Machine Learning

Dr. Shahid Mahmood Awan
Assistant Professor
School of Systems and Technology, University of Management and Technology
shahid.awan@umt.edu.pk

Umer Saeed(MS Data Science, BSc Telecommunication Engineering)
Sr. RF Optimization & Planning Engineer
f2017313017@umt.edu.pk

A First Step towards Eye State Prediction Using EEG

Abstract

Abstract

- In this research paper, measuring brain waves with an Electroencephalography (EEG) device to examine how human eye state (open or close) can be predict.
- The data set consist of 14 electrodes of an EEG headset activation strength and in addition to that we manually marked the eye state corresponding to the recorded data.
- 16 different machine learning classification algorithms are tested on their performance to predict the eye state after training with the dataset.
- The best-performing classifier, K Nearest Neighbors (K-NN), produced a classification error rate of only ***4.04%***, accuracy ***95.96%*** and F-Score ***96.38%***.

A First Step towards Eye State Prediction Using EEG

Introduction

Introduction

- An electroencephalogram (EEG) test performed to evaluate the electrical activity in the brain. Neurons are communicate with each other through electrical impulses. An EEG can be used to diagnostic problems associated with this activity

- An EEG tracks and records brain wave patterns. Small flat metal devices called electrodes are connected to the scalp with wires. The electrodes analyze the electrical impulses in the brain and send signals to a computer that records the outputs.

- The electrical impulses in an EEG recording look like wavy lines with peaks and valleys. These lines allow doctors to quickly assess whether there are abnormal patterns. Any irregularities may be a sign of seizures or other brain disorders.

- An EEG is used to detect problems in the electrical activity of the brain that may be associated with certain brain disorders. The measurements given by an EEG are used to confirm or rule out various conditions, including: head injury, encephalitis, brain tumor, encephalopathy , memory problem, sleep disorders, stroke and dementia.

Introduction

- When someone is in a coma, an EEG may be performed to determine the level of brain activity. The test can also be used to monitor activity during brain surgery.

- There are no risks associated with an EEG. The test is painless and safe. Some EEGs do not include lights or other stimuli. If an EEG does not produce any abnormalities, stimuli such as strobe lights, or rapid breathing may be added to help induce any abnormalities. When someone has epilepsy or another seizure disorder, the stimuli presented during the test (such as a flashing light) may cause a seizure. The technician performing the EEG is trained to safely manage any situation that might occur.

- Before the test, you should take the following steps:
- 1- Wash your hair the night before the EEG, and don't put any products (like sprays or gels) in your hair on the day of the test.
- 2- Ask your doctor if you should stop taking any medications before the test. You should also make a list of your medications and give it to the technician performing the EEG.
- 3- Avoid eating or drinking anything containing caffeine for at least eight hours before the test.

Introduction

- 4- Your doctor may ask you to sleep as little as possible the night before the test if you have to sleep during the EEG. You may also be given a sedative to help you relax and sleep before the test begins.

- An EEG measures the electrical impulses in your brain by using several electrodes that are attached to your scalp. An electrode is a conductor through which an electric current enters or leaves. The electrodes transfer information from your brain to a machine that measures and records the data.

- Specialized technicians administer EEGs at hospitals, doctor's offices, and laboratories. The test usually takes 30 to 60 minutes to complete, and involves the following steps:
- 1- You'll lie down on your back in a reclining chair or on a bed.
- 2- The technician will measure your head and mark where to place the electrodes. These spots are scrubbed with a special cream that helps the electrodes get a high-quality reading.
- 3- The technician will put a sticky gel adhesive on electrodes, and attach them to spots on your scalp.

Introduction

- 4- Once the test begins, the electrodes send electrical impulse data from your brain to the recording machine. This machine converts the electrical impulses into visual patterns that appear on a screen. A computer saves these patterns. The technician may instruct you to do certain things while the test is in progress. They may ask you to lie still, close your eyes, breathe deeply, or look at stimuli (such as a flashing light or a picture).

A First Step towards Eye State Prediction Using EEG

Literature Review

Literature Review

- [1] Brain-Computer Interface (BCI) technology is a fast evolving field that seeks direct interaction between the human neural system and machines, aiming to augment human capabilities by enabling people (especially disabled) to communicate and control devices by mere "thinking" or expressing intent.

- [2] In recent years, researchers have attempted to develop hardware and software systems that can capture emotions automatically. This approach is called affective computing. One of those systems employs electroencephalography (EEG) signals recorded when users perform some brain activities. The advantages of this approach include; 1) Brain activities have direct information about emotion, 2) EEG signals can be measured at any moment and are not dependent on other activities of the user such as speaking or generating a facial expression, and 3) Different recognition techniques can be used

- We are all familiar with the ubiquity of the personal computer (PC) in modern society. Unfortunately, disabilities prevent many people from standard interaction with a PC. Many research activities have been undertaken in recent years to develop technologies to enable universal access to computer systems by reducing the dependency on sophisticated physical or cognitive skills

Literature Review

- [3] A different type of bio-signal, used in brain computer interfaces (BCI), is the brain-wave of the user, utilizing the analysis of an ongoing electroencephalographic (EEG) signal. EEG based systems require no physical activity, thus enabling even users suffering from complete paralysis but normal cognition to communicate using BCI (Markand, 1976). However, two problems are common to EEG based accessibility systems: speed and sensitivity to noise.

- [4] Neuroscientific knowledge can contribute to the military domain in designing and evaluating equipment (neuroergonomics), selection and training of personnel, increasing resilience and in user system communication

- [5] Several papers investigated the differences between the two eye states (that is, whether eyes are open or closed). Came to the conclusion that the “greatest difference between two states was that the power in the eye closed state was much higher than that in the eye open state.” However, the authors did not pursue this finding any further in an attempt to use power as a feature for predicting the eye state.

Literature Review

- [6] Investigated how to track eye blinking (the change of the eye state) based on EEG input. This study was limited to a single classification algorithm (artificial neural networks— ANNs) producing a very poor performance. Furthermore, eye blinking and eye state are intrinsically different properties in that the former is an event of a short duration whereas the latter can vary largely in duration

- [7] "Potential to be used as a switching mechanism for assistive technologies", the authors do not go so far as to implement an algorithm performing said classification, not to speak of measuring its performance on a collected data set.

