Hunting for Pulsars

Joe Davies

Useful Links

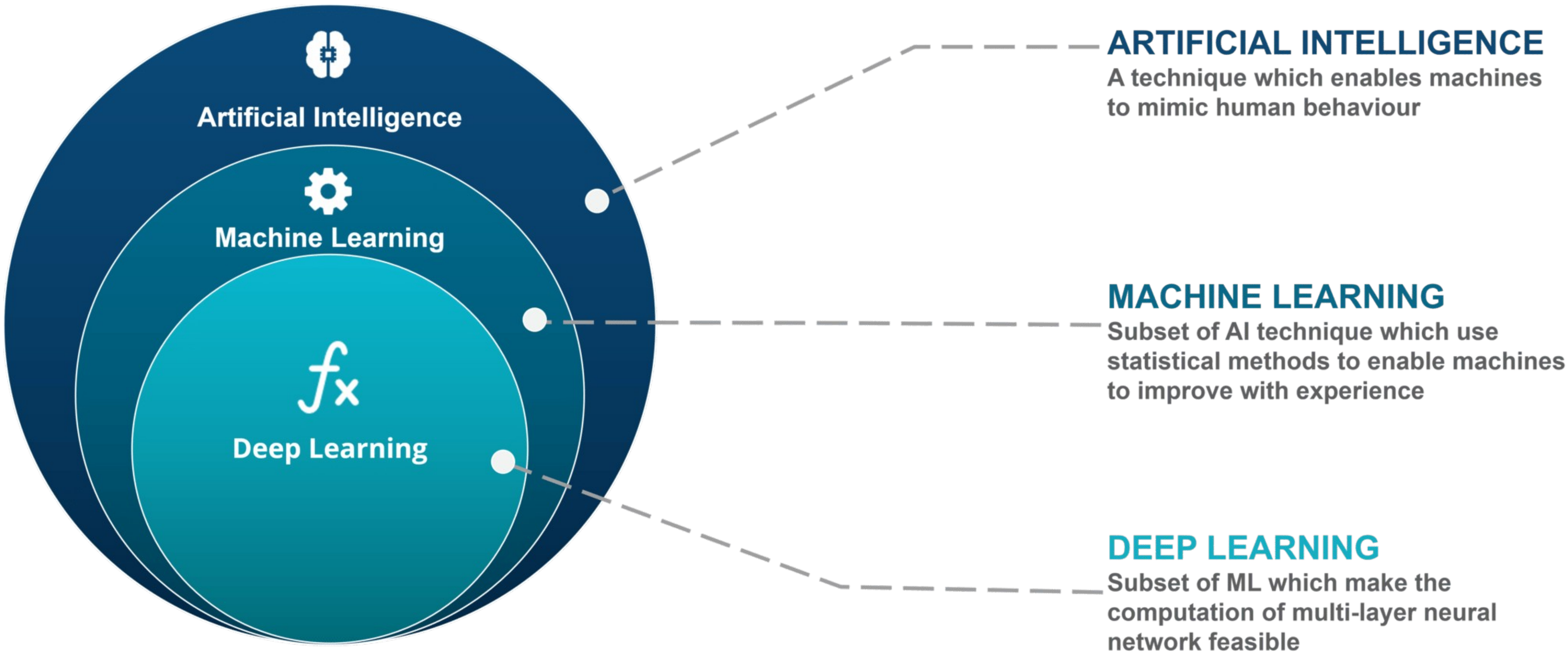
- **Particle physics application:**
https://ilmonteux.github.io/2018/10/15/jet-tagging-cnn.html
- **Kaggle page for the pulsar dataset:**
https://www.kaggle.com/shivam1901/pulsar-star
- **Github repo with all the code from these tutorials:**
https://github.com/adrianbevan/IntroToML/tree/master/Pulsars
- **https://learn.datacamp.com/courses/introduction-to-deep-learning-with-keras**
- **Misc useful links for more information on machine learning:**
  - **https://towardsdatascience.com/**
  - **https://www.coursera.org/learn/machine-learning**
  - **https://elitedatascience.com/learn-machine-learning**
- **j.m.m.davies@qmul.ac.uk**

Introductions!

