Particle detection, Case Study 6

Sanjay Pillay - Wednesday, Oct 29, 2021

**Abstract**

The report uses deep learning model to identify particle producing collisions from background source and investigate the most appropriate tuning parameters for the model.

# Introduction

The study of minute particles that make up matter and radiations is called High Energy Physics (HEP). The study is used to experiment and search for signatures of rare particles which is learned by using monte Carlo simulations of the decaying product that is caused by collision of these particles [2]. The creation of such high energy particle is done using particle accelerators.

The application of these high energy partials is producing rare medical isotopes for research and medical treatments or used in radio therapy. Development of new superconductor material has also been pushed due to this. Other applications are in the area of medical, security, computing, science etc. [3]

In this paper we use the dataset hosted by UCI machine learning repository [1] called HEPMASS dataset. We use various deep learning models as a binary classifier to predict if a collision results in a new particle or not.

# Method

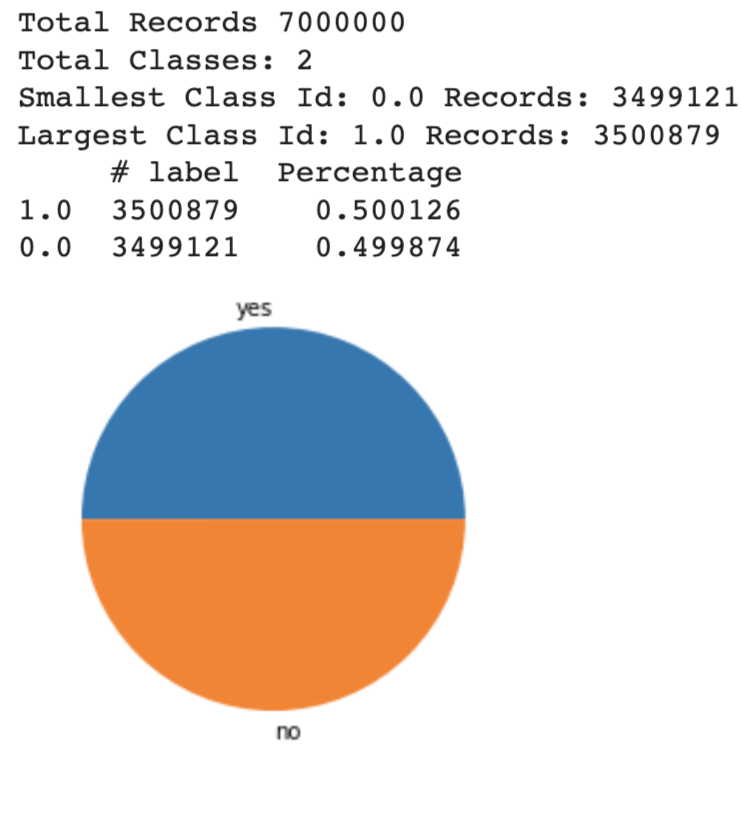
The large HEPMASS dataset that has labeled data of a collisions resulting in particle presence and its associated features was analyzed, scaled and used as training a series of deep neural network models. Various models were analyzed for their tuning parameters such as number of layers, number of nodes, batch size etc. to identify the best performing model using the accuracy score. The best identified model was than evaluated for f1 score, its confusion matrix (CM) and Receiver Operating Characteristics (ROC) / Area Under the Curve (AUC).

We first made a single stratified shuffle split of 90/10 % Where 90% was used for model building and 10% was our hold out for evaluation. The model building split was further split into 80/20 Train/Test split using the same strategy. The models were trained on the 80% train split and evaluated on the 20% test split. The CM and f1 statistics for the best model identified was carried out on the 10% holdout data of the best model identified.

## Data

The data consists of 28 features of complex scientific data points for 7 million collisions and the binary target label that indicates if the collision resulted in a new particle or not. The histogram of most [5.2] variables show normal distribution with no missing values, also none of the features had any significant correlation between each other so all the rows and features were included in our models. The features were scaled using standard scalar to be between 0 and 1. The target class was evenly distributed as shown in [Figure 1]

Figure 1



[Table 1] below shows the final shape of the data used, 10% of the original data set was held back to evaluate the best model’s performance and the remaining 90% was split into Train / Validation data using 80/20 split. Both the splits used a stratified split to maintain original class balance.

Table 1

|  |  |  |
| --- | --- | --- |
| Hold Out | Training | Validation |
| (700000, 28)  (700000,) | (5040000, 28)  (5040000,) | (1260000, 28)  (1260000,) |

## Models

The models that were evaluated were dense neural network consisting of 3 and 5 layers using relu activation and each was tuned using nodes varying between 400 to 800 at the first layer and 8 nodes at the layer before the output layer as it’s a good recommendation of having the last layer around ¼ the number of features and nodes in the middle layers gradually decreasing by the fraction of the difference of first layer nodes and last layer nodes by number of layers [(num\_start\_nodes - num\_end\_nodes)/num\_layers]. The models also used two batch sizes namely 10000 and 20000. We used epoch size of 500 and an early stopping criterion was specified with a patience of 5 allowing the training to stop if there was no improvement seen in the validation score for 5 consecutive epochs.

# Results

Table 2 shows each models parameter its total trainable parameters the number of epochs before early stopping kicks in and the model accuracy. The 5 layer in general performs better than 3 layers. In the 5-layer model the models with batch size of 10000 was slightly better than the ones with 20000. We finally chose the model highlighted in red which had the highest accuracy of 88.5, batch of 10000 and nodes of (600, 452, 304, 156, 1)

Table 2

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Total Param** | **Epochs** | **Accuracy** |
| Activation: relu layers: 3  nodes: 400, 204, 1  batch: 10000 | 93,609 | 70 | 87.81 |
| Activation: relu layers: 3  nodes: 600, 304, 1  batch: 10000 | 200,409 | 68 | 88.13 |
| Activation: relu layers: 3  nodes: 800, 404, 1  batch: 10000 | 347,209 | 86 | 88.36 |
| Activation: relu layers: 3  nodes: 400, 204, 1  batch: 20000 | 93,609 | 45 | 85.61 |
| Activation: relu layers: 3  nodes: 600, 304, 1  batch: 20000 | 200,409 | 18 | 83.62 |
| Activation: relu layers: 3  nodes: 800, 404, 1  batch: 20000 | 347,209 | 34 | 85.55 |
| Activation: relu layers: 5  nodes: 400, 302, 204, 106, 1  batch: 10000 | 216,351 | 49 | 88.17 |
| Activation: relu layers: 5  nodes: 600, 452, 304, 156, 1  batch: 10000 | 474,501 | 66 | 88.5 |
| Activation: relu layers: 5  nodes: 800, 602, 404, 206, 1  batch: 10000 | 832,651 | 61 | 88.4 |
| Activation: relu layers: 5  nodes: 400, 302, 204, 106, 1  batch: 20000 | 216,351 | 89 | 88.1 |
| Activation: relu layers: 5  nodes: 600, 452, 304, 156, 1  batch: 20000 | 474,501 | 89 | 88.22 |
| Activation: relu layers: 5  nodes: 800, 602, 404, 206, 1  batch: 20000 | 832,651 | 50 | 87.75 |

