Deepika Singh

Classification with Keras

Deepika Singh

Apr 10, 2019 • 16 Min read • 54,020 Views

Apr 10, 2019 • 16 Min read • 54,020 Views

Data Keras

Introduction

125

- Introduction
- Problem Statement
- Steps
- Evaluation of the Model Performance
- Conclusion
- Top ^

Introduction

Deep Learning is one of the hottest topics in data science and artificial intelligence today. It is a subfield of machine learning, comprising of a set of algorithms that are based on learning representations of data. Deep Learning has been applied in some of the most exciting technological innovations today like robotics, autonomous vehicles, computer vision, natural language processing, image recognition, and many more.

There are many deep learning libraries out there, but the most popular ones are TensorFlow, Keras, and PyTorch. We will be focussing on Keras in this guide.

Keras is a high-level neural networks API, written in Python, and can run on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. The advantages of using Keras emanates from the fact that it focuses on being user-friendly, modular, and extensible.

In this guide, we will focus on how to use the Keras library to build classification models. If you are looking for a guide on how to carry out Regression with Keras, please refer to my previous guide (/guides/regression-keras/)

Classification with Keras

Classification is a type of supervised machine learning algorithm used to predict a categorical label. A few useful examples of classification include predicting whether a customer will churn or not, classifying emails into spam or not, or whether a bank loan will default or not.

The basic architecture of the deep learning neural network, which we will be following, consists of three main components.

1. Input Layer: This is where the training observations are fed. The number of predictor variables is also specified here through the neurons.

2. Hidden Layers: These are the intermediate layers between the input and output layers. The deep neural network learns about the relationships involved in data in this component.
3. Output Layer: This is the layer where the final output is extracted from what's happening in the previous two layers. In case of regression problems, the output layer will have one neuron.

Problem Statement

Diabetes is a serious health issue which causes an increase in blood sugar. Many complications occur if diabetes remains untreated and unidentified.

The aim of this guide is to build a classification model to detect diabetes. We will be using the diabetes dataset which contains 768 observations and 9 variables, as described below:

- pregnancies - Number of times pregnant
- glucose - Plasma glucose concentration
- diastolic - diastolic blood pressure (mm Hg)
- triceps - Skinfold thickness (mm)
- insulin - Hour serum insulin (mu U/ml)
- bmi – Basal metabolic rate (weight in kg/height in m)
- dpf - Diabetes pedigree function
- age - Age in years
- diabetes - 1 represents the presence of diabetes while 0 represents the absence of it. This is the target variable.

Also, the classification algorithm selected is the Logistic Regression Model, which is one of the oldest and most widely used algorithms.

Evaluation Metric

We will evaluate the performance of the model using accuracy, which represents the percentage of cases correctly classified.

Mathematically, for a binary classifier, it's represented as accuracy = (TP+TN)/(TP+TN+FP+FN), where

- True Positive, or TP, are cases with positive labels which have been correctly classified as positive.
- True Negative, or TN, are cases with negative labels which have been correctly classified as negative.
- False Positive, or FP, are cases with negative labels which have been incorrectly classified as positive.
- False Negative, or FN, are cases with positive labels which have been incorrectly classified as negative.

Steps

Following are the steps which are commonly followed while implementing Regression Models with Keras.

Step 1 - Loading the required libraries and modules

Step 2 - Loading the data and performing basic data checks

Step 3 - Creating arrays for the features and the response variable

Step 4 - Creating the Training and Test datasets

Step 5 - Define, compile, and fit the Keras classification model

Step 6 - Predict on the test data and compute evaluation metrics

The following sections will cover these steps.

Step 1 - Loading the Required Libraries and Modules

```python
# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn

# Import necessary modules
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from math import sqrt

# Keras specific
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
```

Step 2 - Reading the Data and Performing Basic Data Checks

The ***first line*** of code reads in the data as pandas dataframe, while the ***second line*** of code prints the shape - 768 observations of 9 variables. The ***third line*** gives summary statistics of the numerical variables. There are no missing values in the data, as all the variables have 768 as 'count' which is equal to the number of records in the dataset.

```python
df = pd.read_csv('diabetes.csv')
print(df.shape)
df.describe()
```

```
(768, 9)
```

	pregnancies	glucose	diastolic	triceps	insulin	b
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.0
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.99
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884

	pregnancies	glucose	diastolic	triceps	insulin	b
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.3C
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.0C
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.6C
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.1C

Step 3 - Creating Arrays for the Features and the Response Variable.

