: 1.2337  
Epoch 4/5  
4/4 - 0s - 4ms/step - loss: 1.1915  
Epoch 5/5  
4/4 - 0s - 4ms/step - loss: 1.1556
```



Track accuracy as the model trains

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

Epoch 1/5

4/4 - 1s - 137ms/step - accuracy: 0.2946 - loss: 1.3502

Epoch 2/5

4/4 - 0s - 5ms/step - accuracy: 0.3036 - loss: 1.2852

Epoch 3/5

4/4 - 0s - 6ms/step - accuracy: 0.3036 - loss: 1.2337

Epoch 4/5

4/4 - 0s - 15ms/step - accuracy: 0.3304 - loss: 1.1915

Epoch 5/5

4/4 - 0s - 5ms/step - accuracy: 0.3393 - loss: 1.1556



Run a long fit

```
1 model = build_model()  
2 model.compile("adam", "sparse_categorical_crossentropy", \  
3     metrics=["accuracy"])  
4 %time hist = model.fit(X_train, y_train, epochs=500, \  
5     validation_split=0.25, verbose=False)
```

CPU times: user 17.7 s, sys: 2.17 s, total: 19.9 s
Wall time: 17.9 s

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

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

[0.09586220979690552, 0.9736841917037964]



Add early stopping