### Best Model Results

We evaluated the best performing model on the 10% holdout set. Figure 2 shows the drop in validation loss and increase in accuracy as the model trains, the model stops improving after 55 epochs. As seen in Figure 3 the over model f1 score 80%, the f1 score for class Yes is 82% and that for No is 77%, the AUC and Confusion Matrix (CM) are shown in Figure 4, the area under the curve is 95.6 %

Figure 2

|  |  |
| --- | --- |
|  |  |

Figure 3

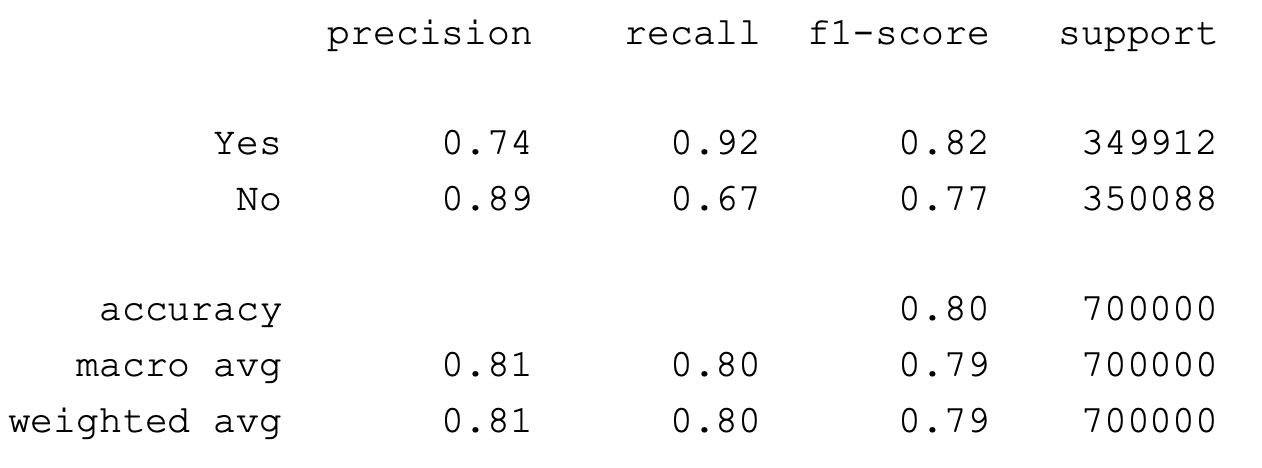


Figure 4

|  |  |
| --- | --- |
|  |  |

# Conclusion

The models were all generated using google colab using GPU and high memory to train the models in a reasonable time for this large dataset of 7 millions rows. Using 5-layers and 474501 tunable parameters with batch size of 10000 we were able to train the model within 54 epochs with an early stopping of 5 epochs over validation loss. The best accuracy of the model was around 88.5% with an f1 score of 80%. The AUC derived using 50% threshold was 95.6%, we can potentially increase the threshold if there is a need to increase the potential of not missing a collision that results in a new particle. We can help the model train further by increasing the nodes (parameters) within the layers and tryout more tuning parameters such as learning rate, batch sizes and increasing the patience from 5 to 10 but that would need more resources.

