三和テッキ　学習用プログラム　説明資料

ここでは例としてc50.ipynbについて説明します。

ほかのプログラムにつきましても引数が異なるだけで関数が同じものについては省略します。動作の結果に関しては実際にこのプログラムを動かしながら確認していただくと理解しやすいと思います。

c50.ipynb

**import** **pandas** **as** **pd**

**import** **glob**

**import** **codecs**

**from** **pandas** **import** Series, DataFrame

**from** **keras.models** **import** Sequential

**from** **keras.layers** **import** Dense, Activation,Dropout

**from** **keras.optimizers** **import** Adam,RMSprop,SGD

**from** **keras.utils** **import** np\_utils

**from** **keras** **import** optimizers,regularizers

**from** **sklearn** **import** datasets

**from** **sklearn.model\_selection** **import** train\_test\_split

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

**from** **google.colab** **import** drive

drive.mount('/content/drive')

今回のプログラムを動作させるために必要なパッケージなどをインポートしています。

最後のgoogle.colabの部分はデータをどこから持ってくるかによります。今回はgoogledriveを使用したためこのようになっています。

*#圧力\_温度\_配管ID.csvの読み込み*

data1 = pd.read\_csv("drive/My Drive/mdb/test/C50/圧力\_温度\_配管ID.csv", encoding='shift\_jis')

data1

圧力＿温度＿配管ID.csvを読み込んでいます。Googledriveを使用しない場合はdrive/My Drive/mdb/test/C50/圧力\_温度\_配管ID.csvの部分が変わります。

今後も同じように出てきますが単にdata1などと変数名が書いてあるのはその時の値を確認しているだけなので省略可能です。

*#配管特性.csvの読み込み*

data2 = pd.read\_csv('drive/My Drive/mdb/test/C50/配管特性.csv',encoding='shift\_jis')

data2

配管特性.csvを読み込んでいます。data1と同様です。

*#圧力\_温度\_配管ID.csvのデータの配管IDに合わせて配管特性.csvのデータを結合*

data1\_frame = DataFrame(data1)

data2\_frame = DataFrame(data2)

data12 = pd.merge(data1\_frame,data2\_frame,how='left',left\_on='配管 ID',right\_on = '配管ID')

data12

data1とdata2を配管IDの列で結合しています。今後データを増やす場合はこれと同様な操作で結合することが可能です。その場合結合するときには配管IDの列以外でも結合することは可能です。

*#'To'データの削除*

data12 = data12.drop("To",axis=1)

**for** col\_name **in** data12.columns:

print(col\_name,data12[col\_name].dtype)

*#結合後のデータ種類*

data12

今回はfromとtoのデータでfromのデータのみを使用したため結合後のdata12からtoデータを削除しています。

*#'配管ID'などの文字データのダミー化*

data12\_dummy = pd.get\_dummies(data12[data12.columns[data12.columns != 'From']])

data12\_dummy['From'] = data12['From']

print(data12\_dummy['配管 ID\_P250S40'])

data12\_dummy

配管IDの列のように文字データになっている値を0,1の数値に変換しています。

使用するデータの内容によってはダミー化するのか、削除するのか変更可能です。

*#Support.csvの読み込み*

data\_out = pd.read\_csv("drive/My Drive/mdb/test/C50/Support.csv",encoding='shift\_jis')

data\_out

Support.csvを読み込んでいます。

*#使用したい "Point","Type"データのみにする*

**for** col\_name **in** data\_out.columns:

**if** col\_name != "Point" **and** col\_name != "Type" :

data\_out = data\_out.drop(col\_name,axis=1)

data\_out

今回使用したのはSupport.csvの中でもPointとTypeのデータのみだったのでそれ以外のデータを削除しています。これに関してもアウトプット、インプットそれぞれで使用するデータを増やす場合は削除せずに学習に使用することになります。

*#重複しているデータの削除*

data\_out = data\_out.drop\_duplicates()

data\_out

今回はPointごとにサポートがついているのかどうかのみを使用したかったので重複しているデータを削除しています。

*#入力データと出力データを'Point'データで結合*

data\_press\_type = pd.merge(data12\_dummy,data\_out,how = "outer",left\_on = 'From',right\_on='Point')

data\_press\_type

用意したdata12とdata\_outをPointの列で結合しています。

*#'Support.csv'の'type'データの確認*

data\_series = Series(data\_press\_type['Type'])

pd.Series.value\_counts(data\_series)

