Case Study 6 Report:

Particle Prediction Classifier

Alexy Morris and Domicia Herring

November 14, 2022

Abstract

The team has been provided with a large complex dataset from a group of scientists to create a dense neural network and classifier to predict the existence of a new particle. The data is binary with 0 representing no detection and 1 representing detection. The goal is to provide a high level accuracy for the algorithm.

1. Introduction

1.1 Business Understanding

The objective for this study is to create a dense neural network classifier to predict the existence of a new particle. The scientific community is interested because the high energy produced when a new particle is made can only be observed by the features that it leaves behind due to it disappearing so quickly.

1.2 Data Meaning Type

To engineer an algorithm that predicts the existence of a new particle, scientists provided a large complex dataset from the Higgs Boson Research. The dataset comprises 6,999,999 and 29 features described in Figure 1.2a.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7000000 entries, 0 to 6999999
Data columns (total 29 columns):
 #   Column     Dtype
---  ------     -----
 0   detection  int64
 1   f0         float64
 2   f1         float64
 3   f2         float64
 4   f3         float64
 5   f4         float64
 6   f5         float64
 7   f6         float64
 8   f7         float64
 9   f8         float64
 10  f9         float64
 11  f10        float64
 12  f11        float64
 13  f12        float64
 14  f13        float64
 15  f14        float64
 16  f15        float64
 17  f16        float64
 18  f17        float64
 19  f18        float64
 20  f19        float64
 21  f20        float64
 22  f21        float64
 23  f22        float64
 24  f23        float64
 25  f24        float64
 26  f25        float64
 27  f26        float64
 28  mass       float64
dtypes: float64(28), int64(1)
```

Figure 1.2a- Feature Descriptions

2. Methods

2.1 Data Preprocessing

The dataset originally consisted of a single dataset with almost 7 million records. To ensure the data was ready for processing, the “# label” column was changed to “detection”. Next, the “detection” variables were transformed from 0.0 and 1.0 float64 to 0 and 1 int64. Then, the dataset was checked for null values and none were found. From the team’s observations, the dataset did not require much preprocessing prior to the exploratory data analysis.

2.2 Exploratory Data Analysis

Conducting an exploratory data analysis, the team saw that the new particles detected (1) versus the not detected (0) were fairly. Depicted in 2.2a and 2.2b, the data shows 3,500,879 new particles detected and 3,499,121 new particles not detected.