# Appendix

## Code

**Some of the output has been cleaned to reduce document.**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **import** **os**  **import** **email**  *#All Python module imports*  *#https://pandas.pydata.org/docs/user\_guide/index.html#user-guide*  **import** **pandas** **as** **pd** *#Pandas Dataframe module*  **import** **numpy** **as** **np**  **from** **math** **import** pi  *#scikit learn*  *#https://scikit-learn.org/stable/modules/classes.html#module-sklearn.linear\_model*  **import** **sklearn** **as** **skl**  *#https://seaborn.pydata.org*  **import** **seaborn** **as** **sns**  **import** **matplotlib.pyplot** **as** **plt**  **import** **matplotlib**  **import** **warnings**  *#Module for formating table for documentation*  *#https://pypi.org/project/tabulate/*  **from** **tabulate** **import** tabulate  **from** **IPython.display** **import** display, Markdown  *#Interactive mode*  **from** **IPython.core.interactiveshell** **import** InteractiveShell  InteractiveShell.ast\_node\_interactivity = "all"  **from** **IPython.display** **import** Image  **from** **sklearn.preprocessing** **import** MinMaxScaler  **from** **sklearn.feature\_selection** **import** SelectKBest, chi2  **from** **sklearn.model\_selection** **import** StratifiedShuffleSplit  **from** **sklearn.preprocessing** **import** StandardScaler  **from** **sklearn.linear\_model** **import** LogisticRegression  **from** **sklearn** **import** metrics **as** mt  **from** **sklearn.metrics** **import** plot\_confusion\_matrix  **from** **sklearn.model\_selection** **import** cross\_val\_score  **from** **sklearn.metrics** **import** classification\_report  **from** **sklearn.linear\_model** **import** LogisticRegression  **from** **sklearn.svm** **import** SVC  **from** **sklearn.decomposition** **import** PCA  **from** **sklearn.metrics** **import** confusion\_matrix  **from** **sklearn.metrics** **import** f1\_score, accuracy\_score  **from** **sklearn.model\_selection** **import** KFold, StratifiedKFold  **from** **sklearn.model\_selection** **import** GridSearchCV **as** gridcv  **from** **sklearn** **import** preprocessing  **from** **sklearn.model\_selection** **import** cross\_validate  **from** **sklearn.metrics** **import** make\_scorer  **from** **sklearn.metrics** **import** mean\_squared\_error  **from** **sklearn.metrics** **import** mean\_absolute\_error  **from** **sklearn.metrics** **import** r2\_score  **import** **pprint**  **import** **re**  **from** **sklearn.model\_selection** **import** cross\_val\_predict  **from** **html.parser** **import** HTMLParser  **from** **bs4** **import** BeautifulSoup  **import** **nltk**  **from** **nltk.corpus** **import** stopwords  **from** **sklearn.feature\_extraction.text** **import** TfidfVectorizer  **from** **sklearn.metrics** **import** roc\_curve  **from** **sklearn.metrics** **import** roc\_auc\_score  **from** **scipy.io** **import** arff  **from** **statsmodels.imputation** **import** mice  **import** **statsmodels** **as** **sm**  **from** **xgboost** **import** XGBClassifier  **from** **numpy** **import** arange  **from** **numpy** **import** argmax  **from** **sklearn.preprocessing** **import** QuantileTransformer  **import** **tensorflow** **as** **tf**  **import** **math**  **from** **tensorflow.keras.models** **import** Sequential  **from** **tensorflow.keras.layers** **import** Dense  **from** **tensorflow.keras.wrappers.scikit\_learn** **import** KerasClassifier  **from** **sklearn.preprocessing** **import** MinMaxScaler  **from** **sklearn.model\_selection** **import** train\_test\_split  **from** **sklearn.model\_selection** **import** GridSearchCV, RandomizedSearchCV  print(tf.\_\_version\_\_)  **import** **warnings**  warnings.filterwarnings('ignore')  **from** **yellowbrick.classifier** **import** ROCAUC  **import** **yellowbrick**  print(yellowbrick.\_\_version\_\_)  /usr/local/lib/python3.7/dist-packages/statsmodels/tools/\_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing instead.  import pandas.util.testing as tm  2.7.0  0.9.1  In [2]:  **from** **google.colab** **import** drive  drive.mount('/content/drive')  Mounted at /content/drive  In [3]:  os.getcwd()  df = pd.read\_csv('./drive/MyDrive/data/all\_train.csv')  df.shape  df.head()  Out[3]:  '/content'  Out[3]:  (7000000, 29)  Out[3]:  In [ ]:  df['# label'].value\_counts()  Out[ ]:  1.0 3500879  0.0 3499121  Name: # label, dtype: int64  In [ ]:  *#Check class distribution*  %**matplotlib** inline  *# Adapted from:*  *# https://www.featureranking.com/tutorials/machine-learning-tutorials/information-gain-computation/*  **def** gini\_index(y):  probs = pd.value\_counts(y,normalize=**True**)  **return** 1 - np.sum(np.square(probs))  **def** plot\_class\_dist(y):  class\_ct = len(np.unique(y['# label']))  vc = pd.value\_counts(y['# label'])  print('Total Records', len(y['# label']))  print('Total Classes:', class\_ct)  print('Smallest Class Id:',vc.idxmin(),'Records:',vc.min())  print('Largest Class Id:',vc.idxmax(),'Records:',vc.max())    position\_counts = pd.DataFrame(y['# label'].value\_counts())  position\_counts['Percentage'] = position\_counts['# label']/position\_counts.sum()[0]  print(position\_counts)  plt.figure(figsize=(4,4))  plt.pie(position\_counts['Percentage'],labels = ['yes', 'no']);    plot\_class\_dist(df.iloc[:,0:1])  Total Records 7000000  Total Classes: 2  Smallest Class Id: 0.0 Records: 3499121  Largest Class Id: 1.0 Records: 3500879  # label Percentage  1.0 3500879 0.500126  0.0 3499121 0.499874    In [ ]:  df.describe().T  Out[ ]:   |  | **count** | **mean** | **std** | **min** | **25%** | **50%** | **75%** | **max** | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **# label** | 7000000.0 | 0.500126 | 0.500000 | 0.000000 | 0.000000 | 1.000000 | 1.000000 | 1.000000 | | **f0** | 7000000.0 | 0.016125 | 1.004417 | -1.960549 | -0.728821 | -0.039303 | 0.690080 | 4.378282 | | **f1** | 7000000.0 | 0.000477 | 0.997486 | -2.365355 | -0.733255 | 0.000852 | 0.734783 | 2.365287 | | **f2** | 7000000.0 | 0.000027 | 1.000080 | -1.732165 | -0.865670 | 0.000320 | 0.865946 | 1.732370 | | **f3** | 7000000.0 | 0.010561 | 0.995600 | -9.980274 | -0.609229 | 0.019633 | 0.679882 | 4.148023 | | **f4** | 7000000.0 | -0.000105 | 0.999867 | -1.732137 | -0.865802 | -0.000507 | 0.865765 | 1.731978 | | **f5** | 7000000.0 | 0.002766 | 1.000957 | -1.054221 | -1.054221 | -0.005984 | 0.850488 | 4.482618 | | **f6** | 7000000.0 | 0.018160 | 0.986775 | -3.034787 | -0.756609 | -0.149953 | 0.768669 | 3.720345 | | **f7** | 7000000.0 | 0.000025 | 0.996587 | -2.757853 | -0.701415 | -0.000107 | 0.701319 | 2.758590 | | **f8** | 7000000.0 | 0.000435 | 1.000007 | -1.732359 | -0.865654 | 0.001385 | 0.866598 | 1.731450 | | **f9** | 7000000.0 | -0.006870 | 1.001938 | -1.325801 | -1.325801 | 0.754261 | 0.754261 | 0.754261 | | **f10** | 7000000.0 | 0.017543 | 0.994151 | -2.835563 | -0.723727 | -0.128573 | 0.647864 | 4.639335 | | **f11** | 7000000.0 | -0.000161 | 0.998450 | -2.602091 | -0.703293 | -0.000576 | 0.704100 | 2.602294 | | **f12** | 