The Dark Machines Anomaly Score Challenge: Benchmark Data and Model Independent Event Classification for the Large Hadron Collider

T. Aarrestad[CERN] M. van Beekveld[Ox] M. Bona[QMUL] A. Boveia[OSU] S. Caron[HEF, Nikhef] J. Davies[QMUL] A. De Simone[SISSA, INFN] C. Doglioni[Lund] J.M. Duarte[UCSD] A. Farbin[UnivArlington] H. Gupta[GSoC] L. Hendriks[HEF, Nikhef] L. Heinrich[CERN] J. Howarth[Glasgow] P. Jawahar[WPI, CERN] A. Jueid[UnivKonkuk] J. Lastow[Lund] A. Leinweber[UnivAdelaide] J. Mamuzic[IFIC] E. Merényi[UnivRice] A. Morandini[RWTH] P. Moskvitina[HEF, Nikhef] C. Nellist[HEF, Nikhef] J. Ngadiuba[FNAL, Caltech] B. Ostdiek[Harvard, AIFI] M. Pierini[CERN] B. Ravina[Glasgow] R. Ruiz de Austri[IFIC] S. Sekmen[KNU] M. Touranakou[NKUA, CERN] M. Vaškevičiūte[Glasgow] R. Vilalta[UnivHouston] J.-R. Vlimant[Caltech] R. Verheyen[UCL] M. White[UnivAdelaide] E. Wulff[Lund] E. Wallin[Lund] K.A. Wozniak[UniVie, CERN] Z. Zhang[HEF, Nikhef]

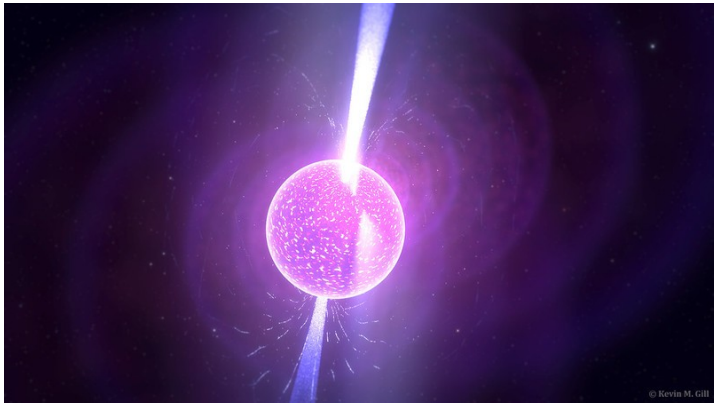
Machine Learning vs Artificial Intelligence



https://www.edureka.co/blog/ai-vs-machine-learning-vs-deep-learning/

The Data

- **Data is based on a Kaggle set investigating pulsars**
- **Pulsars are highly magnetized neutron stars that emit radiation from their magnetic poles**
- **Data contains 8 columns including metrics like kurtosis and dispersion of radiation measures**
- **The data also contains a target class: 0 or 1 depending on not-a-pulsar or pulsar**