- To render our research as useful as possible for the scientific and technological community, Oliver R¨osler and David Suendermann decided not to use a medical EEG but the Emotiv EPOC headset. Compared to a medical EEG, the EPOC headset is much more affordable, and it can be set up quickly and without the help of another person.

Literature Review

- Furthermore, we used the open-source software Python 3.6 (using libraries numpy, matplotlib, sklearn, pandas, theano, tensorflow, keras)

- After running extensive experiments with up to 16 different classifiers, we were able to achieve a classification error rate of 4.04% using the instance-based learner K Nearest Neighbors (K-NN). This result indicates that eye state prediction has the potential to be used as accurate binary input channel.

A First Step towards Eye State Prediction Using EEG

Materials And Methods

Materials And Methods

- The test conducted by Oliver Rosler and David Suendermann from Baden-Wuerttemberg Cooperative State University (DHBW), Stuttgart, Germany using Emotive EEG Neuroheadset from 14 electrode positions AF3, AF4, F3, F4, F7, F8, FC5, FC6, T7, T8, P7, P8, O1, and O2.

- The test was performed in quiet room. During the test, proband was being recoded. The proband was not aware of the exact start time of the measurement. He was advised to sit relaxed, look straight to the camera, and change the eye state at free fill. Only additional constraint was that, accumulated over the entire session, the duration of both eye states should be about the same and that the individual intervals should vary greatly in length.

- The eye state was recognized by means of a camera, amid the EEG continuous measurement and added later physically to the record in the wake of examining the feature outlines. ”1” signifies an eye-closed and”0” the eye-open state. All data samples are in sequential order with the initially measured worth at the highest point. Both, open or partially open eyes were categorized as closed, only completely closed were categorized as closed.

Materials And Methods

- The duration of the recording was 117 seconds. The sampling rate of the EEG headset A/D converter was four times the frame rate of the video camera. The sampling rate of the device is 128 Hz which imposes a minimum threshold on the recognition rate of our algorithm.

- The corpus consists of 14,980 instances of 15 attributes; 14 of them represent signals recorded from locations on the scalp (AF3, AF4, F3, F4, F7, F8, FC5, FC6, T7, T8, P7, P8, O1, and O2) and one represents the eye state (open/closed).

- Three instances (out of 14980) were removed from the corpus because of obvious transmission error.

Materials And Methods

	Eye Open		Eye Closed	
	Ranges	Mean	Ranges	Mean
AF3	4198-4504	4298	4199 -4445	4305
F7	3924-4157	4013	3906-4139	4005
F3	4197-4386	4263	4212-4367	4266
FC5	4073-4250	4123	4058-4214	4121
T7	4305-4464	4342	4310-4435	4342
P7	4566-4757	4621	4575-4709	4619
O1	4027-4178	4072	4026-4167	4074
O2	4567-4732	4615	4568-4696	4617
P8	4152-4320	4200	4148-4288	4203
T8	4153-4363	4229	4174-4323	4233
FC6	4100-4332	4200	4131-4319	4205
F4	4201-4398	4277	4226-4369	4282
F8	4443-4834	4602	4510-4811	4611
AF4	4206-4573	4357	4246-4553	4367

Table-1:: Ranges and means of the sensor values for the eye states

Materials And Methods

- **# Importing the libraries**

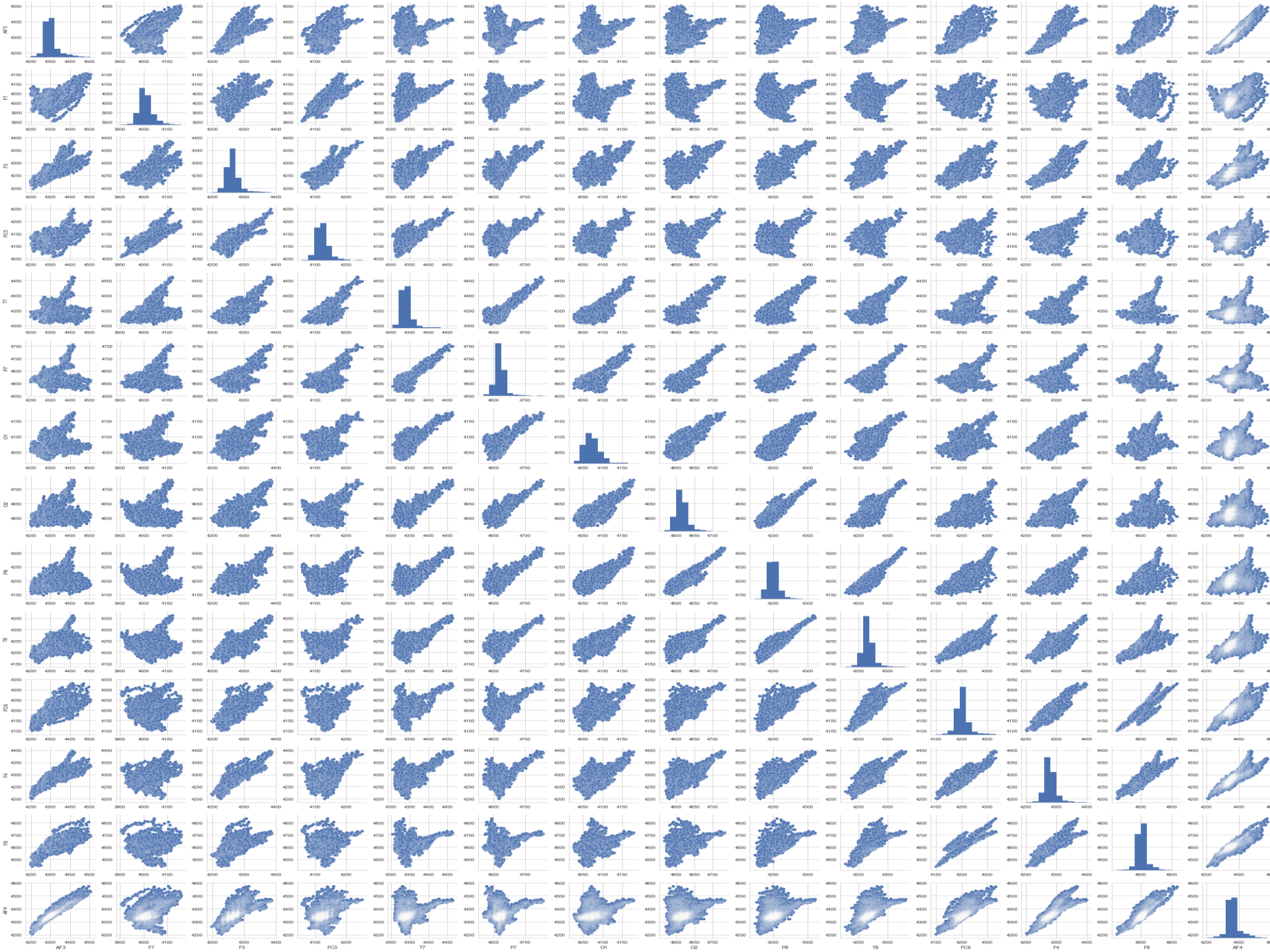
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
```

- **# Importing the data set**

