



DEEP LEARNING IN PYTHON

Creating a keras model



Model building steps

- Specify Architecture
- Compile
- Fit
- Predict



Model specification

```
In [1]: import numpy as np

In [2]: from keras.layers import Dense

In [3]: from keras.models import Sequential

In [4]: predictors = np.loadtxt('predictors_data.csv', delimiter=',')

In [5]: n_cols = predictors.shape[1]

In [6]: model = Sequential()

In [7]: model.add(Dense(100, activation='relu', input_shape = (n_cols,)))

In [8]: model.add(Dense(100, activation='relu'))

In [9]: model.add(Dense(1))
```



DEEP LEARNING IN PYTHON

Let's practice!



DEEP LEARNING IN PYTHON

Compiling and fitting a model

Why you need to compile your model

- Specify the optimizer
 - Many options and mathematically complex
 - “Adam” is usually a good choice
- Loss function
 - “mean_squared_error” common for regression



Compiling a model

```
In [1]: n_cols = predictors.shape[1]

In [2]: model = Sequential()

In [3]: model.add(Dense(100, activation='relu', input_shape=(n_cols,)))

In [4]: model.add(Dense(100, activation='relu'))

In [5]: model.add(Dense(1))

In [6]: model.compile(optimizer='adam', loss='mean_squared_error')
```



What is fitting a model

- Applying backpropagation and gradient descent with your data to update the weights
- Scaling data before fitting can ease optimization

Fitting a model

```
In [1]: n_cols = predictors.shape[1]

In [2]: model = Sequential()

In [3]: model.add(Dense(100, activation='relu', input_shape=(n_cols,)))

In [4]: model.add(Dense(100, activation='relu'))

In [5]: model.add(Dense(1))

In [6]: model.compile(optimizer='adam', loss='mean_squared_error')

In [7]: model.fit(predictors, target)
```



DEEP LEARNING IN PYTHON

Let's practice!



DEEP LEARNING IN PYTHON

Classification models



Classification

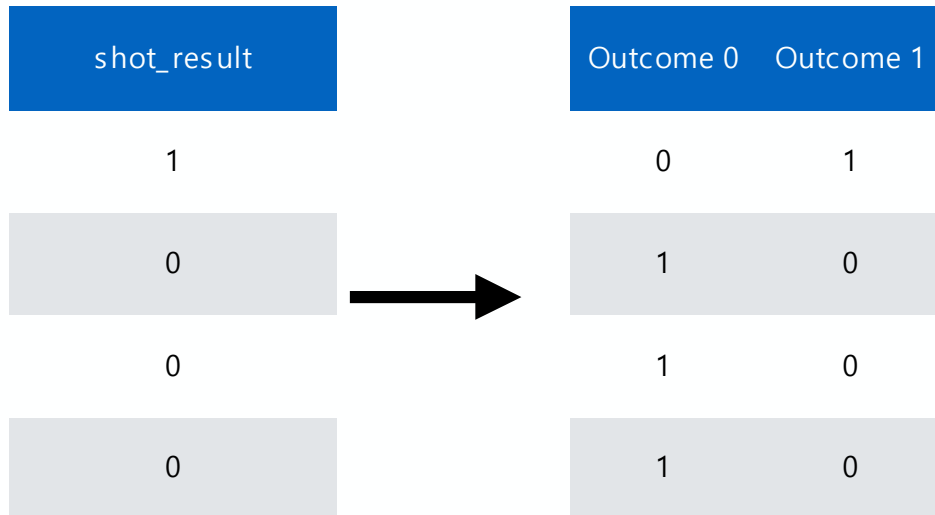
- 'categorical_crossentropy' loss function
- Similar to log loss: Lower is better
- Add metrics = ['accuracy'] to compile step for easy-to-understand diagnostics
- Output layer has separate node for each possible outcome, and uses 'softmax' activation



Quick look at the data

| shot_clock | dribbles | touch_time | shot_dis | close_def_dis | shot_result |
|------------|----------|------------|----------|---------------|-------------|
| 10.8 | 2 | 1.9 | 7.7 | 1.3 | 1 |
| 3.4 | 0 | 0.8 | 28.2 | 6.1 | 0 |
| 0 | 3 | 2.7 | 10.1 | 0.9 | 0 |
| 10.3 | 2 | 1.9 | 17.2 | 3.4 | 0 |

Transforming to categorical





Classification

```
In[1]: from keras.utils import to_categorical

In[2]: data = pd.read_csv('basketball_shot_log.csv')

In[3]: predictors = data.drop(['shot_result'], axis=1).as_matrix()

In[4]: target = to_categorical(data.shot_result)

In[5]: model = Sequential()

In[6]: model.add(Dense(100, activation='relu', input_shape = (n_cols,)))

In[7]: model.add(Dense(100, activation='relu'))

In[8]: model.add(Dense(100, activation='relu'))

In[9]: model.add(Dense(2, activation='softmax'))

In[10]: model.compile(optimizer='adam', loss='categorical_crossentropy',
...:                  metrics=['accuracy'])

In[11]: model.fit(predictors, target)
```



Classification

```
Out[11]:
Epoch 1/10
128069/128069 [=====] - 4s - loss: 0.7706 - acc: 0.5759
Epoch 2/10
128069/128069 [=====] - 5s - loss: 0.6656 - acc: 0.6003
Epoch 3/10
128069/128069 [=====] - 6s - loss: 0.6611 - acc: 0.6094
Epoch 4/10
128069/128069 [=====] - 7s - loss: 0.6584 - acc: 0.6106
Epoch 5/10
128069/128069 [=====] - 7s - loss: 0.6561 - acc: 0.6150
Epoch 6/10
128069/128069 [=====] - 9s - loss: 0.6553 - acc: 0.6158
Epoch 7/10
128069/128069 [=====] - 9s - loss: 0.6543 - acc: 0.6162
Epoch 8/10
128069/128069 [=====] - 9s - loss: 0.6538 - acc: 0.6158
Epoch 9/10
128069/128069 [=====] - 10s - loss: 0.6535 - acc: 0.6157
Epoch 10/10
128069/128069 [=====] - 10s - loss: 0.6531 - acc: 0.6166
```




DEEP LEARNING IN PYTHON

Let's practice!



DEEP LEARNING IN PYTHON

Using models



Using models

- Save
- Reload
- Make predictions



Saving, reloading and using your Model

```
In [1]: from keras.models import load_model  
  
In [2]: model.save('model_file.h5')  
  
In [3]: my_model = load_model('my_model.h5')  
  
In [4]: predictions = my_model.predict(data_to_predict_with)  
  
In [5]: probability_true = predictions[:,1]
```



Verifying model structure

```
In [6]: my_model.summary()  
Out[6]:
```

| Layer (type) | Output Shape | Param # | Connected to |
|-----------------|--------------|---------|---------------------|
| dense_1 (Dense) | (None, 100) | 1100 | dense_input_1[0][0] |
| dense_2 (Dense) | (None, 100) | 10100 | dense_1[0][0] |
| dense_3 (Dense) | (None, 100) | 10100 | dense_2[0][0] |
| dense_4 (Dense) | (None, 2) | 202 | dense_3[0][0] |

```
Total params: 21,502  
Trainable params: 21,502  
Non-trainable params: 0
```



DEEP LEARNING IN PYTHON

Let's practice!