Udacity MLND Capstone Project
Manish Thakrani

I. Definition

Project Overview

The problem arises in the domain of particle physics. Physicists collide particles at high energy and observe the reactions that follow. These observations have led to deeper understanding of matter, and also to the discovery of previously unknown particles that exist only at very high energy.

In order to be certain of the fact, that new particles resulted from a given collision, the scientists collect a set of measurements from the collision and use these as the feature set. These features are then fed to a set of known physics functions to augment the feature space. The total set of features is then analysed by a pre trained model to see whether the collision resulted in a new particle of not. Since training data is very expensive to generate, the models are pre-trained on simulated data.

Datasets and Input

The dataset for model training and validation is provided at
https://archive.ics.uci.edu/ml/datasets/HIGGS

The data has been produced using Monte Carlo simulations. The first 21 features (columns 2-22) are kinematic properties measured by the particle detectors in the accelerator. The last seven features are functions of the first 21 features; these are high-level features derived by physicists to help discriminate between the two classes.

Problem Statement

The problem statement is to find a model that accurately classifies particle collisions that result in the generation of a new particle versus those that do not. We take a set of labelled particle collision data and train a classifier on this data for binary classification (new particle discovered [1] vs not discovered).

The original dataset, contains a set of low level features (direct measurements from particle colliders) and a set of high level features (features derived from the low level features through physical modelling).

In addition to binary classification of the dataset, the project also aims to compare performance of the model when using only low level features versus using all features. The expectation is that a good model only trained on the low level features, will have comparable performance to the one trained on all features (since it will be able to auto learn the high level feature equations

[1]

during training). If this is found to be true, it will alleviate the need for the physicists to do additional modelling before machine learning.

The problem is described in the paper '[Searching for Exotic Particles in High-Energy Physics with Deep Learning]'

Metrics

I have used AUC (area under a [ROC curve]) to evaluate the performance of my models. Since this is the same metric used in the paper, it will make it easier to compare the performance of ensembles to the models described in the paper.

II. Analysis

Data Exploration

The dataset is tabular with 29 columns and 11 million rows. All cell values are numerical.

The first column is the classification of the measurement

- 0 indicates that the measurements do not indicate the discovery of a new particle
- 1 indicates that the measurements indicate the discovery of a new particle

The nest 21 features (columns 1-21) are kinematic properties measured by the particle detectors in the accelerator.

The last seven features (columns 22-27) are functions of the first 21 features; these are high-level features derived by physicists to help discriminate between the two classes.

A sample of the dataset as displayed in the accompanying IPython notebook:

```
data.head()
```

	0	1	2	3	4	5	6	7	8	9	...	19	20	21	22	23	
0	1.0	0.869293	-0.635082	0.225690	0.327470	-0.689993	0.754202	-0.248573	-1.092064	0.000000	...	-0.010455	-0.045767	3.101961	1.353760	0.979563	0
1	1.0	0.907542	0.329147	0.359412	1.497970	-0.313010	1.095531	-0.557525	-1.588230	2.173076	...	-1.138930	-0.000819	0.000000	0.302220	0.833048	0
2	1.0	0.798835	1.470639	-1.635975	0.453773	0.425629	1.104875	1.282322	1.381664	0.000000	...	1.128848	0.900461	0.000000	0.909753	1.108330	0
3	0.0	1.344385	-0.876626	0.935913	1.992050	0.882454	1.786066	-1.646778	-0.942383	0.000000	...	-0.678379	-1.360356	0.000000	0.946652	1.028704	0
4	1.0	1.105009	0.321356	1.522401	0.882808	-1.205349	0.681466	-1.070464	-0.921871	0.000000	...	-0.373566	0.113041	0.000000	0.755856	1.361057	0

5 rows × 29 columns

The accompanying IPython notebook contains the following cell containing a summary statistic table for all the features:

```
In [7]: data.iloc[:, 1:].describe()
```

Out[7]:

	1	2	3	4	5	6	7	8	9	10	...
count	1.100000e+07	1.100000e+07	1.100000e+07	1.100000e+07	1.100000e+07	1.100000e+07	1.100000e+07	1.100000e+07	1.100000e+07	1.100000e+07	...
mean	9.914658e-01	-8.297618e-06	-1.327225e-05	9.985364e-01	2.613459e-05	9.909152e-01	-2.027520e-05	7.716199e-06	9.999687e-01	9.927294e-01	...
std	5.653777e-01	1.008827e+00	1.006346e+00	6.000185e-01	1.006326e+00	4.749747e-01	1.009303e+00	1.005901e+00	1.027808e+00	4.999939e-01	...
min	2.746966e-01	-2.434976e+00	-1.742508e+00	2.370088e-04	-1.743944e+00	1.375024e-01	-2.969725e+00	-1.741237e+00	0.000000e+00	1.889811e-01	...
25%	5.907533e-01	-7.383225e-01	-8.719308e-01	5.768156e-01	-8.712081e-01	6.789927e-01	-6.872450e-01	-8.680962e-01	0.000000e+00	6.564608e-01	...
50%	8.533714e-01	-5.415563e-05	-2.410638e-04	8.916277e-01	2.125454e-04	8.948193e-01	-2.543566e-05	5.813991e-05	1.086538e+00	8.901377e-01	...
75%	1.236226e+00	7.382142e-01	8.709940e-01	1.293056e+00	8.714708e-01	1.170740e+00	6.871941e-01	8.683126e-01	2.173076e+00	1.201875e+00	...
max	1.209891e+01	2.434868e+00	1.743236e+00	1.539682e+01	1.743257e+00	9.940391e+00	2.969674e+00	1.741454e+00	2.173076e+00	1.164708e+01	...