```
dataset =pd.read_csv('EEG_Eye_State_Arff.csv')
```

- **# draw plots**

```
sns.set(style='whitegrid', context='notebook')
cols = ['AF3', 'F7', 'F3', 'FC5', 'T7','P7','O1','O2','P8','T8','FC6','F4','F8','AF4']
sns.pairplot(dataset[cols], size=2.5)
plt.show()
```


AF3
F7
F3
FC5
T7
P7
O1
O2
P8
T8
FC6
F4
F8
AF4

Materials And Methods

- **# Importing the libraries**

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
```

- **# Importing the data set**

```
dataset =pd.read_csv('EEG_Eye_State_Arff.csv')
```

- **# Correlation matrix**

```
corr_matrix=dataset.corr(method='pearson')
print(corr_matrix)
```

Materials And Methods

	AF3	F7	F3	FC5	T7	P7	O1	O2	P8	T8	FC6	F4	F8	AF4
AF3	1	0.585306	0.755241	0.606451	0.34892	0.230565	0.318301	0.224541	0.319118	0.501181	0.649732	0.797345	0.758434	0.943014
F7	0.585306	1	0.565313	0.74988	0.491221	0.329287	0.259225	0.10555	0.143408	0.177211	0.249957	0.376836	0.248467	0.413704
F3	0.755241	0.565313	1	0.765327	0.644033	0.595248	0.490685	0.545646	0.563064	0.640447	0.667272	0.833166	0.624255	0.707749
FC5	0.606451	0.74988	0.765327	1	0.679819	0.54406	0.395715	0.356698	0.368448	0.40066	0.406626	0.56486	0.360685	0.473129
T7	0.34892	0.491221	0.644033	0.679819	1	0.834613	0.662364	0.655191	0.639578	0.62936	0.533134	0.533141	0.36051	0.298553
P7	0.230565	0.329287	0.595248	0.54406	0.834613	1	0.65678	0.720635	0.70829	0.653217	0.49518	0.507925	0.309878	0.214368
O1	0.318301	0.259225	0.490685	0.395715	0.662364	0.65678	1	0.644568	0.667305	0.573874	0.51265	0.582686	0.385166	0.334048
O2	0.224541	0.10555	0.545646	0.356698	0.655192	0.720635	0.644568	1	0.866613	0.724022	0.594247	0.591611	0.414629	0.285467
P8	0.319118	0.143408	0.563064	0.368448	0.639578	0.70829	0.667305	0.866613	1	0.827987	0.678173	0.669177	0.537238	0.389127
T8	0.501181	0.177211	0.640447	0.40066	0.62936	0.653217	0.573874	0.724022	0.827987	1	0.801215	0.764319	0.713768	0.588576
FC6	0.649732	0.249957	0.667272	0.406626	0.533134	0.49518	0.51265	0.594247	0.678173	0.801215	1	0.835653	0.843172	0.74111
F4	0.797345	0.376836	0.833166	0.56486	0.533141	0.507925	0.582686	0.591611	0.669177	0.764319	0.835653	1	0.822963	0.838091
F8	0.758434	0.248467	0.624255	0.360685	0.36051	0.309878	0.385166	0.414629	0.537238	0.713768	0.843172	0.822963	1	0.864271
AF4	0.943014	0.413704	0.707749	0.473129	0.298553	0.214368	0.334048	0.285467	0.389127	0.588576	0.74111	0.838091	0.864271	1

Table-2:: correlation matrix

A First Step towards Eye State Prediction Using EEG

CLASSIFICATION

A First Step towards Eye State Prediction Using EEG

1- Logistic Regression

Hypothesis Representation

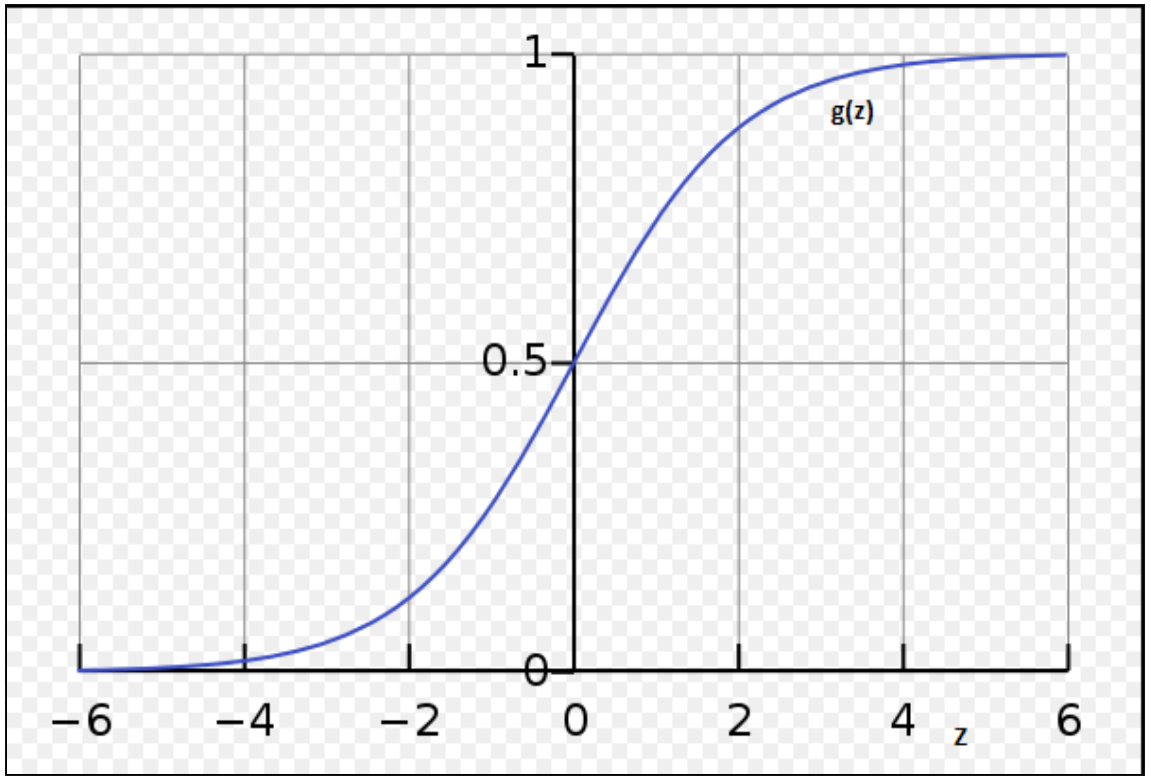
Want :$Logistic\ Regression: 0 \leq h_\theta(x) \leq 1$