7000000.0 | -0.000329 | 1.000078 | -1.732216 | -0.866599 | -0.001282 | 0.865832 | 1.732007 | | **f13** | 7000000.0 | 0.001739 | 0.999737 | -1.161915 | -1.161915 | 0.860649 | 0.860649 | 0.860649 | | **f14** | 7000000.0 | 0.017246 | 0.999465 | -2.454879 | -0.699618 | -0.097493 | 0.634705 | 5.535799 | | **f15** | 7000000.0 | 0.000483 | 0.998429 | -2.437812 | -0.707026 | 0.000298 | 0.708371 | 2.438369 | | **f16** | 7000000.0 | -0.000554 | 0.999861 | -1.732145 | -0.866247 | -0.001377 | 0.864942 | 1.732738 | | **f17** | 7000000.0 | 0.004960 | 1.001006 | -0.815440 | -0.815440 | -0.815440 | 1.226331 | 1.226331 | | **f18** | 7000000.0 | 0.011648 | 1.002725 | -1.728284 | -0.742363 | -0.089925 | 0.642319 | 5.866367 | | **f19** | 7000000.0 | -0.000113 | 1.000038 | -2.281867 | -0.720685 | -0.000067 | 0.720492 | 2.282217 | | **f20** | 7000000.0 | 0.000077 | 1.000033 | -1.731758 | -0.865685 | -0.000442 | 0.865957 | 1.732740 | | **f21** | 7000000.0 | 0.000291 | 1.000170 | -0.573682 | -0.573682 | -0.573682 | -0.573682 | 1.743123 | | **f22** | 7000000.0 | 0.012288 | 1.010477 | -3.631608 | -0.541794 | -0.160276 | 0.481219 | 7.293420 | | **f23** | 7000000.0 | 0.009778 | 1.005418 | -4.729473 | -0.511552 | -0.314403 | 0.163489 | 9.333287 | | **f24** | 7000000.0 | 0.005270 | 1.009990 | -20.622229 | -0.354387 | -0.326523 | -0.233767 | 14.990636 | | **f25** | 7000000.0 | -0.001761 | 0.984451 | -3.452634 | -0.692510 | -0.357030 | 0.475313 | 5.277313 | | **f26** | 7000000.0 | 0.015331 | 0.982280 | -2.632761 | -0.794380 | -0.088286 | 0.761085 | 4.444690 | | **mass** | 7000000.0 | 1000.107387 | 353.425487 | 499.999969 | 750.000000 | 1000.000000 | 1250.000000 | 1500.000000 |   In [ ]:  **def** print\_highly\_correlated(df, features, t=0.8):  *#Method will extractout featuresthat are corelated based on thresh hold*  l = []  c\_df = df[features].corr() *# get correlations*  cor\_features = np.where(np.abs(c\_df) > t) *# nparray method*  cor\_features = [(c\_df.iloc[x,y], x, y) **for** x, y **in** zip(\*cor\_features) **if** x != y **and** x < y]  *#try sorting*  corr\_list = sorted(cor\_features, key=**lambda** x: -abs(x[0]))  **if** corr\_list == []:  print("Nothing above: ", t)  **else**:    **for** v, i, j **in** corr\_list:  cols = df[features].columns  **if** c\_df.index[i] **not** **in** l:  l.append(c\_df.index[i])  **if** c\_df.index[j] **not** **in** l:  l.append(c\_df.index[j])  print ("**%s** and **%s** = **%.3f**" % (c\_df.index[i], c\_df.columns[j], v))  **return** l  print\_highly\_correlated(df, df.columns, t=0.96)  *#prepare the plot pallete*  *#cmap = sns.diverging\_palette(220, 10, as\_cmap=True) # one of the many color mappings*  *#sns.set(style="darkgrid") # one of the many styles to plot using*  *#f, ax = plt.subplots(figsize=(25, 25))*  *#%time sns.heatmap(df\_imputed[print\_highly\_correlated(df, df.columns, t=0.99)].corr(), cmap=cmap, fmt=".2f",annot=True);*  *#f.tight\_layout();*  Nothing above: 0.96  Out[ ]:  []  In [ ]:  percent\_missing = df.isnull().sum() \* 100 / len(df)  missing\_value\_df = pd.DataFrame({'column\_name': df.columns,  'percent\_missing': percent\_missing})  missing\_value\_df.sort\_values('percent\_missing', inplace=**True**, ascending=**False**)  missing\_value\_df.head(15)  Out[ ]:   |  | **column\_name** | **percent\_missing** | | --- | --- | --- | | **# label** | # label | 0.0 | | **f14** | f14 | 0.0 | | **f26** | f26 | 0.0 | | **f25** | f25 | 0.0 | | **f24** | f24 | 0.0 | | **f23** | f23 | 0.0 | | **f22** | f22 | 0.0 | | **f21** | f21 | 0.0 | | **f20** | f20 | 0.0 | | **f19** | f19 | 0.0 | | **f18** | f18 | 0.0 | | **f17** | f17 | 0.0 | | **f16** | f16 | 0.0 | | **f15** | f15 | 0.0 | | **f13** | f13 | 0.0 |   In [4]:  X = df.iloc[:,1:].values  X.shape  y = df['# label'].values  y.shape  *#Normalize data*  *##Scale the transformed data ### for relu 0, 1*  scl\_obj = MinMaxScaler(feature\_range=[0, 1]) *#StandardScaler()*  scl\_obj.fit(X)  X\_scaled = scl\_obj.transform(X)  *#QuantileTransformer(output\_distribution='uniform').fit\_transform(X))*  X\_scaled.shape  *#X\_scaled*  Out[4]:  (7000000, 28)  Out[4]:  (7000000,)  Out[4]:  MinMaxScaler(copy=True, feature\_range=[0, 1])  Out[4]:  (7000000, 28)  In [ ]:  scaled\_train\_df = pd.DataFrame(X\_scaled, columns=df.iloc[:,1:].columns.values)  **for** i **in** scaled\_train\_df:  scaled\_train\_df[i].hist()  plt.show()  Out[ ]:  <matplotlib.axes.\_subplots.AxesSubplot at 0x7f8265673610>    Out[ ]:  <matplotlib.axes.\_subplots.AxesSubplot at 0x7f82655f3350>    Out[ ]:  <matplotlib.axes.\_subplots.AxesSubplot at 0x7f8265547dd0>    Out[ ]:  <matplotlib.axes.\_subplots.AxesSubplot at 0x7f82655291d0>    Out[ ]:  <matplotlib.axes.\_subplots.AxesSubplot at 0x7f82654a0cd0>    Out[ ]:  <matplotlib.axes.\_subplots.AxesSubplot at 0x7f82654a0590>  Out[ ]:  <matplotlib.axes.\_subplots.AxesSubplot at 0x7f8265342b50>    Out[ ]:  <matplotlib.axes.\_subplots.AxesSubplot at 0x7f826527b490>    Out[ ]:  <matplotlib.axes.\_subplots.AxesSubplot at 0x7f82652677d0>      Out[ ]:  <matplotlib.axes.\_subplots.AxesSubplot at 0x7f826509b110>    Out[ ]:  <matplotlib.axes.\_subplots.AxesSubplot at 0x7f82651223d0>    Out[ ]:    Out[ ]:  <matplotlib.axes.\_subplots.AxesSubplot at 0x7f8264e5d410>    Out[ ]:    Out[ ]:  <matplotlib.axes.\_subplots.AxesSubplot at 0x7f82651a4910>    Out[ ]:  <matplotlib.axes.\_subplots.AxesSubplot at 0x7f8265509d50>    Out[ ]:    Out[ ]:  <matplotlib.axes.\_subplots.AxesSubplot at 0x7f82652cf610>    Out[ ]:  <matplotlib.axes.\_subplots.AxesSubplot at 0x7f826503c090>    In [5]:  *#modeltrain/hold 90/10 stratified*  stt = StratifiedShuffleSplit(n\_splits=1, test\_size=0.1, random\_state=45)  train\_index\_clf, test\_index\_clf = next(stt.split(X\_scaled, y))  X\_trainmodel = X[train\_index\_clf]  y\_trainmodel = y[train\_index\_clf].ravel()  X\_hold = X[test\_index\_clf]  y\_hold = y[test\_index\_clf].ravel()  X\_trainmodel.shape  y\_trainmodel.shape  X\_hold.shape  y\_hold.shape  Out[5]:  (6300000, 28)  Out[5]:  (6300000,)  Out[5]:  (700000, 28)  Out[5]:  (700000,)  In [6]:  *#train/test 80/20 stratified*  stt = StratifiedShuffleSplit(n\_splits=1, test\_size=0.2, random\_state=45)  train\_index\_clf, test\_index\_clf = next(stt.split(X\_trainmodel, y\_trainmodel))  X\_train = X[train\_index\_clf]  y\_train = y[train\_index\_clf].ravel()  X\_test = X[test\_index\_clf]  y\_test = y[test\_index\_clf].ravel()  X\_train.shape  y\_train.shape  X\_test.shape  y\_test.shape  Out[6]:  (5040000, 28)  Out[6]:  (5040000,)  Out[6]:  (1260000, 28)  Out[6]:  (1260000,)  In [7]:  **def** FindLayerNodesLinear(n\_layers, first\_layer\_nodes, last\_layer\_nodes):  layers = []    nodes\_increment = (last\_layer\_nodes - first\_layer\_nodes)/ (n\_layers-1)  nodes = first\_layer\_nodes  **for** i **in** range(1, n\_layers+1):  layers.append(math.ceil(nodes))  nodes = nodes + nodes\_increment    **return** layers  In [8]:  **from** **tensorflow.keras.callbacks** **import** EarlyStopping  model\_clf\_stats = pd.DataFrame()  **def** createmodel(n\_layers, first\_layer\_nodes, last\_layer\_nodes, activation\_func, loss\_func):  