Table of Contents

[Loss Function: Mean\_Squared\_Error 1](#_Toc87961053)

[Loss Function: binary\_crossentropy 2](#_Toc87961054)

[Loss Function: categorical\_crossentropy 5](#_Toc87961055)

## Loss Function: Mean\_Squared\_Error

We use a loss function such as mean square error for continuous regression problems. The predictions look like this:

[[20.716156 ]

[24.089813 ]

[28.559914 ]

[13.621188 ]

[19.914501 ]

|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn import metrics  from sklearn.model\_selection import train\_test\_split  from keras.models import Sequential  from keras.layers import Dense  from keras.wrappers.scikit\_learn import KerasRegressor  from sklearn.model\_selection import cross\_val\_score  from sklearn.model\_selection import KFold  # Read the data.  PATH = "/Users/pm/Desktop/DayDocs/data/"  CSV\_DATA = "housing.data"  df = pd.read\_csv(PATH + CSV\_DATA, header=None)  # Show all columns.  pd.set\_option('display.max\_columns', None)  pd.set\_option('display.width', 1000)  print(df.head())  print(df.tail())  print(df.describe())  # Convert DataFrame columns to vertical columns so they can be used by the NN.  dataset = df.values  X = dataset[:, 0:13] # Columns 0 to 12  y = dataset[:, 13] # Columns 13  ROW\_DIM = 0  COL\_DIM = 1  x\_arrayReshaped = X.reshape(X.shape[ROW\_DIM], X.shape[COL\_DIM])  y\_arrayReshaped = y.reshape(y.shape[ROW\_DIM],1)  # Split the data.  X\_train, X\_test, y\_train, y\_test = train\_test\_split(x\_arrayReshaped,  y\_arrayReshaped, test\_size=0.2, random\_state=0)  # Define the model.  def create\_model():  model = Sequential()  model.add(Dense(13, input\_dim=13, kernel\_initializer='normal',  activation='relu'))  model.add(Dense(1, kernel\_initializer='normal'))  model.compile(loss='mean\_squared\_error', optimizer='adam')  return model  # Since this is a linear regression use KerasRegressor.  estimator = KerasRegressor(build\_fn=create\_model, epochs=100,  batch\_size=5, verbose=1)  # Use kfold analysis for a more reliable estimate.  kfold = KFold(n\_splits=10)  results = cross\_val\_score(estimator, X\_train, y\_train, cv=kfold)  print("Baseline Mean (%.2f) MSE (%.2f) " % (results.mean(), results.std()))  print("Baseline RMSE: " + str(np.sqrt(results.std())))  # Build the model.  model = create\_model()  history = model.fit(X\_train, y\_train, epochs=100,  batch\_size=5, verbose=1,  validation\_data=(X\_test, y\_test))  # Evaluate the model.  predictions = model.predict(X\_test)  mse = metrics.mean\_squared\_error(y\_test, predictions)  print("Neural network MSE: " + str(mse))  print("Neural network RMSE: " + str(np.sqrt(mse)))  print(predictions) |

## Loss Function: binary\_crossentropy

Binary\_crossentropy outputs data to a single column. It is for binary categorical predictions.

The output looks like:

|  |
| --- |
| [[0.65173334]  [0.24781752]  [0.22344482]  [0.44778693]  [0.4039366 ]  [0.42592156] |

Here is an example of a binary classification which uses binary\_crossentropy:

|  |
| --- |
| import pandas as pd  from sklearn.model\_selection import train\_test\_split  from sklearn.linear\_model import LogisticRegression  from sklearn import metrics  PATH = "/Users/pm/Desktop/DayDocs/data/"  FILE = "heart\_disease.csv"  data = pd.read\_csv(PATH + FILE)  x\_data = data.drop("target", axis=1)  y\_values = data["target"]  # Show all columns.  pd.set\_option('display.max\_columns', None)  pd.set\_option('display.width', 1000)  print(data.head())  X\_train, X\_test, y\_train, y\_test = train\_test\_split(  x\_data, y\_values, test\_size=0.3, random\_state=42  )  # Stochastic gradient descent models are sensitive to differences  from sklearn.preprocessing import StandardScaler  scaler = StandardScaler()  scaler.fit(X\_train)  X\_trainScaled = scaler.transform(X\_train)  X\_testScaled = scaler.transform(X\_test)  clf = LogisticRegression(max\_iter=1000)  clf.fit(X\_trainScaled, y\_train)  lr\_pred = clf.predict(X\_testScaled)  print("Accuracy:{} ".format(clf.score(X\_testScaled, y\_test) \* 100))  print("Error Rate:{} ".format((1 - clf.score(X\_testScaled, y\_test)) \* 100))  # Show confusion matrix and accuracy scores.  confusion\_matrix = pd.crosstab(y\_test, lr\_pred,  rownames=['Actual'],  colnames=['Predicted'])  print('\nAccuracy: ',metrics.accuracy\_score(y\_test, lr\_pred))  print("\nConfusion Matrix")  print(confusion\_matrix)  COLUMN\_DIMENSION = 1  #######################################################################  # Part 2  from keras.models import Sequential  from keras.layers import Dense  from keras.wrappers.scikit\_learn import KerasClassifier  from sklearn.model\_selection import GridSearchCV  # shape() obtains rows (dim=0) and columns (dim=1)  n\_features = X\_trainScaled.shape[COLUMN\_DIMENSION]  #######################################################################  # Model tuning section.  def create\_model():  model = Sequential()  model.add(Dense(12, input\_dim=n\_features, activation='relu'))  model.add(Dense(1, activation='sigmoid'))  model.compile(loss='binary\_crossentropy', optimizer='adam',  metrics=['accuracy'])  print("Predictions: ")  predictions = model.predict(X\_testScaled)  print(predictions)  return model  model = KerasClassifier(build\_fn=create\_model, verbose=1)  batch\_size = [200]  epochs = [30]  param\_grid = dict(batch\_size=batch\_size, epochs=epochs)  grid = GridSearchCV(estimator=model, param\_grid=param\_grid, n\_jobs=-1, cv=3)  #######################################################################  grid\_result = grid.fit(X\_trainScaled, y\_train)  # summarize results  print("Best: %f using %s" % (grid\_result.best\_score\_, grid\_result.best\_params\_))  means = grid\_result.cv\_results\_['mean\_test\_score']  stds = grid\_result.cv\_results\_['std\_test\_score']  params = grid\_result.cv\_results\_['params']  for mean, stdev, param in zip(means, stds, params):  print("%f (%f) with: %r" % (mean, stdev, param)) |