pulsar_stars.csv (1.67 MB) 9 of 9 columns

	# Mean of the integra	# Standard deviation	# Excess kurtosis of t	# Skewness of the int	# Mean of the DM-SN	# Standard deviation	# Excess kurtosis of t	# Skewness of the DM	# target_class
	1. Mean of the integrated profile.	1. Standard deviation of the integrated profile.	1. Excess kurtosis of the integrated profile.	1. Skewness of the integrated profile.	1. Mean of the DM-SNR curve.	1. Standard deviation of the DM-SNR curve.	1. Excess kurtosis of the DM-SNR curve.	1. Skewness of the DM-SNR curve.	9.class of pulsar star. 1 for pulsar star,0 for not a star
	5.81 – 193	24.8 – 98.8	-1.88 – 8.07	-1.79 – 68.1	0.21 – 223	7.37 – 111	-3.14 – 34.5	-1.98 – 1.19k	0
1	140.5625	55.68378214	-0.234571412	-0.699648398	3.199832776	19.11042633	7.975531794	74.24222492	0
2	102.5078125	58.88243001	0.465318154	-0.515087909	1.677257525	14.86014572	10.57648674	127.3935796	0
3	103.015625	39.34164944	0.323328365	1.051164429	3.121237458	21.74466875	7.735822015	63.17190911	0
4	136.75	57.17844874	-0.068414638	-0.636238369	3.642976589	20.9592803	6.89649891	53.59366067	0
5	88.7265625	40.67222541	0.600866079	1.123491692	1.178929766	11.4687196	14.26957284	252.5673058	0
6	93.5703125	46.69811352	0.53190485	0.416721117	1.636287625	14.54507425	10.6217484	131.3940043	0
7	119.484375	48.76505927	0.03146022	-0.112167573	0.99916388	9.279612239	19.20623018	479.7565669	0
8	130.3828125	39.84405561	-0.158322759	0.389540448	1.220735786	14.37894124	13.53945602	198.2364565	0
9	107.25	52.62707834	0.452688025	0.170347382	2.331939799	14.48685311	9.001004441	107.9725056	0
10	107.2578125	39.49648839	0.465881961	1.162877124	4.079431438	24.98041798	7.397079948	57.78473789	0
11	142.078125	45.28807262	-0.320328426	0.283952506	5.376254181	29.00989748	6.076265849	37.83139335	0
12	133.2578125	44.05824378	-0.081059862	0.115361506	1.632107023	12.00780568	11.97206663	195.5434476	0
13	134.9609375	49.55432662	-0.135303833	-0.080469602	10.69648829	41.34204361	3.893934139	14.13120625	0
14	117.9453125	45.50657724	0.325437564	0.661459458	2.836120401	23.11834971	8.943211912	82.47559187	0
15	138.1796875	51.5244835	-0.031852329	0.046797173	6.330267559	31.57634673	5.155939859	26.14331017	0
16	114.3671875	51.94571552	-0.094498904	-0.287984087	2.738294314	17.19189079	9.050612454	96.61190318	0
17	109.640625	49.01765217	0.13763583	-0.256699775	1.508361204	12.07290134	13.36792556	223.4384192	0
18	100.8515625	51.74352161	0.393836792	-0.011240741	2.841137124	21.63577754	8.302241891	71.58436903	0
19	136.09375	51.69100464	-0.045908926	-0.271816393	9.342809365	38.09639955	4.345438138	18.67364854	0
20	99.3671875	41.57220208	1.547196967	4.154106043	27.55518395	61.71901588	2.20880796	3.662680136	1
21	100.890625	51.89039446	0.627486528	-0.026497802	3.883779264	23.04526673	6.953167635	52.27944038	0
22	105.4453125	41.13996851	0.142653801	0.320419676	3.551839465	20.75501684	7.739552295	68.51977061	0
23	95.8671875	42.05992212	0.326386917	0.803501794	1.83277592	12.24896949	11.249331	177.2307712	0
24	117.3671875	53.90861351	0.257953441	-0.405049077	6.018394649	24.76612335	4.807783224	25.52261561	0

What are we doing?