$$h_\theta(x) = g(\theta^T \mathrm{x})$$

$$g(z) = \frac{1}{1+e^{-z}}\ ; \text{Logistic/Sigmoid function}$$

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T \mathrm{x}}}$$

- The function g(z), shown here, maps any real number to the (0, 1) interval, making it useful for transforming an arbitrary-valued function into a function better suited for classification.



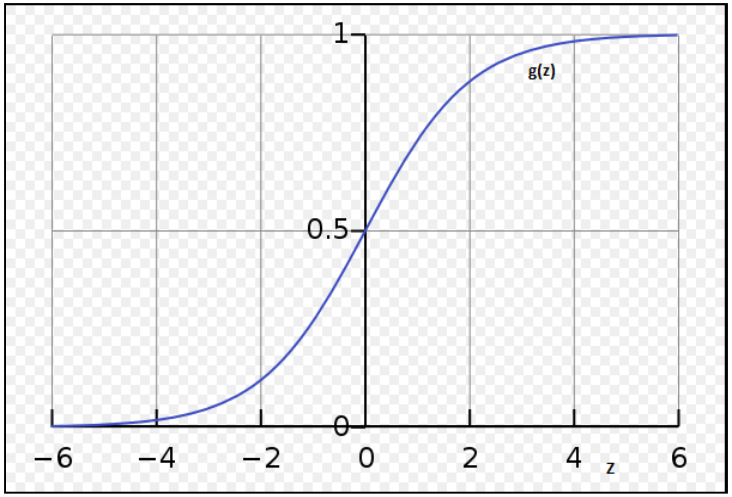
Decision Boundary

$$h_\theta(x) = g(\theta^T x) \qquad g(z) = \frac{1}{1+e^{-z}} \qquad h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

$$\textbf{Suppose predict "y=1"} if\ h_\theta(x) \geq 0.5$$

$$g(z) \geq 0.5 when\ z \geq 0$$

$$h_\theta(x) = g(\theta^T x) \geq 0.5 whenever\ \theta^T x \geq 0$$



$$\textbf{Predict "y=0"} if\ h_\theta(x) < 0.5$$

$$g(z) < 0.5 when\ z < 0$$

$$h_\theta(x) = g(\theta^T x) < 0.5 whenever\ \theta^T x < 0$$

Logistic Regression Cost Function

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost\left(h_\theta\left(x^i\right), y^{(i)}\right)$$

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y\left(log(h_\theta(x)\right) + (1 - \ y)\ log(1 - h_\theta(x))]$$

- This cost function is drives from statistics using the principle max likelihood estimation.
- The property of max likelihood estimation that is “convex”.

$$\boldsymbol{To\ fit\ parameters\ \theta}: \min_{\theta}(J(\theta))$$

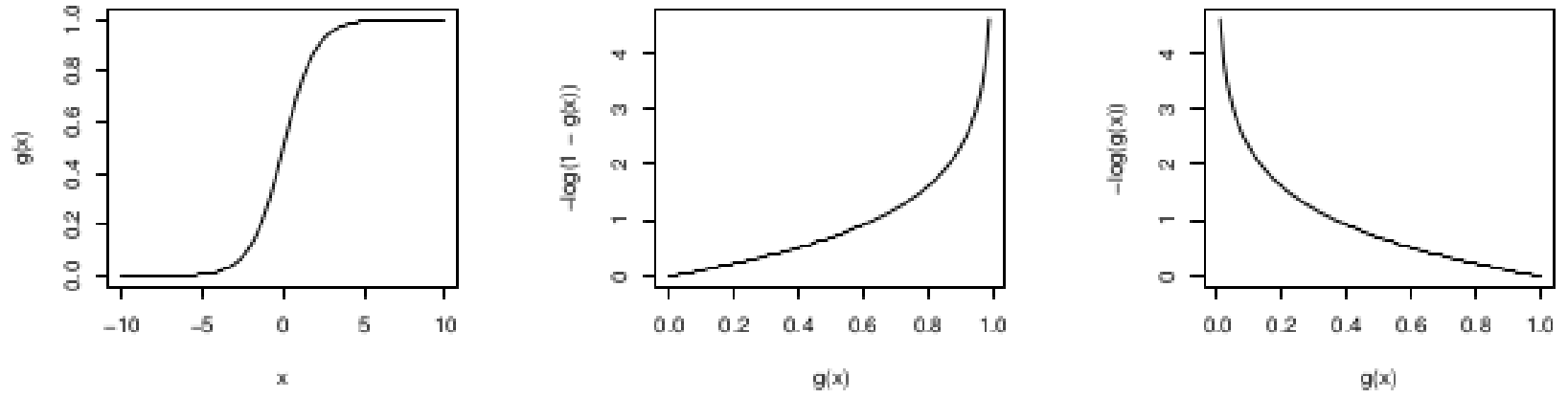
- To make a prediction given new x;

$$Logistic\ Regression\ Hypothesis: h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

$$h_\theta(x) = estimated\ probability\ that\ y = 1\ on\ input\ x\ (P(y = 1|x; \theta)$$

Logistic Regression Cost Function

$$Cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)) \;\; if \; y = 1 \\ -log(1 - h_\theta(x)) \;\; if \; y = 0 \end{cases}$$



(a) *Sigmoid function.* **(b)** *Cost for* $y = 0$. **(c)** *Cost for* $y = 1$.