```
1    3500879
0    3499121
Name: detection, dtype: int64
```

Figure 2.2a- Detection Breakdown

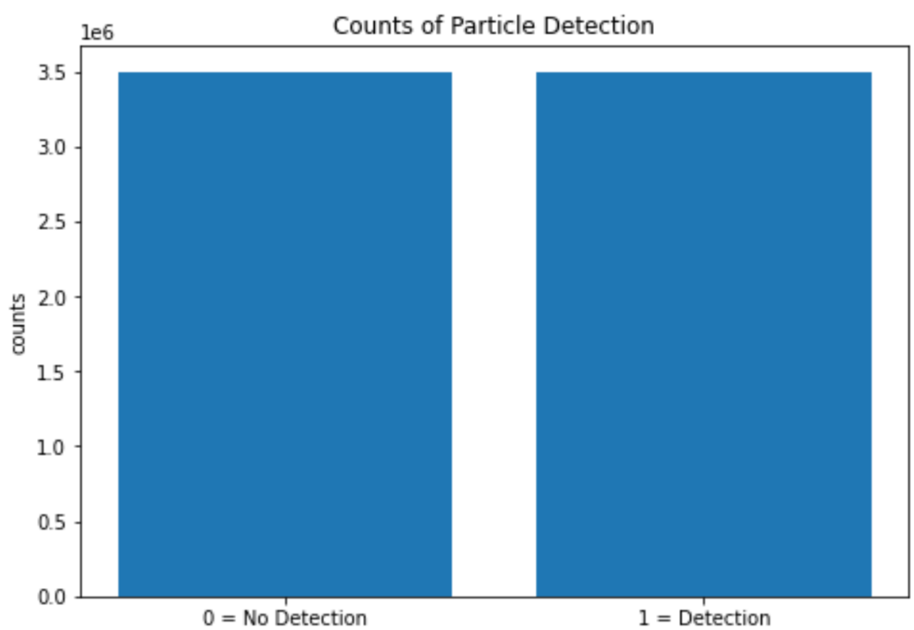

Figure 2.2b- Detection Bar Graph

Next, the team split our dataset into data/target train and test subsets using the train_test_split function from the sklearn model_selection package. Then, the data was rescaled to the default normalization using MinMaxScaler to scale variables into the range 0 to 1. After the aforementioned procedures were conducted, the model was ready to run.

2.3 Dense Neural Network Models

2.3.1 Model 1

For the first model, the team used a very basic selection of layers. The team defined the shape of the data, then proceeded to add the first dense layer. The activation relu was used as the team believed it would be well suited to the data and a good starting point for experimentation. For the same reason, the team decided to set the neurons to 50. Then, the team compiled the function. The team decided to use the optimizer “Adam” as the adaptive nature and low memory requirements are well suited for large data. Now addressing the fit, the team decided on a relatively low number of epochs in order to conserve resources and the same will apply to following models. The same logic applies to our batch size.

2.3.2 Model 2

For the second model, the same fit and compile methods have been applied as they have served well and the team have continued the study with layering the models. With the second model, the team tried various parameters before deciding that this combination was most optimal. The team has added a layer with 25 neurons and the activation sigmoid. The sigmoid function was chosen as it was deemed a good compliment to our previous layer according to a broad net of research. This did in fact bolster our results in a very distinctive manner and as such this model was chosen to continue our experimentation.

2.3.3 Model 3

The last model is the direct successor to model 2 and the team have once again decided to add a layer. For this layer, the team examined the results gained from the previous model and decided not to use a more involved activation and simply assigned the linear activation to our newest layer and to provide further variation we utilized the neuron scheme of 64, 32, and 16. The results from this were not ideal and as such the team decided to re-examine the activation structure. After a few rounds of slightly more empirical alterations, the team decided upon the final model of activation tanh, relu, and linear. The results gained were slightly above the given accuracy range, however the team believes this is the result of true methods and a data leak is unlikely. Therefore, this is also the team's best model.

3. Results

3.1 Dense Neural Network

3.1.1 Model 3

As stated in the previous section, the accuracy has gone slightly beyond the defined range but the team has decided that the results are trustworthy. Thus, the team is very satisfied with both our loss and accuracy metrics. The team used the cross validation method to judge the model in terms of new data and as shown in Figure 3.1.1.a and Figure 3.1.1.b below there are variations with the validation set but overall it does follow the training set well.

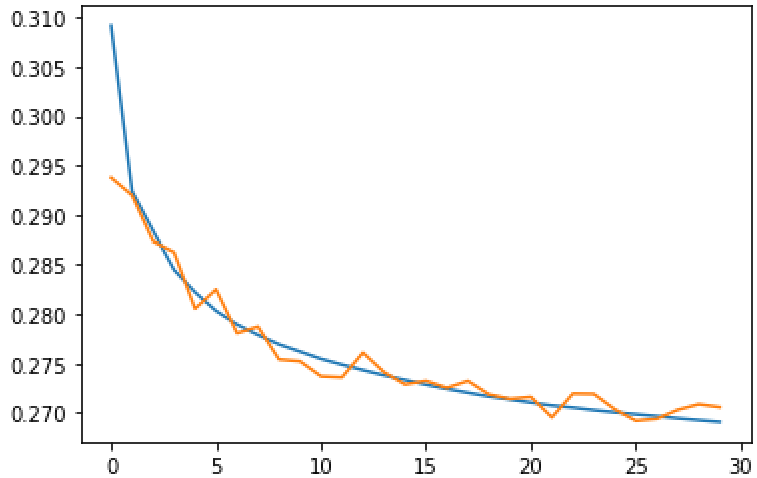

Figure 3.1.1.a-The loss attributed to our training (blue) and validation (orange) sets

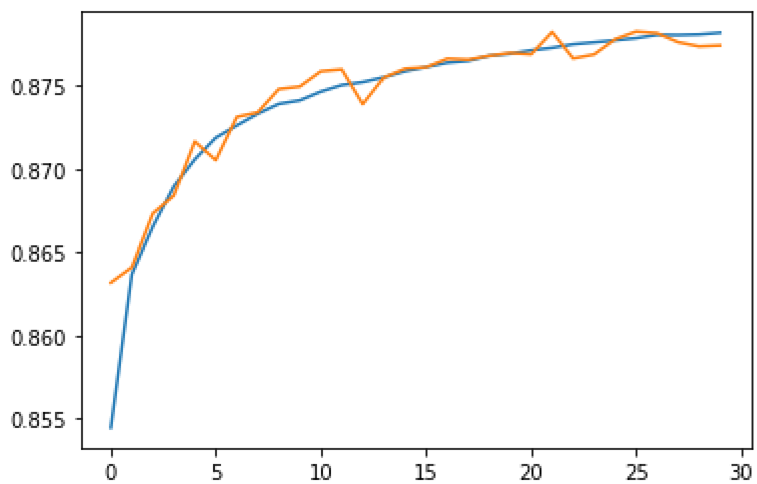

Figure 3.1.1.b-The accuracy attributed to our training (blue) and validation (orange) sets

4. Conclusion

The team was able to classify particles as new or not by utilizing the given records and processing them through a wide variety of neural network models. The last model reached a maximum accuracy of 87.83% and the loss reached a low of 26.92%. Thus, the team believes that as of now this is the best method for an automated particle classifier. At this time, the team does not believe that it can further improve the model with more time.

5. Citation

5.1. Baldi, P., P. Sadowski, and D. Whiteson. “Searching for Exotic Particles in High-energy Physics with Deep Learning.” Nature Communications 5 (July 2, 2014).

6. Code

In [ ]:

```
#import libraries
from __future__ import print_function
import pandas as pd
import numpy as np
import os
import tensorflow as tf
import cv2
from tensorflow.python import keras
from tensorflow.python.keras.utils.np_utils import to_categorical
from PIL import Image
import matplotlib.pyplot as plt
import datetime
import keras
from keras.models import Sequential
from tensorflow.keras import layers
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
```

In [ ]:

```
#Mount drive
from google.colab import drive
drive.mount('/content/drive/')
```

In [ ]:

```
#Unzip file and store in csv
import gzip


with gzip.open('/content/drive/MyDrive/Colab Notebooks/CS6/all_train.csv.gz', '
    csv_data = csv_file.read()
    with open('/content/drive/MyDrive/Colab Notebooks/CS6/all_train.csv', 'wt')
        out_file.write(csv_data)
```

In [ ]:

```
#Load up data and show.
 df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/CS6/all_train.csv')
 df.head()
```

In [ ]:

```
#Rename labels column
df2=df
df2.rename(columns = {'# label':'detection'}, inplace = True)
df2.head()
```

In [ ]:

```
#Display a summary of variables
df2.info()
```

In [ ]:

```
#Summary of Attributes in the dataframe
df2.describe()
```

In [ ]:

```
#Replace all '0.0' with '0' and '1.0' with '1' in the "# labels" column.
df2['detection'] = df2['detection'].map({0.0: 0, 1.0: 1})
df2.head()
```

In [ ]:
```
df2.info()
```

In [ ]:
```
#see if there are nulls
df2.isnull().sum()
```

In [ ]:
```
#Count values in label
df2['detection'].value_counts()
```

In [ ]:
```
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
labels = ["0 = No Detection", "1 = Detection"]
ax.bar(labels,df2["detection"].value_counts())
plt.ylabel("counts")
plt.title('Counts of Particle Detection')
plt.show()
```

In [ ]:
```
#Examine the mean of the Numerical Values by detection
df2.groupby(['detection']).mean()
```

In [ ]:
```
#Examine the mean of the numerical values by mass
df2.groupby('mass')['detection'].value_counts()
```

In [ ]:
```
import seaborn as sns
```

In [ ]:
```
#Create a Violinplot of Detection and Mass
import seaborn as sns
sns.set_palette("pastel")
sns.violinplot(x='detection',y='mass', data = df2)
plt.title('Detection by Mass')
plt.show()
```

In [ ]:
```
#Potential attribute with all factors
sns.pairplot(df2, kind="scatter", hue = "detection", markers = ["o", "s"], pale
plt.show()
```

In [ ]:
```
now = datetime.datetime.now

batch_size = 128
num_classes = 10
epochs = 30
# number of convolutional filters to use
filters = 32
```