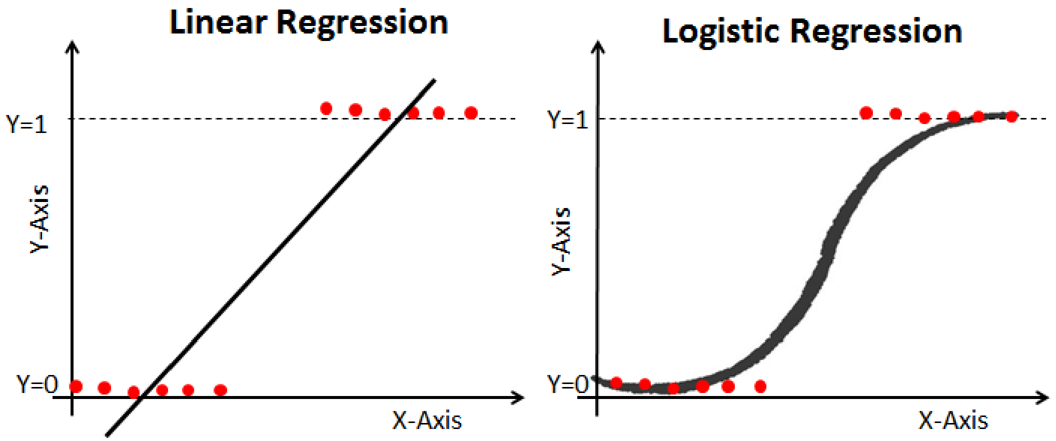
- Create 3 machine learning algorithms that identify pulsars from the data
- Understand how to use specific python modules to do this
- Get an idea of when to use and not use each algorithm
- sklearn, scipy, pandas, matplotlib, numpy, keras, tensorflow

Logistic Regression

- **Statistical model that uses a logistic function to (usually) model a binary variable (0 or 1)**
- **An example of this function is the sigmoid:**

$$f(x) = \frac{1}{1 + e^{-x}}$$

- **Uses gradient descent for the meat of the regression**
- **An issue with this approach is that extreme cases are presumed to become progressively rarer at a specific rate**
  - Not very good if one doesn't have much data



Logistic Regression

Example file: pulsar_logistic_sklearn.py

Data Preperation

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from scikitplot.estimators import plot_learning_curve
df=pd.read_csv('~/Documents/ML/pulsars/pulsar_stars.csv')
```

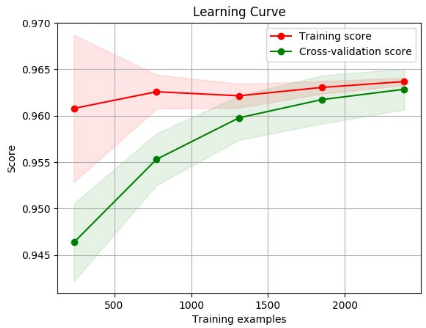
```
x_data=df.drop(columns='target_class')
x=(x_data-np.min(x_data))/(np.max(x_data)-np.min(x_data)) #scaling
y=df.target_class.values

compare_score=[]

#training and testing split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=123)
```

Model Creation

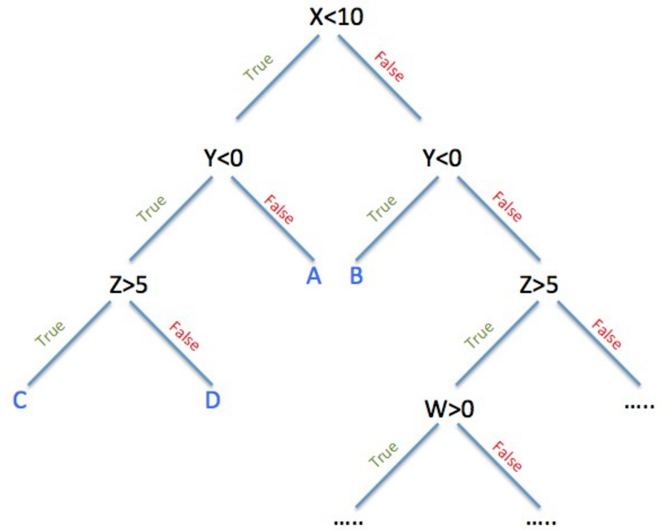
```
lr=LogisticRegression()
lr.fit(x_train, y_train)
```

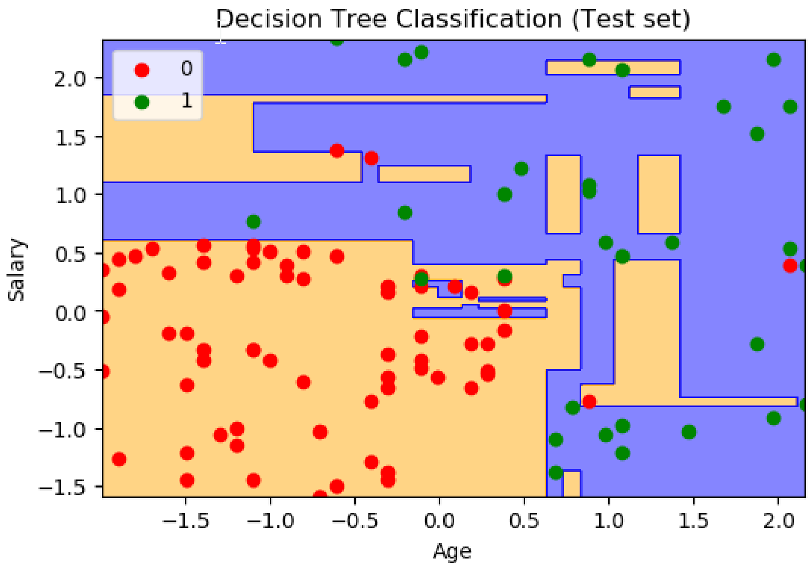
Visualization