Logarithmic transformation of the sigmoid function.

Logistic Regression

- **# Importing the libraries**

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

- **# Importing the dataset**

```
dataset =pd.read_csv('EEG_Eye_State_Arff.csv')
X = dataset.iloc[:, 0:14].values
y = dataset.iloc[:, 14].values
```

Logistic Regression

- **# Splitting the dataset into the Training set and Test set**

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)
```

- **# Feature Scaling**

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Logistic Regression

- **# Fitting Logistic Regression to the Training set**

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
```

- **# Predicting the Test set results**

```
y_pred = classifier.predict(X_test)
```

- **# Making the Confusion Matrix**

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

Logistic Regression

Actual class\Predicted class	eye-open=0	eye-closed=1	Total
eye-open=0	**1285(TP)**	**379(FN)**	**1664(P)**
eye-closed=1	**680(FP)**	**652(TN)**	**1332(N)**
Total	**1965**	**1031**	**2996**

$$Accuracy\ (or\ Recognition\ Rate) = \frac{TP + TN}{P + N} = \frac{1285 + 652}{1664 + 1332} = \frac{1937}{2996} = 0.6465 \approx \mathbf{64.65\%}$$

$$Error\ Rate\ (or\ Misclassification\ Rate) = \frac{FP + FN}{P + N} = \frac{680 + 379}{1664 + 1332} = \frac{1059}{2996} = 0.3535 \approx \mathbf{35.35\%}$$

$$Sensitivity(or\ True\ Positive\ Rate\ or\ Recall) = \frac{TP}{P} = \frac{1285}{1664} = 0.7722 \approx \mathbf{77.22\%}$$

$$Specificity\ (True\ negative\ Rate) = \frac{TN}{N} = \frac{652}{1332} = 0.4895 \approx \mathbf{48.95\%}$$

$$Precision = \frac{TP}{TP + FP} = \frac{1285}{1285 + 680} = \frac{1285}{1965} = 0.6539 \approx \mathbf{65.39\%}$$

$$F - Score = \frac{2 \times precision \times recall}{precision + recall} = \frac{2 \times 0.6539 \times 0.7722}{0.6539 + 0.7722} = \frac{1.01}{1.4262} = 0.7082 \approx \mathbf{70.82\%}$$

A First Step towards Eye State Prediction Using EEG

2- K Nearest Neighbors(K-NN)

Partitioning Algorithms

- Partitioning method: Partitioning a database ***D*** of ***n*** objects into a set of ***k*** clusters, such that the sum of squared distances is minimized (where c_i is the centroid or medoid of cluster C_i)

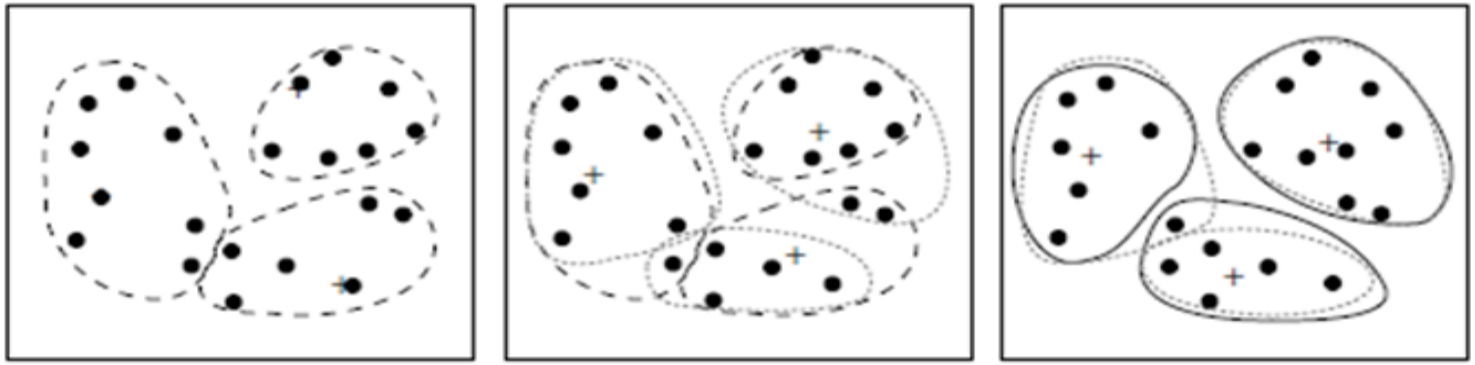
$$E = \Sigma_{i=1}^{k} \Sigma_{p \in C_i} (p - c_i)^2$$

- Given *k*, find a partition of *k clusters* that optimizes the chosen partitioning criterion
  - Global optimal: exhaustively enumerate all partitions
  - Heuristic methods: *k-means* and *k-medoids* algorithms
  - *k-means* (MacQueen'67, Lloyd'57/'82): Each cluster is represented by the center of the cluster
  - *k-medoids* or PAM (Partition around medoids) (Kaufman & Rousseeuw'87): Each cluster is represented by one of the objects in the cluster

The K-Means Clustering Method

- Given *k*, the *k-means* algorithm is implemented in four steps:
  1. Partition objects into *k* nonempty subsets
  2. Compute seed points as the centroids of the clusters of the current partitioning (the centroid is the center, i.e., *mean point*, of the cluster)
  3. Assign each object to the cluster with the nearest seed point
  4. Go back to Step 2, stop when the assignment does not change

The K-Means Clustering Method



(a) Initial clustering (b) Iterate (c) Final clustering

Clustering of a set of objects using the *k*-means method; for (b) update cluster centers and reassign objects accordingly (the mean of each cluster is marked by a +).