8 rows × 28 columns

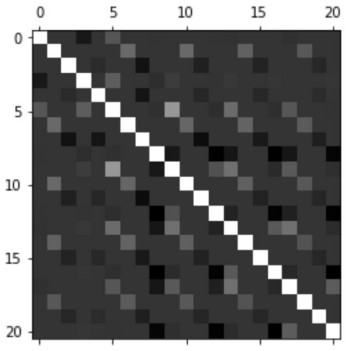
It was verified that the input dataset contains no missing values

Exploratory Visualization

I plotted the correlation matrix, to see whether there are any correlated features, I can eliminate, to speed up the following analysis. The correlation found is displayed below:

```
plt.matshow(data.iloc[:, 1:22].corr(), cmap=plt.cm.gray)
```

```
<matplotlib.image.AxesImage at 0x7f8ce5201320>
```

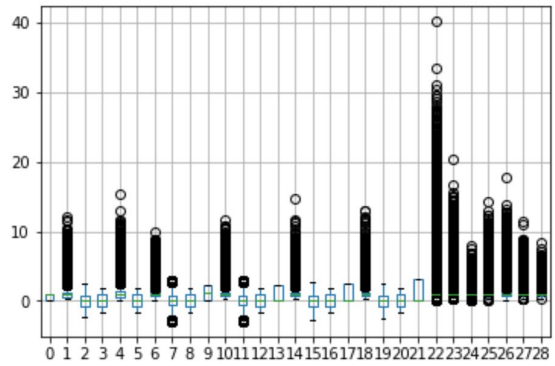


White in the above matrix, indicates maximum positive correlation.
Since there not a lot of non diagonal whites in the above images, all the features were retained for further analysis.

I tried to analyze the presence of outliers, by plotting a box plot of the data.
A boxplot on the original dataset is as shown below:

```
data.boxplot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8ce51b3278>
```



The black dots in the above figure, point to potential outliers in the data.

Algorithms and Techniques

I want to propose a solution that uses boosted trees (XGBoost), to construct a classifier that can discriminate between collisions that result in the generation of new particles versus those that do not. The expectation is that since ensembles can learn hidden non linear relationships in the data, we might be able to get a good performing classifier without having to augment the data with known nonlinear data mappings.

Since the original paper contains classification results obtained using a Decision Tree classifier and varying depth deep neural networks, these can be used to compare the quality of the results obtained using the boosted trees approach.

Benchmark

The paper contains results of running a Decision Tree classifier and varying depth deep neural networks for classifying the data. These results are used as a benchmark to evaluate the results obtained in this capstone. AUC scores in the original paper are as shown:

	AUC		
Technique	Low-level	High-level	Complete
BDT	0.73 (0.01)	0.78 (0.01)	0.81 (0.01)
NN	0.733 (0.007)	0.777 (0.001)	0.816 (0.004)
DN	0.880 (0.001)	0.800 (< 0.001)	0.885 (0.002)

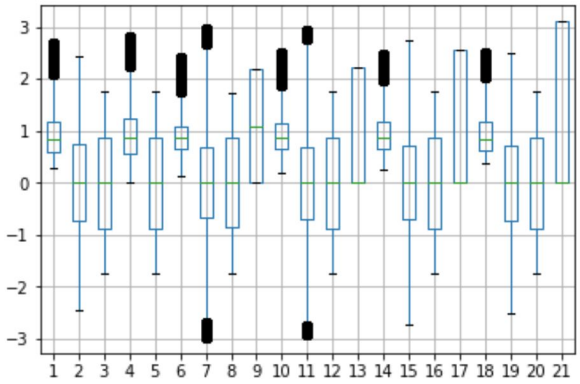
III. Methodology

Data Preprocessing

Even though ensembles are fairly resistant to outliers, I thought it would be better to trim the outliers from the dataset in order to improve on training and test times. A boxplot of the data, after removing all values > 3 standard deviations away from the mean, is shown below:

```
data.iloc[:, 1:22][(np.abs(stats.zscore(data)) < 3).all(axis=1)].boxplot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8ce0e677f0>
```



Implementation

I implemented the XGBoost classifier with default parameters on the entire dataset, only on low level features and then only on high level features and compared the AUC scores. The default XGBoost classifier params and the AUC score table is shown below:

	AUC Scores		
	All features	Low level features	High level features
XGBoost Classifier learning_rate = 0.3 max_depth = 6 subsample = 1 colsample_bytree = 1 reg_lambda = 1 n_estimators = 100	0.7052909173964135	0.6069491767076749	0.6904762580156374

Refinement

I used GridSearchCV to tune model parameters. The tuning parameters and their value ranges are as follows:

- 'learning_rate' : [0.05, 0.1, 0.2, 0.3],
- 'max_depth': [3, 5, 6, 10, 20],
- 'subsample': [0.5, 1],
- 'colsample_bytree': [0.5, 1],
- 'reg_lambda': [0,1],
- 'n_estimators': [100, 500, 1000]

The grid search based model was trained only on the low level features, since the expectation was that ensembles should alleviate the need to physically model high level features.