## Loss Function: categorical\_crossentropy

Target categorical data that is one-hot encoded uses ‘categorical\_crossentropy’ for loss. Here we are using one-hot encoding so we must use categorical\_crossentropy. One-hot encoding is a fancy way to say multi-column binary encoding. Here is our sample target variable:

# Y\_train  
# [[0 1]  
# [1 0]  
# [0 1]

The output looks like:

\*\*\* Predictions

[[0.45627576 0.54372424]

[0.45627576 0.54372424]

[0.45627576 0.54372424]

[0.45627576 0.54372424]

Here is a code sample which uses categorical\_crossentropy

|  |
| --- |
| import pandas as pd  import re  from keras\_preprocessing.sequence import pad\_sequences  from sklearn.model\_selection import train\_test\_split  from tensorflow.python.keras import Sequential  from tensorflow.python.keras.layers import Embedding, LSTM, Dense  PATH = "/Users/pm/Desktop/DayDocs/data/"  FILE = "yelp\_mini.csv"  data = pd.read\_csv(PATH + FILE)  # Show all columns.  pd.set\_option('display.max\_columns', None)  pd.set\_option('display.width', 1000)  # Create a sentiment column.  # Ratings above 3 are positive, otherwise they are negative.  data['sentiment'] = ['pos' if (x>3) else 'neg' for x in data['stars']]  data['text'] = data['text'].apply((lambda x: re.sub('[^a-zA-z0-9\s]','',x)))  from keras.preprocessing.text import Tokenizer  VOCABULARY\_SIZE = 2500  tokenizer = Tokenizer(num\_words=VOCABULARY\_SIZE, lower=True,split=' ')  tokenizer.fit\_on\_texts(data['text'].values)  X = tokenizer.texts\_to\_sequences(data['text'].values)  X = pad\_sequences(X)  WORDS\_PER\_SENTENCE = X.shape[0]  NUM\_REVIEWS = X.shape[1]  import numpy as np  VOCABULARY\_SIZE = np.amax(X) + 1  word\_vector\_sz = 128 # Size of output vector for each word.  sequence\_sz = 200 # Vector size for info about entire sequence. Can be changed.  batch\_size = 32  model = Sequential()  model.add(Embedding(VOCABULARY\_SIZE, word\_vector\_sz))  model.add(LSTM(sequence\_sz, dropout=0.2))  model.add(Dense(2, activation='softmax'))  # Target data is one-hot encoded so we must use ‘categorical\_crossentropy’ for loss.  # Here we are using one-hot encoding so we must use categorical\_crossentropy.  # One-hot encoding is a fancy way to say multi-column binary encoding.  # Y\_train  # [[0 1]  # [1 0]  # [0 1]  model.compile(loss = 'categorical\_crossentropy', optimizer='adam',metrics = ['accuracy'])  print(model.summary())  Y = pd.get\_dummies(data['sentiment']).values  X\_train, X\_test, Y\_train, y\_test = train\_test\_split(X,Y, test\_size = 0.20)  history = model.fit(X\_train, Y\_train, batch\_size =batch\_size, epochs =4,  verbose = 1, validation\_data=(X\_test, y\_test))  score, acc = model.evaluate(X\_test, y\_test, verbose=2, batch\_size=batch\_size)  print("Score: %.2f" % (score))  print("Validation Accuracy: %.2f" % (acc))  print("\n\*\*\* Predictions")  predictions = model.predict(y\_test)  print(predictions)  import matplotlib.pyplot as plt  def showLoss(history):  # Get training and test loss histories  training\_loss = history.history['loss']  validation\_loss = history.history['val\_loss']  # Create count of the number of epochs  epoch\_count = range(1, len(training\_loss) + 1)  plt.subplot(1, 2, 1)  # Visualize loss history for training data.  plt.plot(epoch\_count, training\_loss, label='Train Loss', color='red')  # View loss on unseen data.  plt.plot(epoch\_count, validation\_loss, 'r--', label='Validation Loss',  color='black')  plt.xlabel('Epoch')  plt.legend(loc="best")  plt.title("Loss")  def showAccuracy(history):  # Get training and test loss histories  training\_loss = history.history['accuracy']  validation\_loss = history.history['val\_accuracy']  # Create count of the number of epochs  epoch\_count = range(1, len(training\_loss) + 1)  plt.subplot(1, 2, 2)  # Visualize loss history for training data.  plt.plot(epoch\_count, training\_loss, label='Train Accuracy', color='red')  # View loss on unseen data.  plt.plot(epoch\_count, validation\_loss, 'r--',  label='Validation Accuracy', color='black')  plt.xlabel('Epoch')  plt.legend(loc="best")  plt.title('Accuracy')  plt.subplots(nrows=1, ncols=2, figsize=(14,7))  showLoss(history)  showAccuracy(history)  plt.show() |