K Nearest Neighbors(K-NN)

- **# Importing the libraries**

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

- **# Importing the dataset**

```
dataset =pd.read_csv('EEG_Eye_State_Arff.csv')
X = dataset.iloc[:, 0:14].values
y = dataset.iloc[:, 14].values
```

K Nearest Neighbors(K-NN)

- **# Splitting the dataset into the Training set and Test set**

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)
```

- **# Feature Scaling**

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

K Nearest Neighbors(K-NN)

- **# Fitting K-NN to the Training set**

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)
```

- **# Predicting the Test set results**

```
y_pred = classifier.predict(X_test)
```

- **# Making the Confusion Matrix**

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

K Nearest Neighbors (K-NN)

Actual class\Predicted class	eye-open=0	eye-closed=1	Total
eye-open=0	**1610(TP)**	**54(FN)**	**1664(P)**
eye-closed=1	**67(FP)**	**1265(TN)**	**1332(N)**
Total	**1677**	**1319**	**2996**

$$Accuracy\ (or\ Recognition\ Rate) = \frac{TP + TN}{P + N} = \frac{1610 + 1265}{1664 + 1332} = \frac{2875}{2996} = 0.9596 \approx \mathbf{95.96\%}$$

$$Error\ Rate\ (or\ Misclassification\ Rate) = \frac{FP + FN}{P + N} = \frac{67 + 54}{1664 + 1332} = \frac{121}{2996} = 0.0404 \approx \mathbf{4.04\%}$$

$$Sensitivity(or\ True\ Positive\ Rate\ or\ Recall) = \frac{TP}{P} = \frac{1610}{1664} = 0.9675 \approx \mathbf{96.75\%}$$

$$Specificity\ (True\ negative\ Rate) = \frac{TN}{N} = \frac{1265}{1332} = 0.9497 \approx \mathbf{94.97\%}$$

$$Precision = \frac{TP}{TP + FP} = \frac{1610}{1610 + 67} = \frac{1610}{1677} = 0.9600 \approx \mathbf{96.00\%}$$

$$F - Score = \frac{2 \times precision \times recall}{precision + recall} = \frac{2 \times 0.96 \times 0.9675}{0.96 + 0.9675} = \frac{1.858}{1.928} = 0.9638 \approx \mathbf{96.38\%}$$

A First Step towards Eye State Prediction Using EEG

3-Support Vector Machine (SVM)

Support Vector Machine (SVM)

- **CLASSIFICATION SVM TYPE 1**

- For this type of SVM, training involves the minimization of the error function:

$$\frac{1}{2}w^T w + C\sum_{i=1}^{N}\varepsilon_i$$

subject to the constraints:

$$y_i(w^T\varphi(x_i) + b) \geq 1 - \varepsilon_i \ and \ \varepsilon_i \geq 0, i = 1,2,3 \ldots, N$$

Where C is the capacity constant, w is the vector of coefficients, b is a constant and ε_i represents parameters for handling non-separable data (inputs). The index i labels the N training cases. Note that $y \in \pm 1$ represents the class labels and xi represents the independent variables. The kernel φ is used to transform data from the input (independent) to the features space. It should be noted that the larger the C, the more the error is penalized, Thus, C should be chosen with carte to avoid over fitting.

Support Vector Machine (SVM)

- **CLASSIFICATION SVM TYPE 2**

- In contrast to Classification SVM Type 1, the Classification SVM Type 2 model minimizes the error function:

$$\frac{1}{2}w^T w - v\rho + \frac{1}{N}\sum_{i=1}^{N}\varepsilon_i$$

- subject to the constraints:

$$y_i(w^T\varphi(x_i) + b) \geq \rho - \varepsilon_i \text{ and } \varepsilon_i \geq 0, i = 1,2,3 \ldots, N \text{ and } \rho \geq 0$$

Support Vector Machine (SVM)

- **# Importing the libraries**

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

- **# Importing the dataset**

```
dataset =pd.read_csv('EEG_Eye_State_Arff.csv')
X = dataset.iloc[:, 0:14].values
y = dataset.iloc[:, 14].values
```

Support Vector Machine (SVM)

- **# Splitting the dataset into the Training set and Test set**

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)
```

- **# Feature Scaling**

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Support Vector Machine (SVM)

- **# Fitting SVM to the Training set**

```
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)
classifier.fit(X_train, y_train)
```

- **# Predicting the Test set results**

```
y_pred = classifier.predict(X_test)
```

- **# Making the Confusion Matrix**

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

Support Vector Machine (SVM)

Actual class\Predicted class	eye-open=0	eye-closed=1	Total
eye-open=0	**1370(TP)**	**294(FN)**	**1664(P)**
eye-closed=1	**700(FP)**	**632(TN)**	**1332(N)**
Total	**2070**	**926**	**2996**

$$Accuracy\ (or\ Recognition\ Rate) = \frac{TP + TN}{P + N} = \frac{1370 + 632}{1664 + 1332} = \frac{2002}{2996} = 0.6682 \approx \mathbf{66.82\%}$$

$$Error\ Rate\ (or\ Misclassification\ Rate) = \frac{FP + FN}{P + N} = \frac{700 + 294}{1664 + 1332} = \frac{994}{2996} = 0.3318 \approx \mathbf{33.18\%}$$

$$Sensitivity(or\ True\ Positive\ Rate\ or\ Recall) = \frac{TP}{P} = \frac{1370}{1664} = 0.8233 \approx \mathbf{82.33\%}$$

$$Specificity\ (True\ negative\ Rate) = \frac{TN}{N} = \frac{632}{1332} = 0.4745 \approx \mathbf{47.45\%}$$

$$Precision = \frac{TP}{TP + FP} = \frac{1370}{1370 + 700} = \frac{1370}{2070} = 0.6618 \approx \mathbf{66.18\%}$$

$$F - Score = \frac{2 \times precision \times recall}{precision + recall} = \frac{2 \times 0.6618 \times 0.8233}{0.6618 + 0.8233} = \frac{1.09}{1.485} = 0.7338 \approx \mathbf{73.38\%}$$

A First Step towards Eye State Prediction Using EEG

4- SVM (Radial Basis Function kernel (RBF) or Gaussian kernel)

SVM (Radial Basis Function kernel (RBF) or Gaussian kernel)

- It is general-purpose kernel; used when there is no prior knowledge about the data. Equation is:

$$k(x_i, x_j) = \exp\left(-\gamma \|x_i - x_j\|^2\right) \qquad for\ \gamma > 0$$

- Sometimes parametrized using:

$$\gamma = \frac{1}{2\sigma^2}$$

SVM (Radial Basis Function kernel (RBF) or Gaussian kernel)

- **# Importing the libraries**

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

- **# Importing the dataset**