Since training was taking a long time

- The training and test evaluation was done on a GPU enabled machine
- Training and test samples were reduced in size
- Used stratified sampling to ensure original distribution of data

The most optimal parameters and the associated AUC score is shown below:
(The AUC score shown is the one reported by grid search cv)

	AUC Scores
	Low level features
XGBoost Classifier learning_rate = 0.1 max_depth = 20 subsample = 0.5 colsample_bytree = 0.5 reg_lambda = 1 n_estimators = 1000	0.7503220789072178

IV. Results

Model Evaluation and Validation

The final model was tested on both on

- a test sample, that the model had not seen before
- The complete dataset

The AUC scores were

AUC	
Test data	All data
0.6884253689615669	0.7231039563087688

The results do not show huge variation which suggests that the model is relatively stable.

Justification

These final results **are not** stronger than the results obtained in the original paper. The authors got better results by using deep neural networks on the same feature set.

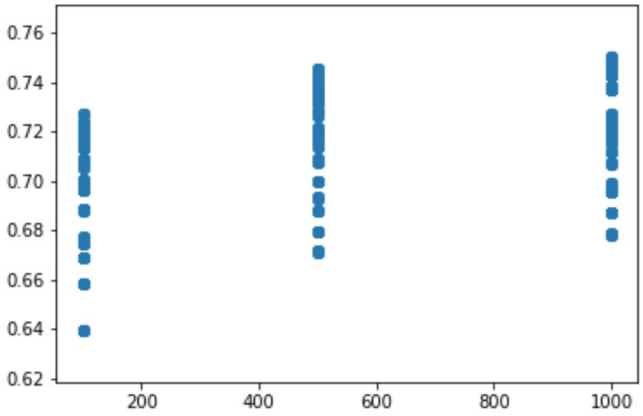
My final model was highly complex (max tree depth of 20 in a feature space of 21) and took a long time to train. Thus it did not even have the time or the interpretability advantage when compared to the deep neural network approach proposed by the authors.

Overall, the attributes if the data and the results from this exercise suggest that the inherent relationship between the features is highly nonlinear and a deep neural network might be the best way to model this relationship

V. Conclusion

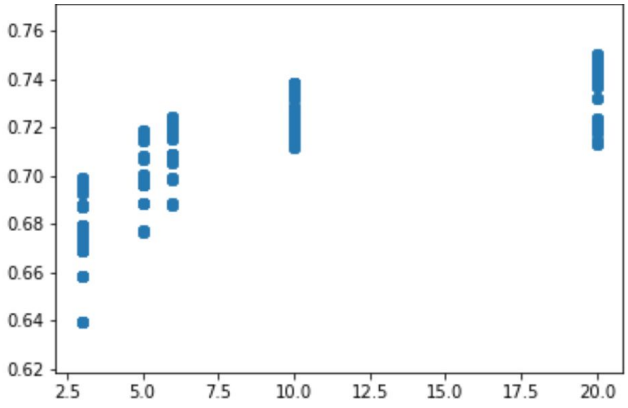
Free-Form Visualization

I plotted the num_estimators vs the mean_test_score scatter plot



On the num_estimators, I saw that I could have done equally well with much less estimators. (Use ~500 instead of the suggested optimal value of 1000)
I would have lost a little bit accuracy but gained a lot in terms of training time.

And the tree_depth vs the mean_test_score scatter plot



With tree depth, I have the same observation. A tree_depth of ~10 can give approximately optimal results with corresponding change in other params. This can reduce training time a lot with only a slight decrease in overall accuracy.

Reducing complexity in these hyper params also might protect us from overfitting our dataset.

Reflection

The overall process for this project was straightforward - train, test, tune repeat.
Since the dataset was purely numerical, I was not required to transform data types.
The dataset also did not have any class imbalance.
My proposal was to use ensembles to solve this problem, and since ensembles in general are not very sensitive to outliers, I did not have to apply any complicated pre-processing steps to this dataset before applying machine learning techniques to it.

From the machine learning perspective, I could not beat the accuracy of the neural network described in the original paper. This seems to suggest that the underlying function is more complicated that what my ensemble could model.

Improvement

Training of boosted trees on large datasets takes considerable amount of time. Running a grid search on XGBoost with my param set, took the training ~20 hrs to complete on a GPU enabled machine.

XGBoost supports running on distributed platforms like Spark, Dataflow etc
One can try running this implementation on a distributed system to reduce training times.

The author also suggests the possible use of reinforcement learning algorithms on this dataset in the future. One could try and prototype a RL solution this problem and compare accuracy.