model = Sequential()  n\_nodes = FindLayerNodesLinear(n\_layers, first\_layer\_nodes, last\_layer\_nodes)  **for** i **in** range(1, n\_layers):    **if** i==1:  print("building node:",i)  model.add(Dense(first\_layer\_nodes, input\_dim=X\_train.shape[1], activation=activation\_func))  **else**:  print("building node:",i)  model.add(Dense(n\_nodes[i-1], activation=activation\_func))    *#Finally, the output layer should have a single node in binary classification*  model.add(Dense(1, activation='sigmoid'))  model.compile(optimizer='adam', loss=loss\_func, metrics = ["accuracy"]) *#note: metrics could also be 'mse'*    **return** model  In [9]:  **def** test\_model(layers, start, end, activation, batch, X\_train, y\_train, X\_test, y\_test, ver=1):  *#relu, l=5, nodes=600, e\_nodes=8, e=500, b=20000*  print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*Execution started for\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  print("Activation:",activation," layers:", layers, " nodes:", start," batch:", batch)  safety = EarlyStopping(monitor='val\_loss', patience=5)  seed = 45 *#88.27*  m = createmodel(n\_layers=layers, first\_layer\_nodes=start, last\_layer\_nodes=end,  activation\_func=activation, loss\_func=tf.keras.losses.BinaryCrossentropy()) *#tanh*  hist = m.fit(X\_train, y\_train, epochs=500, batch\_size=batch,  validation\_data=(X\_test, y\_test), callbacks=[safety], verbose=ver) *# add validation left out here*  best\_score = max(hist.history['accuracy'])  print("Best score: ",best\_score)  model\_stat = pd.DataFrame()  model\_stat['Accuracy'] = [best\_score]  model\_stat['Model'] = ["Activation:" + activation + " layers:" + str(layers) + " nodes:" + str(start) + " batch:" + str(batch)]  m.summary()  tf.keras.backend.clear\_session()  **del** m  print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*Execution ended\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***\n\n**")  **return** model\_stat  In [63]:  *#p, m = test\_model(3, 400, 8, 'relu', 10000, X\_train, y\_train, X\_test, y\_test)*  *#tf.keras.backend.clear\_session()*  *#del m*  In [66]:  p = test\_model(3, 400, 8, 'relu', 10000, X\_train, y\_train, X\_test, y\_test)  model\_clf\_stats = model\_clf\_stats.append(p)  p = test\_model(3, 600, 8, 'relu', 10000, X\_train, y\_train, X\_test, y\_test)  model\_clf\_stats = model\_clf\_stats.append(p)  p = test\_model(3, 800, 8, 'relu', 10000, X\_train, y\_train, X\_test, y\_test)  model\_clf\_stats = model\_clf\_stats.append(p)  model\_clf\_stats  \*\*\*\*\*\*\*\*\*\*\*\*\*\*Execution started for\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Activation: relu layers: 3 nodes: 400 batch: 10000  building node: 1  building node: 2  Epoch 1/500  504/504 [==============================] - 3s 5ms/step - loss: 1.2628 - accuracy: 0.7409 - val\_loss: 0.4395 - val\_accuracy: 0.7955  Epoch 2/500  504/504 [==============================] - 2s 4ms/step - loss: 0.5452 - accuracy: 0.7825 - val\_loss: 5.9589 - val\_accuracy: 0.4992  Epoch 3/500  …  Epoch 70/500  504/504 [==============================] - 2s 4ms/step - loss: 0.2698 - accuracy: 0.8782 - val\_loss: 0.2707 - val\_accuracy: 0.8777  Best score: 0.8781597018241882  Model: "sequential\_2"  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Layer (type) Output Shape Param #  =================================================================  dense\_6 (Dense) (None, 400) 11600    dense\_7 (Dense) (None, 204) 81804    dense\_8 (Dense) (None, 1) 205    =================================================================  Total params: 93,609  Trainable params: 93,609  Non-trainable params: 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \*\*\*\*\*\*\*\*\*\*\*\*\*\*Execution ended\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*Execution started for\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Activation: relu layers: 3 nodes: 600 batch: 10000  building node: 1  building node: 2  Epoch 1/500  504/504 [==============================] - 3s 6ms/step - loss: 1.4198 - accuracy: 0.7220 - val\_loss: 1.6064 - val\_accuracy: 0.5557  …  504/504 [==============================] - 3s 6ms/step - loss: 0.2640 - accuracy: 0.8814 - val\_loss: 0.2662 - val\_accuracy: 0.8802  Best score: 0.8813777565956116  Model: "sequential"  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Layer (type) Output Shape Param #  =================================================================  dense (Dense) (None, 600) 17400    dense\_1 (Dense) (None, 304) 182704    dense\_2 (Dense) (None, 1) 305    =================================================================  Total params: 200,409  Trainable params: 200,409  Non-trainable params: 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \*\*\*\*\*\*\*\*\*\*\*\*\*\*Execution ended\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*Execution started for\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Activation: relu layers: 3 nodes: 800 batch: 10000  building node: 1  building node: 2  Epoch 1/500  504/504 [==============================] - 4s 7ms/step - loss: 2.3333 - accuracy: 0.7142 - val\_loss: 0.4521 - val\_accuracy: 0.8181  Epoch 2/500  504/504 [==============================] - 3s 7ms/step - loss: 0.6233 - accuracy: 0.7695 - val\_loss: 0.3900 - val\_accuracy: 0.8263  …  504/504 [==============================] - 4s 7ms/step - loss: 0.2600 - accuracy: 0.8836 - val\_loss: 0.2630 - val\_accuracy: 0.8819  Best score: 0.8836385011672974  Model: "sequential"  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Layer (type) Output Shape Param #  =================================================================  dense (Dense) (None, 800) 23200    dense\_1 (Dense) (None, 404) 323604    dense\_2 (Dense) (None, 1) 405    =================================================================  Total params: 347,209  Trainable params: 347,209  Non-trainable params: 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \*\*\*\*\*\*\*\*\*\*\*\*\*\*Execution ended\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Out[66]:   |  | **Accuracy** | **Model** | | --- | --- | --- | | **0** | 0.878160 | Activation:relu layers:3 nodes:400 batch:10000 | | **0** | 0.881378 | Activation:relu layers:3 nodes:600 batch:10000 | | **0** | 0.883639 | Activation:relu layers:3 nodes:800 batch:10000 |   In [67]:  p = test\_model(3, 400, 8, 'relu', 20000, X\_train, y\_train, X\_test, y\_test)  model\_clf\_stats = model\_clf\_stats.append(p)  p = test\_model(3, 600, 8, 'relu', 20000, X\_train, y\_train, X\_test, y\_test)  model\_clf\_stats = model\_clf\_stats.append(p)  p = test\_model(3, 800, 8, 'relu', 20000, X\_train, y\_train, X\_test, y\_test)  model\_clf\_stats = model\_clf\_stats.append(p)  model\_clf\_stats  \*\*\*\*\*\*\*\*\*\*\*\*\*\*Execution started for\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Activation: relu layers: 3 nodes: 400 batch: 20000  building node: 1  building node: 2  Epoch 1/500  252/252 [==============================] - 2s 7ms/step - loss: 1.9611 - accuracy: 0.6956 - val\_loss: 0.4396 - val\_accuracy: 