```
dataset =pd.read_csv('EEG_Eye_State_Arff.csv')
X = dataset.iloc[:, 0:14].values
y = dataset.iloc[:, 14].values
```

SVM (Radial Basis Function kernel (RBF) or Gaussian kernel)

- **# Splitting the dataset into the Training set and Test set**

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)
```

- **# Feature Scaling**

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

SVM (Radial Basis Function kernel (RBF) or Gaussian kernel)

- **# Fitting Kernel SVM to the Training set**

```
from sklearn.svm import SVC
classifier = SVC(kernel = 'rbf', random_state = 0)
classifier.fit(X_train, y_train)
```

- **# Predicting the Test set results**

```
y_pred = classifier.predict(X_test)
```

- **# Making the Confusion Matrix**

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

SVM (Radial Basis Function kernel (RBF) or Gaussian kernel)

Actual class\Predicted class	eye-open=0	eye-closed=1	Total
eye-open=0	**1543(TP)**	**121(FN)**	**1664(P)**
eye-closed=1	**189(FP)**	**1143(TN)**	**1332(N)**
Total	**1732**	**1264**	**2996**

$$Accuracy\ (or\ Recognition\ Rate) = \frac{TP + TN}{P + N} = \frac{1543 + 1143}{1664 + 1332} = \frac{2686}{2996} = 0.8965 \approx \mathbf{89.65\%}$$

$$Error\ Rate\ (or\ Misclassification\ Rate) = \frac{FP + FN}{P + N} = \frac{189 + 121}{1664 + 1332} = \frac{310}{2996} = 0.1035 \approx \mathbf{10.35\%}$$

$$Sensitivity(or\ True\ Positive\ Rate\ or\ Recall) = \frac{TP}{P} = \frac{1543}{1664} = 0.9273 \approx \mathbf{92.73\%}$$

$$Specificity\ (True\ negative\ Rate) = \frac{TN}{N} = \frac{1143}{1332} = 0.8581 \approx \mathbf{85.81\%}$$

$$Precision = \frac{TP}{TP + FP} = \frac{1543}{1543 + 189} = \frac{1543}{1732} = 0.8909 \approx \mathbf{89.09\%}$$

$$F - Score = \frac{2 \times precision \times recall}{precision + recall} = \frac{2 \times 0.8909 \times 0.9273}{0.8909 + 0.9273} = \frac{1.652}{1.818} = 0.9087 \approx \mathbf{90.87\%}$$

A First Step towards Eye State Prediction Using EEG

5- SVM (Polynomial kernel)

SVM (Polynomial kernel)

- It is popular in image processing. Equation is:

$$k(x_i, x_j) = (x_i \cdot x_j + 1)^d$$

- where d is the degree of the polynomial.

SVM (Polynomial kernel)

- **# Importing the libraries**

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

- **# Importing the dataset**

```
dataset =pd.read_csv('EEG_Eye_State_Arff.csv')
X = dataset.iloc[:, 0:14].values
y = dataset.iloc[:, 14].values
```

SVM (Polynomial kernel)

- **# Splitting the dataset into the Training set and Test set**

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)
```

- **# Feature Scaling**

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

SVM (Polynomial kernel)

- **# Fitting Kernel SVM to the Training set**

```
from sklearn.svm import SVC
classifier = SVC(kernel = 'poly', random_state = 0)
classifier.fit(X_train, y_train)
```

- **# Predicting the Test set results**

```
y_pred = classifier.predict(X_test)
```

- **# Making the Confusion Matrix**

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

SVM (Polynomial kernel)

Actual class\Predicted class	eye-open=0	eye-closed=1	Total
eye-open=0	**1637(TP)**	**27(FN)**	**1664(P)**
eye-closed=1	**644(FP)**	**688(TN)**	**1332(N)**
Total	**2281**	**715**	**2996**

$$Accuracy\ (or\ Recognition\ Rate) = \frac{TP + TN}{P + N} = \frac{1637 + 668}{1664 + 1332} = \frac{2325}{2996} = 0.7760 \approx \mathbf{77.60\%}$$

$$Error\ Rate\ (or\ Misclassification\ Rate) = \frac{FP + FN}{P + N} = \frac{644 + 27}{1664 + 1332} = \frac{671}{2996} = 0.2240 \approx \mathbf{22.40\%}$$

$$Sensitivity(or\ True\ Positive\ Rate\ or\ Recall) = \frac{TP}{P} = \frac{1637}{1664} = 0.9838 \approx \mathbf{98.38\%}$$

$$Specificity\ (True\ negative\ Rate) = \frac{TN}{N} = \frac{688}{1332} = 0.5165 \approx \mathbf{51.65\%}$$

$$Precision = \frac{TP}{TP + FP} = \frac{1637}{1637 + 644} = \frac{1637}{2281} = 0.7177 \approx \mathbf{71.77\%}$$

$$F - Score = \frac{2 \times precision \times recall}{precision + recall} = \frac{2 \times 0.7177 \times 0.9838}{0.7177 + 0.9838} = \frac{1.412}{1.701} = 0.8299 \approx \mathbf{82.99\%}$$

A First Step towards Eye State Prediction Using EEG

6- SVM (sigmoid kernel)

SVM (sigmoid kernel)

- We can use it as proxy for neural networks. Equation is

$$k(x, y) = \tan \mathrm{h}(\alpha x^T y + C)$$

SVM (sigmoid kernel)

- **# Importing the libraries**

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

- **# Importing the dataset**

```
dataset =pd.read_csv('EEG_Eye_State_Arff.csv')
X = dataset.iloc[:, 0:14].values
y = dataset.iloc[:, 14].values
```

SVM (sigmoid kernel)

- **# Splitting the dataset into the Training set and Test set**

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)
```

- **# Feature Scaling**

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

SVM (sigmoid kernel)

- **# Fitting Kernel SVM to the Training set**

```
from sklearn.svm import SVC
classifier = SVC(kernel = 'sigmoid', random_state = 0)
classifier.fit(X_train, y_train)
```

- **# Predicting the Test set results**

```
y_pred = classifier.predict(X_test)
```

- **# Making the Confusion Matrix**

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

SVM (sigmoid kernel)