```

1 model = build_model()
2 model.compile("adam", "sparse_categorical_crossentropy", \
3               metrics=["accuracy"])
4
5 es = EarlyStopping(restore_best_weights=True, patience=50,
6                   monitor="val_accuracy")
7 %time hist_es = model.fit(X_train, y_train, epochs=500, \
8                           validation_split=0.25, callbacks=[es], verbose=False);
9
10 print(f"Stopped after {len(hist_es.history['loss'])} epochs.")

```

CPU times: user 2.97 s, sys: 349 ms, total: 3.32 s
 Wall time: 3.02 s
 Stopped after 68 epochs.

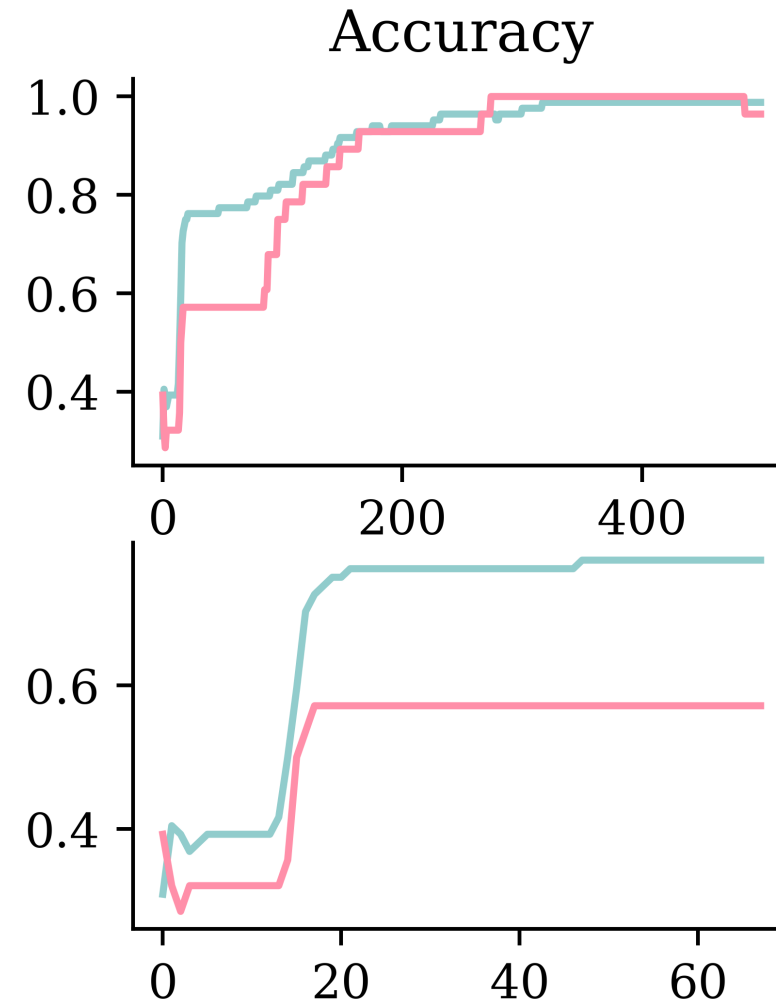
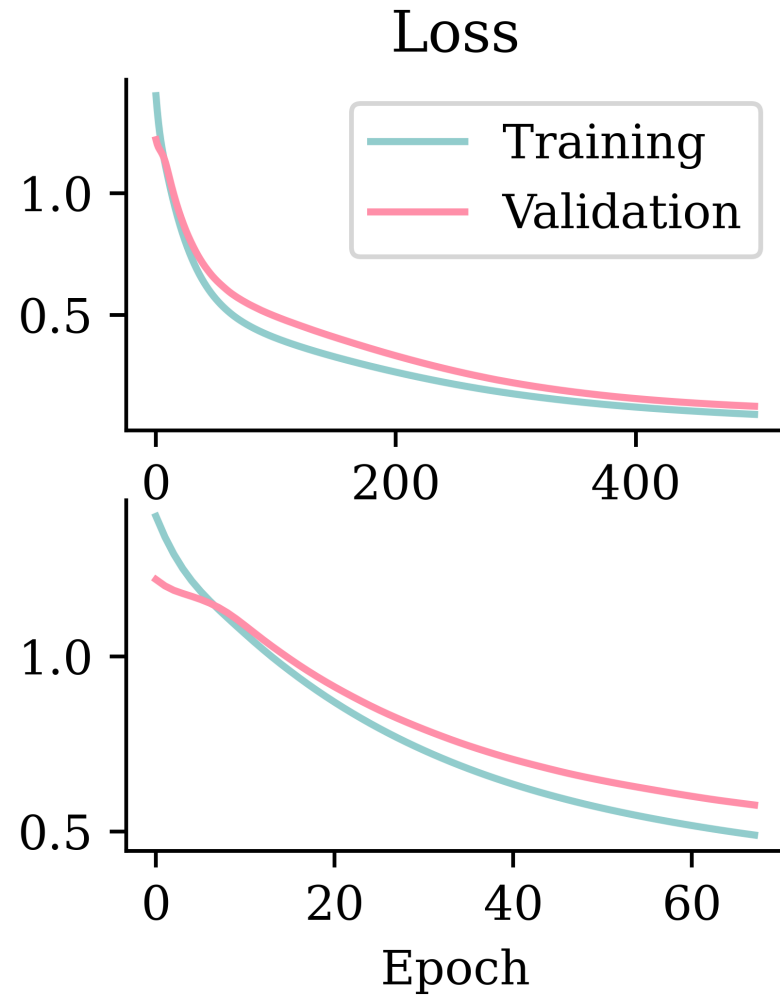
Evaluation on test set:

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

[0.9856260418891907, 0.5263158082962036]



Fitting metrics



What is the softmax activation?

It creates a “probability” vector: $\text{Softmax}(\mathbf{x}) = \frac{e_i^x}{\sum_j e_j^x}$.

In NumPy:

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

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

In Keras:

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

```
<tf.Tensor: shape=(1, 3), dtype=float32, numpy=array([[0.269, 0.001, 0.731]], dtype=float32)>
```



Prediction using classifiers

```
1 y_test[:4]
```

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

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