```
lr_score=lr.score(x_test, y_test) * 100
compare_score.append(lr_score)

print('Test accuracy: {}%'.format(lr_score))

plot_learning_curve(lr, x_test, y_test)
plt.show()
```



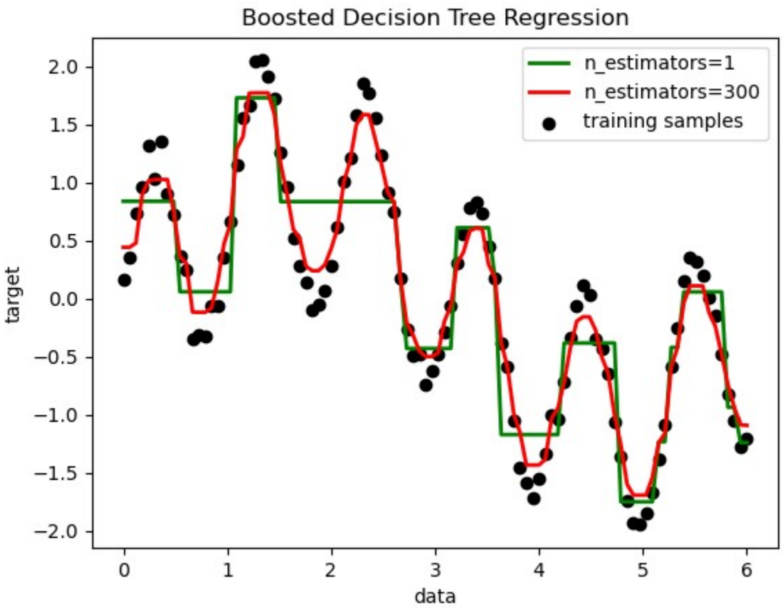
Decision Trees

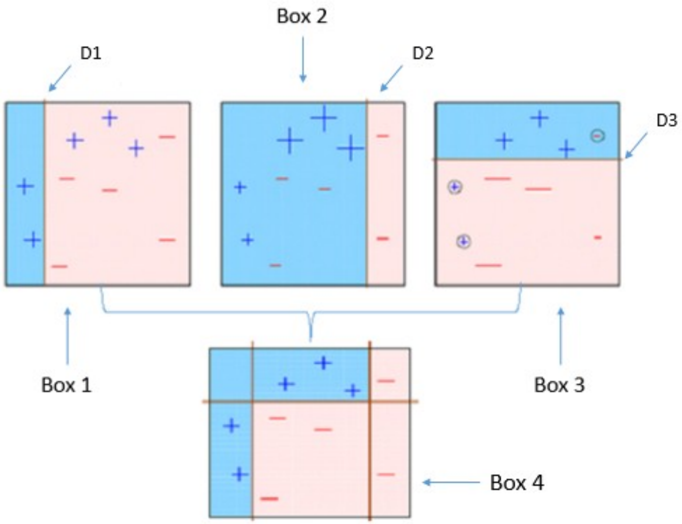




Advantages	Disadvantages
Simple to understand and interpret	Not very robust
Useful for classification and regression	Make locally optimal decisions (info gain max, entropy min)
Requires little data preparation and can handle large data sets	Prone to over-fitting

Boosted Decision Tree





- Boosting involves running multiple trees one after the other and minimizing error
- Each tree output, h(x), is given a weight, w, based on accuracy: $\hat{y}(x) = \sum_t w_t h_t(x)$
- Each data sample is given a weight based on misclassification
- We minimize the objective function: $O(x) = \sum_i l(\hat{y}_i, y_i) + \sum_t \Omega(f_t)$
- First term is loss function, second is regularization (penalizes complexity of the t-th tree)

https://towardsdatascience.com/understanding-adaboost-2f94f22d5bfe
https://indico.fnal.gov/event/15356/contributions/31377/attachments/19671/24560/DecisionTrees.pdf
https://scikit-learn.org/stable/auto_examples/ensemble/plot_adaboost_regression.html

Boosted Decision Tree

Example file: pulsar_bdt.py

Data Preperation

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
#Vizualization
import matplotlib.pyplot as plt
from scikitplot.estimators import plot_learning_curve

data=pd.read_csv('~/Documents/ML/pulsars/pulsar_stars.csv')
```

```
features=data.columns[:-1]
X=data[features]
#output:
y=data.target_class

#Now we need to split the data using train_test_split
X_train, X_test, y_train, y_test=train_test_split(X, y,
                                        test_size=0.2, random_state=42)
```