The *first line* of code creates an object of the target variable, while the *second line* of code gives the list of all the features after excluding the target variable, 'diabetes'.

The *third line* does normalization of the predictors via scaling between 0 and 1. This is needed to eliminate the influence of the predictor's units and magnitude on the modelling process.

The *fourth line* displays the summary of the normalized data. The target variable remains unchanged.

```python
target_column = ['diabetes']
predictors = list(set(list(df.columns))-set(target_column))
df[predictors] = df[predictors]/df[predictors].max()
df.describe()
```

pregnancies	glucose	diastolic	triceps	insulin	bmi
count	768.000000	768.000000	768.000000	768.000000	768.000000
mean	0.226180	0.607510	0.566438	0.207439	0.094326
std	0.198210	0.160666	0.158654	0.161134	0.136222
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.058824	0.497487	0.508197	0.000000	0.000000
50%	0.176471	0.587940	0.590164	0.232323	0.036052
75%	0.352941	0.704774	0.655738	0.323232	0.150414
max	1.000000	1.000000	1.000000	1.000000	1.000000

Step 4 - Creating the Training and Test Datasets

The *first couple of lines* creates arrays of independent (X) and dependent (y) variables, respectively. The *third line* splits the data into training and test datasets, with 30% of the observations in the test set. The *fourth line* of code prints the shape of the training set (537 observations of 8 variables) and test set (231 observations of 8 variables).

```python
X = df[predictors].values
y = df[target_column].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, r
print(X_train.shape); print(X_test.shape)
```

```
(537, 8)
(231, 8)
```

Since our target variable represents a binary category which has been coded as numbers 0 and 1, we will have to encode it. We can easily achieve that using the "to_categorical" function from the Keras utilities package. The ***two lines*** of code below accomplishes that in both training and test datasets.

```python
# one hot encode outputs
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

count_classes = y_test.shape[1]
print(count_classes)
```

```
2
```

Step 5 - Define, Compile, and Fit the Keras Classification Model

We will start by setting up the model. The ***first line*** of code calls for the Sequential constructor. We are using the Sequential model because our network consists of a linear stack of layers.

The ***second line*** of code represents the input layer which specifies the activation function and the number of input dimensions, which in our case is 8 predictors. Then we repeat the same process in the ***third and fourth line*** of codes for the two hidden layers, but this time without the input_dim parameter. The activation function used is a rectified linear unit, or ReLU. ReLU is the most widely used activation function because it is nonlinear, and has the ability to not activate all the neurons at the same time.

The ***fifth line*** of code creates the output layer with two nodes because there are two output classes, 0 and 1. We use 'softmax' as the activation function for the output layer, so that the sum of the predicted values from all the neurons in the output layer adds up to one.

In the above lines of codes, we have defined our deep learning model architecture. But before we can start training the model, we will configure the learning process. This is done in the last line of code using the model.compile() function.

In defining our compiler, we will use 'categorical cross-entropy' as our loss measure, 'adam' as the optimizer algorithm, and 'accuracy' as the evaluation metric.

The main advantage of the "adam" optimizer is that we don't need to specify the learning rate, as is the case with gradient descent. Using “adam” will, thereby, save us the task of optimizing the learning rate for our model.

```python
model = Sequential()
model.add(Dense(500, activation='relu', input_dim=8))
model.add(Dense(100, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(2, activation='softmax'))

# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Now we are ready to build the model which is done in the code below. We also provide the argument, epochs, which represents the number of training iterations. We have taken 20 epochs.