```
array([[0.1397096 , 0.5175301 , 0.34276026],
       [0.24611065, 0.44371164, 0.3101777 ],
       [0.26309973, 0.43174297, 0.3051573 ],
       [0.259089  , 0.44883674, 0.29207426]], dtype=float32)
```

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

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

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

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



Cross-entropy loss: ELI5

Neural Networks Part 6: Cross Entropy

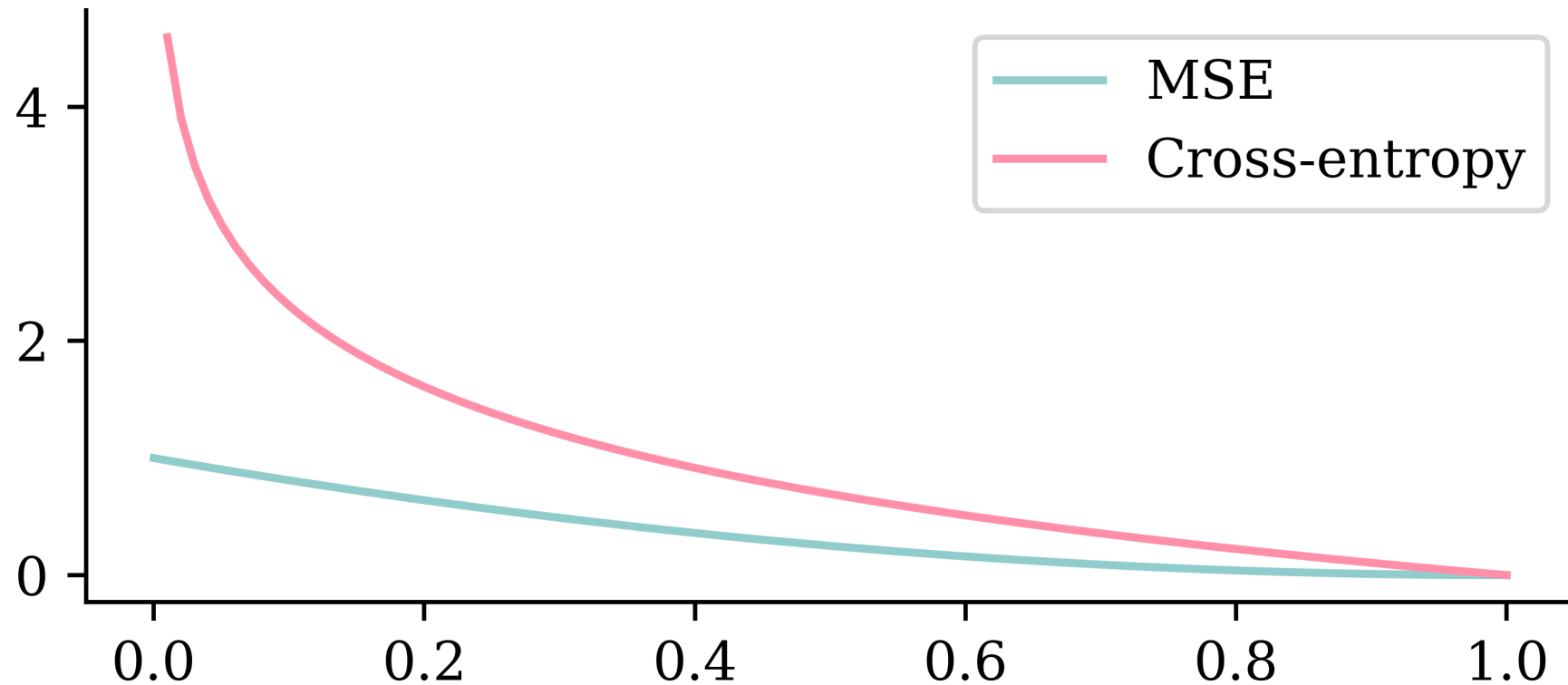


Neural Networks Part 7: Cross Entropy Derivativ...



Why use cross-entropy loss?

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



One-hot encoding

```
1 from sklearn.preprocessing import OneHotEncoder
2
3 enc = OneHotEncoder(sparse_output=False)
4
5 y_train_oh = enc.fit_transform(y_train)
6 y_test_oh = enc.transform(y_test)
```

```
1 y_train[:5]
```

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

```
1 y_train_oh[:5]
```

| | XO_0 | XO_1 | XO_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*):

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

Fit the model (*new target variables*):

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

Evaluate the model (*new target variables*):

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

[0.347093790769577, 0.9473684430122375]



Lecture Outline

- Classification
- **Stroke Prediction**



The data

Dataset source: [Kaggle Stroke Prediction Dataset](#).

```
1 data = pd.read_csv("stroke.csv")
2 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.

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

```
bmi      201
dtype: int64
```

```
1 features = data.drop(["id", "stroke"], axis=1)
2 target = data["stroke"]
3
4 X_main, X_test, y_main, y_test = train_test_split(
5     features, target, test_size=0.2, random_state=7)
6 X_train, X_val, y_train, y_val = train_test_split(
7     X_main, y_main, test_size=0.25, random_state=12)
8
9 X_train.shape, X_val.shape, X_test.shape
```

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



What values do we see in the data?