結合したデータのTypeの列の値の種類を確認しています。次のプログラムはここで得られた種類によって変わります。

*#文字データなのでダミー化*

data\_press\_type.at[data\_press\_type['Type'] == "Guide",'Type'] = 1

data\_press\_type.at[data\_press\_type['Type'] == "V - Stop",'Type'] = 2

data\_press\_type.at[data\_press\_type['Type'] == "Spr. Hgr",'Type'] = 3

data\_press\_type.at[data\_press\_type['Type'] == "Spr. Can",'Type'] = 4

data\_test\_press = data\_press\_type.fillna(0)

data\_series = Series(data\_test\_press['Type'])

print(pd.Series.value\_counts(data\_series))

data\_test\_press = data\_test\_press[data\_test\_press["呼び径 (mm)\_x"] != 0]

data\_test\_press

先ほどのプログラムにより得られた値の種類はすべて文字データなのでそれぞれを数字で置き換えています。data\_press\_type.fillna(0)はデータの入っていない（サポートがついていない）部分を0で置き換えています。

最後のdata\_test\_press[data\_test\_press["呼び径 (mm)\_x"] != 0]は呼び径が入力されていないデータを削除しています。今回は削除していますがサポートのデータのみ入力されている部分でもあるのでここに関しては検討する必要があると思います。

np.random.seed(0) *# 乱数を固定値で初期化し再現性を持たせる*

X = data\_test\_press.drop(['From','Point','Type'],axis=1)

T = data\_test\_press['Type']

(n\_samples,n\_features)=X.shape

n\_classes=len(np.unique(T))

print(n\_features)

T = np\_utils.to\_categorical(T)

*# 数値を、位置に変換 [0,1,2] ==> [ [1,0,0],[0,1,0],[0,0,1] ]*

train\_x, test\_x, train\_t, test\_t = train\_test\_split(X, T, train\_size=0.7, test\_size=0.3)

*#モデル作成*

model = Sequential()

model.add(Dense(100,activation="relu",input\_shape=(n\_features,)))

model.add(Dropout(0.2))

model.add(Dense(50,activation="relu"))

model.add(Dropout(0.2))

model.add(Dense(100,activation="relu"))

model.add(Dropout(0.2))

model.add(Dense(units=n\_classes,activation="softmax"))

**from** **keras.utils** **import** plot\_model

plot\_model(model,show\_shapes=**True**, to\_file='model.png')

model.summary()

model.compile(loss='mean\_squared\_error', optimizer=Adam(),metrics=["accuracy"])

ここまでで学習に使用するデータの変換は終わりましたのでここからは学習する条件やデータを入力用（X）と出力用（T）の訓練データ、テストデータに分けています。ここに関してはデータを増やした時には入力、出力ともに変更可能です。また、＃モデル作成以後の部分に関しても学習に使用するモデルによって変更します。

*#トレーニング*

history=model.fit(X,T,epochs=600,batch\_size=100,verbose=1,validation\_data=(test\_x,test\_t))

result=model.predict\_classes(X,verbose=0)

以上の条件により学習を行っています。ここで指定しているエポック数やバッチサイズなどに関しても検討ののち変更可能です。

*#以下で損失関数のグラフを描画*

val\_loss,=plt.plot(history.history["val\_loss"],c="orange",ls="--",linewidth=3)

loss,=plt.plot(history.history["loss"],c="blue",linewidth=3)

plt.legend([loss,val\_loss],["loss","val\_loss"])

plt.xlabel("epoch")

plt.ylabel("loss")

plt.show()

*#accuracyをグラフ化*

val\_acc,=plt.plot(history.history["val\_acc"],c="yellowgreen",ls="--",linewidth=3)

acc,=plt.plot(history.history["acc"],c="red",linewidth=3)

plt.legend([acc,val\_acc],["acc","val\_acc"])

plt.xlabel("epoch")

plt.ylabel("acc")

plt.show()

ここでは出力結果の表示を行っています。表示させたいデータを増やす場合には同様の方法で表示することが可能です。

c50.ipynbに関しては以上ですが＊\_merge.ipynbのなかで次の関数があるので説明します。

data\_test = pd.concat([data\_test\_press\_c50,data\_test\_press\_q10])

これは[]内のデータを行基準で結合しています。