0.8073  …  252/252 [==============================] - 1s 6ms/step - loss: 0.3127 - accuracy: 0.8561 - val\_loss: 0.3115 - val\_accuracy: 0.8579  Epoch 45/500  252/252 [==============================] - 1s 6ms/step - loss: 0.3187 - accuracy: 0.8530 - val\_loss: 0.3098 - val\_accuracy: 0.8578  Best score: 0.8561081290245056  Model: "sequential"  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Layer (type) Output Shape Param #  =================================================================  dense (Dense) (None, 400) 11600    dense\_1 (Dense) (None, 204) 81804    dense\_2 (Dense) (None, 1) 205    =================================================================  Total params: 93,609  Trainable params: 93,609  Non-trainable params: 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \*\*\*\*\*\*\*\*\*\*\*\*\*\*Execution ended\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*Execution started for\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Activation: relu layers: 3 nodes: 600 batch: 20000  building node: 1  building node: 2  Epoch 1/500  252/252 [==============================] - 3s 10ms/step - loss: 3.6500 - accuracy: 0.6804 - val\_loss: 0.4217 - val\_accuracy: 0.8168  Epoch 2/500  252/252 [==============================] - 2s 9ms/step - loss: 0.5889 - accuracy: 0.7594 - val\_loss: 0.4149 - val\_accuracy: 0.8231  …  252/252 [==============================] - 2s 9ms/step - loss: 0.3760 - accuracy: 0.8283 - val\_loss: 0.3551 - val\_accuracy: 0.8303  Epoch 16/500  252/252 [==============================] - 2s 9ms/step - loss: 0.3759 - accuracy: 0.8277 - val\_loss: 0.4047 - val\_accuracy: 0.8053  Epoch 17/500  252/252 [==============================] - 2s 9ms/step - loss: 0.4296 - accuracy: 0.8141 - val\_loss: 0.3726 - val\_accuracy: 0.8240  Epoch 18/500  252/252 [==============================] - 2s 9ms/step - loss: 0.4188 - accuracy: 0.8191 - val\_loss: 0.3583 - val\_accuracy: 0.8378  Best score: 0.8362652659416199  Model: "sequential"  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Layer (type) Output Shape Param #  =================================================================  dense (Dense) (None, 600) 17400    dense\_1 (Dense) (None, 304) 182704    dense\_2 (Dense) (None, 1) 305    =================================================================  Total params: 200,409  Trainable params: 200,409  Non-trainable params: 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \*\*\*\*\*\*\*\*\*\*\*\*\*\*Execution ended\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*Execution started for\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Activation: relu layers: 3 nodes: 800 batch: 20000  building node: 1  building node: 2  Epoch 1/500  252/252 [==============================] - 4s 13ms/step - loss: 4.3767 - accuracy: 0.6665 - val\_loss: 0.4411 - val\_accuracy: 0.8118  …  252/252 [==============================] - 3s 12ms/step - loss: 0.3140 - accuracy: 0.8546 - val\_loss: 0.3127 - val\_accuracy: 0.8542  Epoch 34/500  252/252 [==============================] - 3s 12ms/step - loss: 0.3125 - accuracy: 0.8551 - val\_loss: 0.3198 - val\_accuracy: 0.8554  Best score: 0.855080783367157  Model: "sequential"  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Layer (type) Output Shape Param #  =================================================================  dense (Dense) (None, 800) 23200    dense\_1 (Dense) (None, 404) 323604    dense\_2 (Dense) (None, 1) 405    =================================================================  Total params: 347,209  Trainable params: 347,209  Non-trainable params: 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \*\*\*\*\*\*\*\*\*\*\*\*\*\*Execution ended\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Out[67]:   |  | **Accuracy** | **Model** | | --- | --- | --- | | **0** | 0.878160 | Activation:relu layers:3 nodes:400 batch:10000 | | **0** | 0.881378 | Activation:relu layers:3 nodes:600 batch:10000 | | **0** | 0.883639 | Activation:relu layers:3 nodes:800 batch:10000 | | **0** | 0.856108 | Activation:relu layers:3 nodes:400 batch:20000 | | **0** | 0.836265 | Activation:relu layers:3 nodes:600 batch:20000 | | **0** | 0.855081 | Activation:relu layers:3 nodes:800 batch:20000 |   In [68]:  p = test\_model(5, 400, 8, 'relu', 10000, X\_train, y\_train, X\_test, y\_test)  model\_clf\_stats = model\_clf\_stats.append(p)  p = test\_model(5, 600, 8, 'relu', 10000, X\_train, y\_train, X\_test, y\_test)  model\_clf\_stats = model\_clf\_stats.append(p)  p = test\_model(5, 800, 8, 'relu', 10000, X\_train, y\_train, X\_test, y\_test)  model\_clf\_stats = model\_clf\_stats.append(p)  model\_clf\_stats  \*\*\*\*\*\*\*\*\*\*\*\*\*\*Execution started for\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Activation: relu layers: 5 nodes: 400 batch: 10000  building node: 1  building node: 2  building node: 3  building node: 4  Epoch 1/500  504/504 [==============================] - 4s 7ms/step - loss: 0.8635 - accuracy: 0.7412 - val\_loss: 0.3898 - val\_accuracy: 0.8235  Epoch 2/500  504/504 [==============================] - 3s 6ms/step - loss: 0.3927 - accuracy: 0.8176 - val\_loss: 0.3812 - val\_accuracy: 0.8197  …  504/504 [==============================] - 3s 6ms/step - loss: 0.2633 - accuracy: 0.8816 - val\_loss: 0.2649 - val\_accuracy: 0.8807  Epoch 48/500  504/504 [==============================] - 3s 6ms/step - loss: 0.2634 - accuracy: 0.8815 - val\_loss: 0.2647 - val\_accuracy: 0.8806  Epoch 49/500  504/504 [==============================] - 3s 6ms/step - loss: 0.2632 - accuracy: 0.8818 - val\_loss: 0.2652 - val\_accuracy: 0.8804  Best score: 0.8817861080169678  Model: "sequential"  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Layer (type) Output Shape Param #  =================================================================  dense (Dense) (None, 400) 11600    dense\_1 (Dense) (None, 302) 121102    dense\_2 (Dense) (None, 204) 61812    dense\_3 (Dense) (None, 106) 21730    dense\_4 (Dense) (None, 1) 107    =================================================================  Total params: 216,351  Trainable params: 216,351  Non-trainable params: 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \*\*\*\*\*\*\*\*\*\*\*\*\*\*Execution ended\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*Execution started for\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Activation: relu layers: 5 nodes: 600 batch: 10000  building node: 1  building node: 2  building node: 3  building node: 4  Epoch 1/500  504/504 [==============================] - 6s 10ms/step - loss: 0.8281 - accuracy: 0.7444 - val\_loss: 0.3882 - val\_accuracy: 0.8230  Epoch 2/500  504/504 [==============================] - 5s 9ms/step - loss: 0.4050 - accuracy: 0.8108 - val\_loss: 0.3587 - val\_accuracy: 0.8306  …  504/504 [==============================] - 5s 9ms/step - loss: 0.2592 - accuracy: 0.8839 - val\_loss: 0.2629 - val\_accuracy: 0.8817  Epoch 65/500  504/504 [==============================] - 5s 9ms/step - loss: 0.2588 - accuracy: 0.8840 - val\_loss: 0.2636 - val\_accuracy: 0.8814  Epoch 66/500  504/504 [==============================] - 5s 9ms/step - loss: 0.2590 - accuracy: 0.8840 - val\_loss: 0.2627 - val\_accuracy: 0.8821  Best score: 0.8840420842170715  Model: "sequential"  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Layer (type) Output Shape Param #  =================================================================  dense (Dense) (None, 600) 17400    dense\_1 (Dense) (None, 452) 271652    dense\_2 (Dense) (None, 304) 137712    dense\_3 (Dense) (None, 156) 47580    dense\_4 (Dense) (None, 1) 157    =================================================================  Total params: 474,501  Trainable params: 474,501  Non-trainable params: 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \*\*\*\*\*\*\*\*\*\*\*\*\*\*Execution ended\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*Execution started for\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Activation: relu layers: 5 nodes: 800 batch: 10000  building node: 1  building node: 2  building node: 3  building node: 4  Epoch 1/500  504/504 [==============================] - 7s 14ms/step - loss: 1.1224 - accuracy: 0.7436 - val\_loss: 0.3881 - val\_accuracy: 0.8219  …  Epoch 59/500  504/504 [==============================] - 7s 13ms/step - loss: 0.2575 - accuracy: 0.8848 - val\_loss: 0.2634 - val\_accuracy: 0.8820  Epoch 60/500  504/504 [==============================] - 7s 13ms/step - loss: 0.2574 - accuracy: 0.8850 - val\_loss: 0.2615 - val\_accuracy: 0.8824  Epoch 61/500  504/504 [==============================] - 7s 13ms/step - loss: 0.2572 - accuracy: 0.8850 - val\_loss: 0.2618 - val\_accuracy: 0.8827  Best score: 0.8850095272064209  Model: "sequential"  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Layer (type) Output Shape Param #  =================================================================  dense (Dense) (None, 800) 23200    dense\_1 (Dense) (None, 602) 482202    dense\_2 (Dense) (None, 404) 243612    dense\_3 (Dense) (None, 206) 83430    dense\_4 (Dense) (None, 1) 207    =================================================================  Total params: 832,651  Trainable params: 832,651  Non-trainable params: 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \*\*\*\*\*\*\*\*\*\*\*\*\*\*Execution ended\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Out[68]:   |  | **Accuracy** | **Model** | | --- | --- | --- | | **0** | 0.878160 | Activation:relu layers:3 nodes:400 batch:10000 | | **0** | 0.881378 | Activation:relu layers:3 nodes:600 batch:10000 | | **0** | 0.883639 | Activation:relu layers:3 nodes:800 batch:10000 | | **0** | 0.856108 | Activation:relu layers:3 nodes:400 batch:20000 | | **0** | 0.836265 | Activation:relu layers:3 nodes:600 batch:20000 | | **0** | 0.855081 | Activation:relu layers:3 nodes:800 batch:20000 | | **0** | 0.881786 | Activation:relu layers:5 nodes:400 batch:10000 | | **0** | 0.884042 | Activation:relu layers:5 nodes:600 batch:10000 | | **0** | 0.885010 | Activation:relu layers:5 nodes:800 batch:10000 |   In [69]:  p = test\_model(5, 400, 8, 'relu', 20000, X\_train, y\_train, X\_test, y\_test)  model\_clf\_stats = model\_clf\_stats.append(p)  p = test\_model(5, 600, 8, 'relu', 20000, X\_train, y\_train, X\_test, y\_test)  model\_clf\_stats = model\_clf\_stats.append(p)  p = test\_model(5, 800, 8, 'relu', 20000, X\_train, y\_train, X\_test, y\_test)  model\_clf\_stats = model\_clf\_stats.append(p)  model\_clf\_stats  \*\*\*\*\*\*\*\*\*\*\*\*\*\*Execution started for\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Activation: relu layers: 5 nodes: 400 batch: 20000  building node: 1  building node: 2  building node: 3  building node: 4  Epoch 1/500  252/252 [==============================] - 3s 11ms/step - loss: 1.3383 - accuracy: 0.7007 - val\_loss: 0.4294 - val\_accuracy: 0.8067  …  252/252 [==============================] - 3s 11ms/step - loss: 0.2647 - accuracy: 0.8809 - val\_loss: 0.2668 - val\_accuracy: 0.8797  Epoch 88/500  252/252 [==============================] - 3s 11ms/step - loss: 0.2644 - accuracy: 0.8811 - val\_loss: 0.2656 - val\_accuracy: 0.8803  Epoch 89/500  252/252 [==============================] - 3s 11ms/step - loss: 0.2645 - accuracy: 0.8810 - val\_loss: 0.2657 - val\_accuracy: 0.8802  Best score: 0.8810511827468872  Model: "sequential"  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Layer (type) Output Shape Param #  =================================================================  dense (Dense) (None, 400) 11600    dense\_1 (Dense) (None, 302) 121102    dense\_2 (Dense) (None, 204) 61812    dense\_3 (Dense) (None, 106) 21730    dense\_4 (Dense) (None, 1) 107    =================================================================  Total params: 216,351  Trainable params: 216,351  Non-trainable params: 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \*\*\*\*\*\*\*\*\*\*\*\*\*\*Execution ended\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*Execution started for\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Activation: relu layers: 5 nodes: 600 batch: 20000  building node: 1  building node: 2  building node: 3  building node: 4  Epoch 1/500  252/252 [==============================] - 5s 17ms/step - loss: 1.8562 - accuracy: 0.6664 - val\_loss: 0.4256 - val\_accuracy: 0.8077  …  252/252 [==============================] - 4s 16ms/step - loss: 0.2624 - accuracy: 0.8820 - val\_loss: 0.2646 - val\_accuracy: 0.8807  Epoch 89/500  252/252 [==============================] - 4s 16ms/step - loss: 0.2619 - accuracy: 0.8823 - val\_loss: 0.2640 - val\_accuracy: 0.8811  Best score: 0.8822767734527588  Model: "sequential"  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Layer (type) Output Shape Param #  =================================================================  dense (Dense) (None, 600) 17400    dense\_1 (Dense) (None, 452) 271652    dense\_2 (Dense) (None, 304) 137712    dense\_3 (Dense) (None, 156) 47580    dense\_4 (Dense) (None, 1) 157    =================================================================  Total params: 474,501  Trainable params: 474,501  Non-trainable params: 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \*\*\*\*\*\*\*\*\*\*\*\*\*\*Execution ended\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*Execution started for\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Activation: relu layers: 5 nodes: 800 batch: 20000  building node: 1  building node: 2  building node: 3  building node: 4  Epoch 1/500  252/252 [==============================] - 7s 25ms/step - loss: 1.9167 - accuracy: 0.6863 - val\_loss: 0.4180 - val\_accuracy: 0.8077  Epoch 2/500  252/252 [==============================] - 6s 24ms/step - loss: 0.4175 - accuracy: 0.8018 - val\_loss: 0.3867 - val\_accuracy: 0.8188  …  252/252 [==============================] - 6s 24ms/step - loss: 0.2791 - accuracy: 0.8726 - val\_loss: 0.2782 - val\_accuracy: 0.8732  Epoch 50/500  252/252 [==============================] - 6s 24ms/step - loss: 0.2768 - accuracy: 0.8740 - val\_loss: 0.2764 - val\_accuracy: 0.8740  Best score: 0.877587080001831  Model: "sequential"  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Layer (type) Output Shape Param #  =================================================================  dense (Dense) (None, 800) 23200    dense\_1 (Dense) (None, 602) 482202    dense\_2 (Dense) (None, 404) 243612    dense\_3 (Dense) (None, 206) 83430    dense\_4 (Dense) (None, 1) 207    =================================================================  Total params: 832,651  Trainable params: 832,651  Non-trainable params: 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \*\*\*\*\*\*\*\*\*\*\*\*\*\*Execution ended\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Out[69]:   |  | **Accuracy** | **Model** | | --- | --- | --- | | **0** | 0.878160 | Activation:relu layers:3 nodes:400 batch:10000 | | **0** | 0.881378 | Activation:relu layers:3 nodes:600 batch:10000 | | **0** | 0.883639 | Activation:relu layers:3 nodes:800 batch:10000 | | **0** | 0.856108 | Activation:relu layers:3 nodes:400 batch:20000 | | **0** | 0.836265 | Activation:relu layers:3 nodes:600 batch:20000 | | **0** | 0.855081 | Activation:relu layers:3 nodes:800 batch:20000 | | **0** | 0.881786 | Activation:relu layers:5 nodes:400 batch:10000 | | **0** | 0.884042 | Activation:relu layers:5 nodes:600 batch:10000 | | **0** | 0.885010 | Activation:relu layers:5 nodes:800 batch:10000 | | **0** | 0.881051 | Activation:relu layers:5 nodes:400 batch:20000 | | **0** | 0.882277 | Activation:relu layers:5 nodes:600 batch:20000 | | **0** | 0.877587 | Activation:relu layers:5 nodes:800 batch:20000 |   In [10]:  *#Refit best model to get reference*  safety = EarlyStopping(monitor='val\_loss', patience=5)  seed = 45 *#88.27*  m = createmodel(n\_layers=5, first\_layer\_nodes=600, last\_layer\_nodes=8,  activation\_func='relu', loss\_func=tf.keras.losses.BinaryCrossentropy()) *#tanh*  hist = m.fit(X\_train, y\_train, epochs=500, batch\_size=10000,  validation\_data=(X\_test, y\_test), callbacks=[safety], verbose=1) *# add validation left out here*  best\_score = max(hist.history['accuracy'])  print("Best score: ",best\_score)  building node: 1  building node: 2  building node: 3  building node: 4  Epoch 1/500  504/504 [==============================] - 7s 10ms/step - loss: 0.9710 - accuracy: 0.7246 - val\_loss: 0.3895 - val\_accuracy: 0.8203  Epoch 2/500  504/504 [==============================] - 5s 9ms/step - loss: 0.3894 - accuracy: 0.8184 - val\_loss: 0.3650 - val\_accuracy: 0.8266  …  504/504 [==============================] - 5s 9ms/step - loss: 0.2604 - accuracy: 0.8832 - val\_loss: 0.2620 - val\_accuracy: 0.8823  Epoch 54/500  504/504 [==============================] - 5s 9ms/step - loss: 0.2600 - accuracy: 0.8834 - val\_loss: 0.2623 - val\_accuracy: 0.8822  Best score: 0.8834400773048401  In [14]:  m.summary()  Model: "sequential"  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Layer (type) Output Shape Param #  =================================================================  dense (Dense) (None, 600) 17400    dense\_1 (Dense) (None, 452) 271652    dense\_2 (Dense) (None, 304) 137712    dense\_3 (Dense) (None, 156) 47580    dense\_4 (Dense) (None, 1) 157    =================================================================  Total params: 474,501  Trainable params: 474,501  Non-trainable params: 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  In [15]:  train\_loss = m.history.history['loss']  val\_loss = m.history.history['val\_loss']  \_=plt.plot(train\_loss, label='Train')  \_=plt.plot(val\_loss, label='Test')  \_=plt.xlabel("Epoch")  \_=plt.ylabel("Val Loss")  \_=plt.legend()  plt.show()    In [14]:  train\_acc = m.history.history['accuracy']  val\_acc = m.history.history['val\_accuracy']  \_=plt.plot(train\_acc, label='Train')  \_=plt.plot(val\_acc, label='Test')  \_=plt.xlabel("Epoch")  \_=plt.ylabel("Accuracy")  \_=plt.legend()  plt.show()    In [49]:  m.summary()  Model: "sequential"  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Layer (type) Output Shape Param #  =================================================================  dense (Dense) (None, 400) 11600    dense\_1 (Dense) (None, 204) 81804    dense\_2 (Dense) (None, 1) 205    =================================================================  Total params: 93,609  Trainable params: 93,609  Non-trainable params: 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  In [17]:  **from** **sklearn.metrics** **import** roc\_curve  y\_pred\_keras = m.predict(X\_hold).ravel()  fpr\_keras, tpr\_keras, thresholds\_keras = roc\_curve(y\_hold, y\_pred\_keras)  In [18]:  **from** **sklearn.metrics** **import** auc  auc\_keras = auc(fpr\_keras, tpr\_keras)  In [19]:  plt.figure(1)  plt.plot(fpr\_keras, tpr\_keras, label='Keras (area = **{:.3f}**)'.format(auc\_keras))  plt.xlabel('False positive rate')  plt.ylabel('True positive rate')  plt.title('ROC curve')  plt.legend(loc='best')  plt.show()  Out[19]:  <Figure size 576x396 with 0 Axes>  Out[19]:  [<matplotlib.lines.Line2D at 0x7f6f0fec1b50>]  Out[19]:  Text(0.5, 0, 'False positive rate')  Out[19]:  Text(0, 0.5, 'True positive rate')  Out[19]:  Text(0.5, 1.0, 'ROC curve')  Out[19]:  <matplotlib.legend.Legend at 0x7f6f0fe61550>    In [20]:  y\_pred\_keras[y\_pred\_keras <= 0.5] = 0.  y\_pred\_keras[y\_pred\_keras > 0.5] = 1.  In [21]:  **from** **sklearn.metrics** **import** confusion\_matrix  **import** **itertools**  cm = confusion\_matrix(y\_true=y\_hold, y\_pred=y\_pred\_keras)  **def** plot\_confusion\_matrix(cm, classes,  normalize=**False**,  title='Confusion matrix',  cmap=plt.cm.Blues):  *"""*  *This function prints and plots the confusion matrix.*  *Normalization can be applied by setting `normalize=True`.*  *"""*  plt.imshow(cm, interpolation='nearest', cmap=cmap)  plt.title(title)  plt.colorbar()  tick\_marks = np.arange(len(classes))  plt.xticks(tick\_marks, classes, rotation=45)  plt.yticks(tick\_marks, classes)  **if** normalize:  cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]  print("Normalized confusion matrix")  **else**:  print('Confusion matrix, without normalization')  print(cm)  thresh = cm.max() / 2.  **for** i, j **in** itertools.product(range(cm.shape[0]), range(cm.shape[1])):  plt.text(j, i, cm[i, j],  horizontalalignment="center",  color="white" **if** cm[i, j] > thresh **else** "black")  plt.tight\_layout()  plt.ylabel('True label')  plt.xlabel('Predicted label')  cm\_plot\_labels = ['Yes','No']  plot\_confusion\_matrix(cm=cm, classes=cm\_plot\_labels, title='Confusion Matrix')  Confusion matrix, without normalization  [[301564 48348]  [ 33471 316617]]    In [61]:  print(classification\_report(y\_hold, y\_pred\_keras, target\_names=['Yes','No']))  precision recall f1-score support  Yes 0.74 0.92 0.82 349912  No 0.89 0.67 0.77 350088  accuracy 0.80 700000  macro avg 0.81 0.80 0.79 700000  weighted avg 0.81 0.80 0.79 700000 |

# References

1. Data set info : https://archive.ics.uci.edu/ml/datasets/HEPMASS
2. Article on particle physics: <https://towardsdatascience.com/how-deep-learning-can-solve-problems-in-high-energy-physics-53ed3cf5e1c5>
3. Particel physics: <https://en.wikipedia.org/wiki/Particle_physics>