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

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

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

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

```
1 X_train["Residence_type"].value_counts()
```

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

```
1 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
```

```
1 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

```

1 from sklearn.pipeline import make_pipeline
2
3 cat_vars = ["gender", "ever_married", "Residence_type",
4             "work_type", "smoking_status"]
5
6 ct = make_column_transformer(
7     (OneHotEncoder(sparse_output=False, handle_unknown="ignore"), cat_vars),
8     ("passthrough", ["hypertension", "heart_disease"]),
9     remainder=make_pipeline(SimpleImputer(), StandardScaler()),
10    verbose_feature_names_out=False
11 )
12
13 X_train_ct = ct.fit_transform(X_train)
14 X_val_ct = ct.transform(X_val)
15 X_test_ct = ct.transform(X_test)
16
17 for name, X in zip(("train", "val", "test"), (X_train_ct, X_val_ct, X_test_ct)):
18     num_na = X.isna().sum().sum()
19     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

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

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

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

| gender | |
|--------|--------|
| 4970 | Male |
| 3116 | Other |
| 4140 | Male |
| 2505 | Female |

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

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

```
1 gender_cols = X_val_ct[["gender_Female"]
2 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

```

1 def create_model(seed=42):
2     random.seed(seed)
3     model = Sequential()
4     model.add(Input(X_train_ct.shape[1:]))
5     model.add(Dense(32, "leaky_relu"))
6     model.add(Dense(16, "leaky_relu"))
7     model.add(Dense(1, "sigmoid"))
8     return model

```

```

1 model = create_model()
2 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

```

1 model = create_model()
2
3 pr_auc = keras.metrics.AUC(curve="PR", name="pr_auc")
4 model.compile(optimizer="adam", loss="binary_crossentropy",
5               metrics=[pr_auc, "accuracy", "auc"])
6
7 es = EarlyStopping(patience=50, restore_best_weights=True,
8                   monitor="val_pr_auc", verbose=1)
9 model.fit(X_train_ct, y_train, callbacks=[es], epochs=1_000, verbose=0,
10         validation_data=(X_val_ct, y_val));

```

Epoch 65: early stopping

Restoring model weights from the end of the best epoch: 15.

```

1 model.evaluate(X_val_ct, y_val, verbose

```

```

[0.14444081485271454,
 0.13122102618217468,
 0.9589040875434875,
 0.8215014934539795]

```



Overweight the minority class

```

1 model = create_model()
2
3 pr_auc = keras.metrics.AUC(curve="PR", name="pr_auc")
4 model.compile(optimizer="adam", loss="binary_crossentropy",
5               metrics=[pr_auc, "accuracy", "auc"])
6
7 es = EarlyStopping(patience=50, restore_best_weights=True,
8                   monitor="val_pr_auc", verbose=1)
9 model.fit(X_train_ct, y_train.to_numpy(), callbacks=[es], epochs=1_000, verbose=0,
10         validation_data=(X_val_ct, y_val), class_weight={0: 1, 1: 10});

```

Epoch 74: early stopping

Restoring model weights from the end of the best epoch: 24.

```

1 model.evaluate(X_val_ct, y_val, verbose

```

```

[0.3345569670200348,
 0.13615098595619202,
 0.8062622547149658,
 0.8122206330299377]

```

```

1 model.evaluate(X_test_ct, y_test, verbo

```

```

[0.3590189516544342,
 0.1449822038412094,
 0.8023483157157898,
 0.7915638089179993]