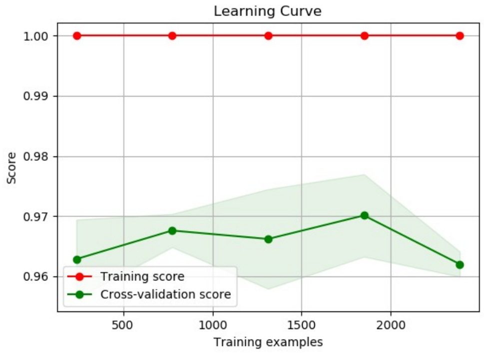
Model Creation

```
classifier=AdaBoostClassifier(DecisionTreeClassifier())
```

```
classifier=classifier.fit(X_train, y_train)
#Predcting the response for the test dataset
y_pred=classifier.predict(X_test)
```

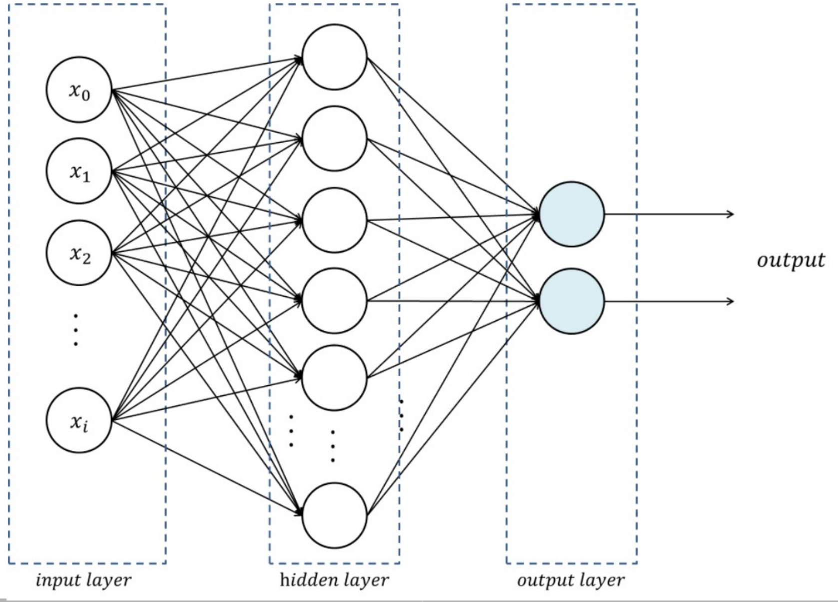
Visualization

```
score=round(metrics.accuracy_score(y_test, y_pred)*100, 2)
print('Accuracy = {}%'.format(score))

plot_learning_curve(classifier, X_test, y_test)
plt.show()
```