```python
# build the model
model.fit(X_train, y_train, epochs=20)
```

Epoch 1/20 537/537 ============================== - 0s 743us/step - loss: 0.6540 - acc: 0.6667

Epoch 2/20 537/537 ============================== - 0s 127us/step - loss: 0.6199 - acc: 0.6704

Epoch 3/20 537/537 ============================== - 0s 118us/step - loss: 0.5860 - acc: 0.7058

Epoch 4/20 537/537 ============================== - 0s 116us/step - loss: 0.5679 - acc: 0.7244

Epoch 5/20 537/537 ============================== - 0s 123us/step - loss: 0.5525 - acc: 0.7430

Epoch 6/20 537/537 ============================== - 0s 127us/step - loss: 0.5163 - acc: 0.7505

Epoch 7/20 537/537 ============================== - 0s 127us/step - loss: 0.5130 - acc: 0.7616

Epoch 8/20 537/537 ============================== - 0s 115us/step - loss: 0.5306 - acc: 0.7449

Epoch 9/20 537/537 ============================== - 0s 119us/step - loss: 0.4964 - acc: 0.7691

Epoch 10/20 537/537 ============================== - 0s 110us/step - loss: 0.4985 - acc: 0.7691

Epoch 11/20 537/537 ============================== - 0s 145us/step - loss: 0.4838 - acc: 0.7784

```
Epoch 12/20 537/537 ============================== - 0s 111us/step - loss:
0.4855 - acc: 0.7579

Epoch 13/20 537/537 ============================== - 0s 126us/step - loss:
0.4546 - acc: 0.7914

Epoch 14/20 537/537 ============================== - 0s 124us/step - loss:
0.4586 - acc: 0.7784

Epoch 15/20 537/537 ============================== - 0s 129us/step - loss:
0.4466 - acc: 0.8026

Epoch 16/20 537/537 ============================== - 0s 114us/step - loss:
0.4397 - acc: 0.7970

Epoch 17/20 537/537 ============================== - 0s 122us/step - loss:
0.4386 - acc: 0.8026

Epoch 18/20 537/537 ============================== - 0s 133us/step - loss:
0.4549 - acc: 0.7858

Epoch 19/20 537/537 ============================== - 0s 141us/step - loss:
0.4705 - acc: 0.7765

Epoch 20/20 537/537 ============================== - 0s 124us/step - loss:
0.4694 - acc: 0.7821
```

```
<keras.callbacks.History at 0x7f119cc681d0>
```

Step 6 - Predict on the Test Data and Compute Evaluation Metrics;

The *first line* of code predicts on the train data, while the *second line* evaluates the model, and the *third line* prints the accuracy and error on the training data.

The same is repeated in the *fourth, fifth and sixth lines* of code which is performed on the test data.

python

```
pred_train= model.predict(X_train)
scores = model.evaluate(X_train, y_train, verbose=0)
print('Accuracy on training data: {}% \n Error on training data: {}'.format

pred_test= model.predict(X_test)
scores2 = model.evaluate(X_test, y_test, verbose=0)
print('Accuracy on test data: {}% \n Error on test data: {}'.format(scores2
```

```
Accuracy on training data: 0.8081936690838422%
 Error on training data: 0.19180633091615784
Accuracy on test data: 0.7575757580918151%
 Error on test data: 0.2424242419081849
```

Evaluation of the Model Performance

The output above shows the performance of the model on both training and test data. The accuracy was around 81% on the training data and 76% on the test data. Ideally, the higher the accuracy value, the better the model performance.

Conclusion

In this guide, we have built Classification models using the deep learning framework, Keras. The guide used the diabetes dataset and built a classifier algorithm to predict detection of diabetes.

Our model is achieving a decent accuracy of 81% and 76% on training and test data, respectively. We see that the accuracy decreases for the test data set, but that is often the case while working with hold out validation approach.

The model can be further improved by doing cross-validation, feature engineering, trying out more advanced machine learning algorithms, or changing the arguments in the deep learning network we built above. However, that is not in the scope of this guide which is aimed at enabling individuals to solve classification problems using deep learning library Keras.

cookies, click here (https://www.pluralsight.com/privacy.html).

ALLOW DECLINE

RESOURCES

Download Pluralsight (/product/downloads)

SUPPORT

Contact (/contact)