```

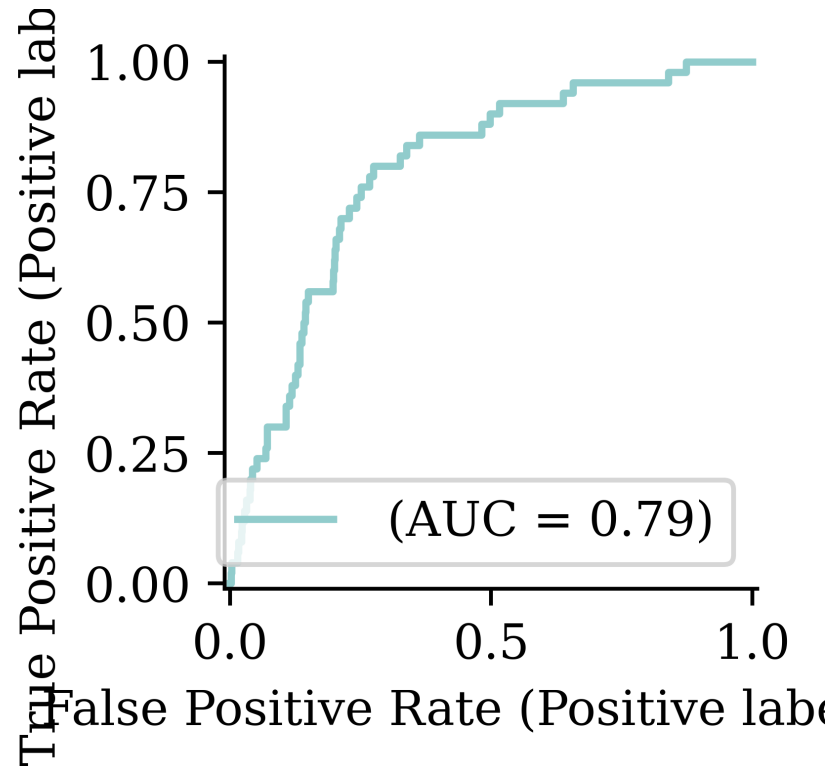


Classification Metrics

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

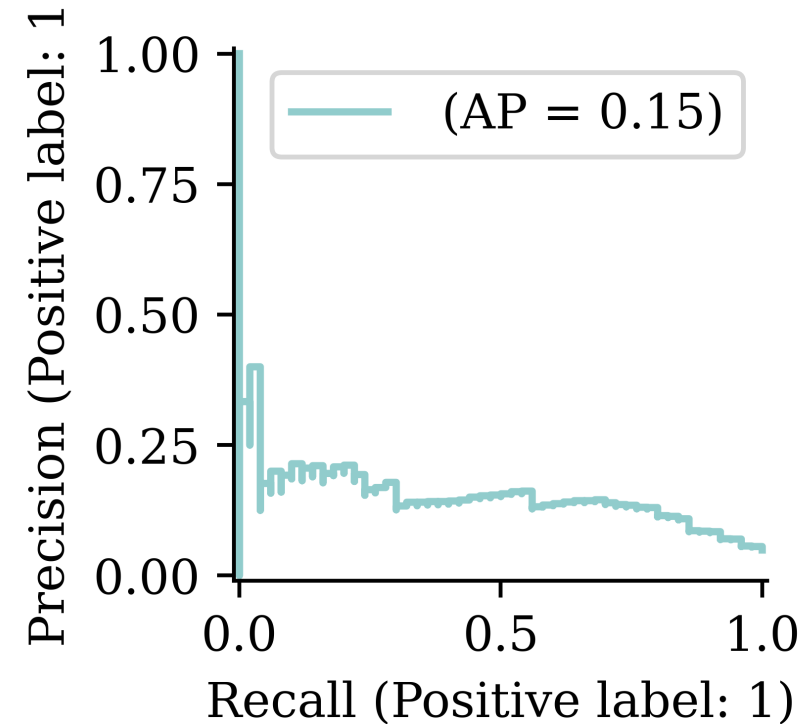
```
1 RocCurveDisplay.from_predictions(y_test, y_pred, name="");
```

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



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

```
array([[792, 180],
       [ 22,  28]])
```



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

```
array([[662, 310],
       [ 10,  40]])
```



Package Versions

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

Python implementation: CPython

Python version : 3.11.9

IPython version : 8.24.0

keras : 3.3.3

matplotlib: 3.8.4

numpy : 1.26.4

pandas : 2.2.2

seaborn : 0.13.2

scipy : 1.11.0

torch : 2.0.1

tensorflow: 2.16.1

tf_keras : 2.16.0



Glossary

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