Artificial Neural Networks

$$y = f\left(\sum_{i=1}^{N} w_i x_i + \theta\right)$$
$$= f(w^T x + \theta)$$



$$w_{r+1} = w_r - \gamma \sum_{i=1}^{N} \frac{\partial E_i}{\partial w}$$

$$\theta_{r+1} = \theta_r - \gamma \sum_{i=1}^{N} \frac{\partial E_i}{\partial \theta}$$

Advantages	Disadvantages
Among the most accurate of modelling approaches	Computationally intensive
Useful for classification and regression	Easy to over- or under-training data
Makes few assumptions about relationships in the data	Results in a complex black box model

Artificial Neural Network

Example file: pulsar_ann.py

Data Preperation

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from keras import Sequential
from keras.layers import Dense
from sklearn.metrics import confusion_matrix
from scikitplot.estimators import plot_learning_curve
df=pd.read_csv('~/Documents/ML/pulsars/pulsar_stars.csv')
```

```
x_data=df.drop(columns='target_class')
X=StandardScaler().fit_transform(x_data)
y=df.target_class.values

x_train, x_test, y_train, y_test=train_test_split(X, y,
                                        test_size=0.2, random_state=42)
```

Model Creation

```
classifier=Sequential()

#first hidden layer
#we have 8 input features, 1 output and the kernel_initializer uses a normal distribution to
#function
classifier.add(Dense(8, activation='relu', kernel_initializer='random_normal', input_dim=8))

#second
classifier.add(Dense(8, activation='relu', kernel_initializer='random_normal'))

#output
classifier.add(Dense(1, activation='sigmoid', kernel_initializer='random_normal'))

#compiling the network
classifier.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

#fitting the data to the training set
history=classifier.fit(x_train, y_train, validation_split=0.33, batch_size=10, epochs=15)

#evaluate the loss value and metrics values for the model in test mode using evaluate funcn.
eval_model=classifier.evaluate(x_train, y_train)
print('eval model: ', eval_model)
```

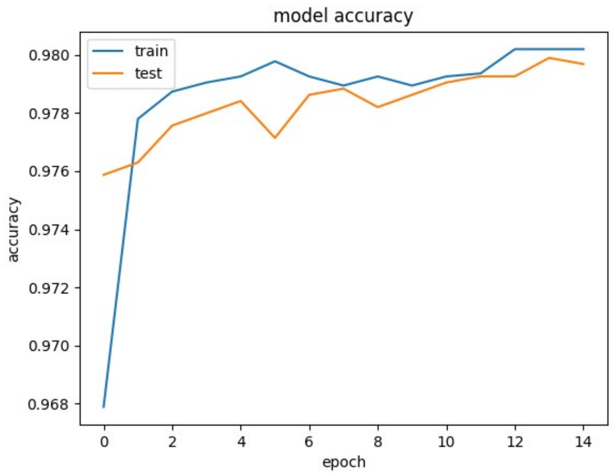
- 1 input layer, 1 hidden layer, 1 output layer
- 8 features, 1 output
- Minimum 8 neurons in the hidden layer
- Validation set to get an idea of accuracy per epoch

Artificial Neural Network

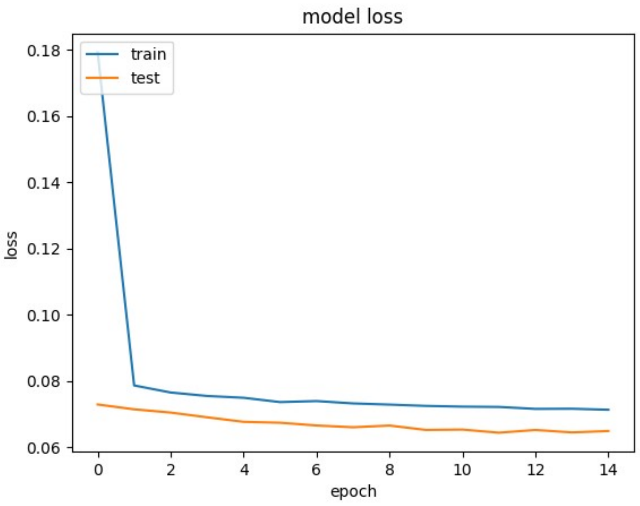
Example file: pulsar_ann.py

Visualization