```
# size of pooling area for max pooling
pool_size = 2
# convolution kernel size
kernel_size = 3
```

In [ ]:

```
x = df2.loc[:,df2.columns !='detection']
y = df2['detection']
```

In [ ]:

```
# Range of 0,1 is important for well trained neural networks
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0, 1))
scaled_train = scaler.fit_transform(x)

# Print out the adjustment that the scaler applied to the total_earnings column
print("Note: median values were scaled by multiplying by {:.10f} and adding {:.
multiplied_by = scaler.scale_[27]
added = scaler.min_[27]

scaled_train_df = pd.DataFrame(scaled_train, columns=x.columns.values)
```

In [ ]:

```
%matplotlib inline

for i in scaled_train_df:
    scaled_train_df[i].hist()
    plt.title(i)
    plt.show()
```

In [ ]:

```
# Split into test/train
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(scaled_train_df, y, test_si
```

In [ ]:

```
# Create the model and build layers
model = tf.keras.Sequential()
model.add(tf.keras.Input(shape=(28,)))
model.add(layers.Dense(50,activation='relu'))
```

In [ ]:

```
model.compile(optimizer='Adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=Tr
              metrics=['mean_squared_error','accuracy'])
```

In [ ]:

```
model.fit(X_train, y_train, epochs=30, validation_data=(X_test,y_test), batch_s
```

In [ ]:

```
#Plot loss
train_loss = model.history.history['loss']
val_loss = model.history.history['val_loss']
plt.plot(train_loss)
plt.plot(val_loss)
plt.show()
```

In [ ]:
```
#Plot Accuacy
train_acc = model.history.history['accuracy']
val_acc = model.history.history['val_accuracy']
plt.plot(train_acc)
plt.plot(val_acc)
plt.show()
```

In [ ]:
```
# Create the model and build layers
model = tf.keras.Sequential()
model.add(tf.keras.Input(shape=(28,)))
model.add(layers.Dense(50,activation='relu'))
model.add(layers.Dense(25, activation='sigmoid'))
```

In [ ]:
```
model.compile(optimizer='Adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=Tr
              metrics=['mean_squared_error','accuracy'])
```

In [ ]:
```
model.fit(X_train, y_train, epochs=30, validation_data=(X_test,y_test), batch_s
```

In [ ]:
```
#Plot Loss
train_loss = model.history.history['loss']
val_loss = model.history.history['val_loss']
plt.plot(train_loss)
plt.plot(val_loss)
plt.show()
```

In [ ]:
```
#Plot Accuacy
train_acc = model.history.history['accuracy']
val_acc = model.history.history['val_accuracy']
plt.plot(train_acc)
plt.plot(val_acc)
plt.show()
```

In [ ]:
```
# Create the model and build layers
model = tf.keras.Sequential()
model.add(tf.keras.Input(shape=(28,)))
model.add(layers.Dense(64,activation='tanh'))
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(16, activation='linear'))
```

In [ ]:
```
model.compile(optimizer='Adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=Tr
              metrics=['mean_squared_error','accuracy'])
```

In [ ]:
```
model.fit(X_train, y_train, epochs=30, validation_data=(X_test,y_test), batch_s
```

In [ ]:

```
#Plot Loss
train_loss = model.history.history['loss']
val_loss = model.history.history['val_loss']
plt.plot(train_loss)
plt.plot(val_loss)
plt.show()
```

In [ ]:

```
#Plot Accuacy
train_acc = model.history.history['accuracy']
val_acc = model.history.history['val_accuracy']
plt.plot(train_acc)
plt.plot(val_acc)
plt.show()
```