```
cm=confusion_matrix(y_test, y_pred)
print(cm)

# list all data in history
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

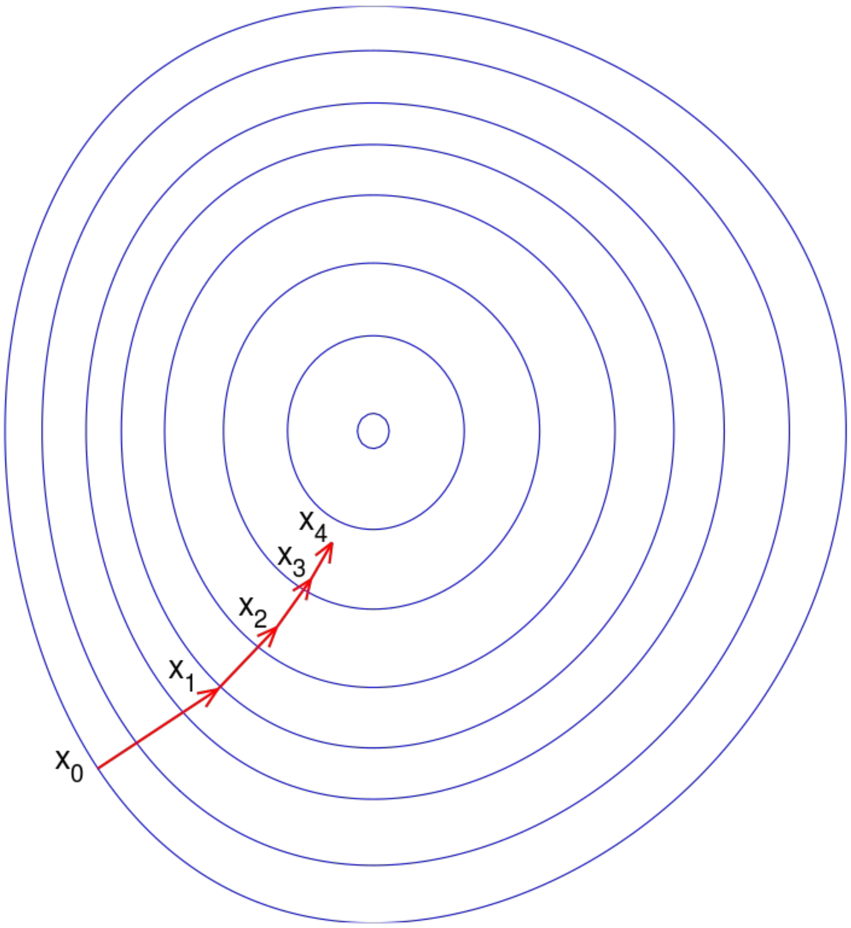



```
[[3233   26]
 [  42  279]]
```



Something cool to end: GANs

Backup: Gradient Descent



$$\theta_{i+1} = \theta_i - \gamma \nabla F(\theta_i)$$

- F(x) is differentiable around point theta
- F will decrease fastest if going in the direction of negative gradient
- Gamma is some real, positive number
- Converges to a local minimum