Actual class\Predicted class	eye-open=0	eye-closed=1	Total
eye-open=0	**910(TP)**	**754(FN)**	**1664(P)**
eye-closed=1	**692(FP)**	**640(TN)**	**1332(N)**
Total	**1602**	**1394**	**2996**

$$Accuracy\ (or\ Recognition\ Rate) = \frac{TP + TN}{P + N} = \frac{910 + 640}{1664 + 1332} = \frac{1550}{2996} = 0.5174 \approx \mathbf{51.74\%}$$

$$Error\ Rate\ (or\ Misclassification\ Rate) = \frac{FP + FN}{P + N} = \frac{692 + 754}{1664 + 1332} = \frac{1446}{2996} = 0.4826 \approx \mathbf{48.26\%}$$

$$Sensitivity(or\ True\ Positive\ Rate\ or\ Recall) = \frac{TP}{P} = \frac{910}{1664} = 0.5469 \approx \mathbf{54.69\%}$$

$$Specificity\ (True\ negative\ Rate) = \frac{TN}{N} = \frac{640}{1332} = 0.4805 \approx \mathbf{48.05\%}$$

$$Precision = \frac{TP}{TP + FP} = \frac{910}{910 + 692} = \frac{910}{1602} = 0.5680 \approx \mathbf{56.80\%}$$

$$F - Score = \frac{2 \times precision \times recall}{precision + recall} = \frac{2 \times 0.5680 \times 0.5469}{0.5680 + 0.5469} = \frac{0.621}{1.115} = 0.5573 \approx \mathbf{55.73\%}$$

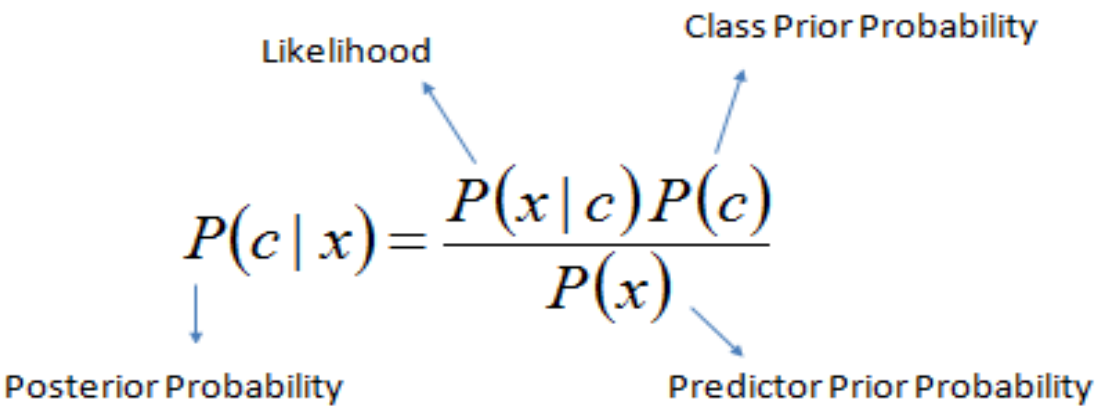
A First Step towards Eye State Prediction Using EEG

7-Naïve Bayes

Bayes' Theorem: Basics

- Total probability Theorem: $P(B) = \sum_{i=1}^{M} P(B|A_i)P(A_i)$
- Bayes' Theorem: $$P(H|\mathbf{X}) = \frac{P(\mathbf{X}|H)P(H)}{P(\mathbf{X})} = P(\mathbf{X}|H) \times P(H) / P(\mathbf{X})$$
  - Let **X** be a data sample ("*evidence*"): class label is unknown
  - Let H be a *hypothesis* that X belongs to class C
  - Classification is to determine P(H|**X**), (i.e., *posteriori probability):* the probability that the hypothesis holds given the observed data sample **X**
  - P(H) (*prior probability*): the initial probability
    - E.g., **X** will buy computer, regardless of age, income, …
  - P(**X**): probability that sample data is observed
  - P(**X**|H) (likelihood): the probability of observing the sample **X**, given that the hypothesis holds
    - E.g., Given that **X** will buy computer, the prob. that X is 31..40, medium income

Bayes' Theorem: Basics



$$P(c \mid \mathbf{X}) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

- *P*(*c*|*x*) is the posterior probability of *class* (c, *target*) given *predictor* (x, *attributes*).
- *P*(*c*) is the prior probability of *class*.
- *P*(*x*|*c*) is the likelihood which is the probability of *predictor* given *class*.
- *P*(*x*) is the prior probability of *predictor*.

Prediction Based on Bayes' Theorem

- Given training data **X**, *posteriori probability of a hypothesis* H, P(H|**X**), follows the Bayes' theorem

$$P(H|\mathbf{X}) = \frac{P(\mathbf{X}|H)P(H)}{P(\mathbf{X})} = P(\mathbf{X}|H) \times P(H) / P(\mathbf{X})$$

- Informally, this can be viewed as

  posteriori = likelihood x prior/evidence

- Predicts **X** belongs to C_i iff the probability $P(C_i|\mathbf{X})$ is the highest among all the $P(C_k|X)$ for all the *k* classes
- Practical difficulty: It requires initial knowledge of many probabilities, involving significant computational cost

Classification Is to Derive the Maximu Posteriori

- Let D be a training set of tuples and their associated class labels, and each tuple is represented by an n-D attribute vector **X** = (x_1, x_2, …, x_n)
- Suppose there are *m* classes C_1, C_2, …, C_m.
- Classification is to derive the maximum posteriori, i.e., the maximal P(C_i|**X**)
- This can be derived from Bayes' theorem

$$P(C_i|\mathbf{X}) = \frac{P(\mathbf{X}|C_i)P(C_i)}{P(\mathbf{X})}$$

- Since P(X) is constant for all classes, only

$$P(C_i|\mathbf{X}) = P(\mathbf{X}|C_i)P(C_i)$$

needs to be maximized

Naïve Bayes Classifier

- A simplified assumption: attributes are conditionally independent (i.e., no dependence relation between attributes):

$$P(\mathbf{X}|C_i) = \prod_{k=1}^{n} P(x_k|C_i) = P(x_1|C_i) \times P(x_2|C_i) \times ... \times P(x_n|C_i)$$

- This greatly reduces the computation cost: Only counts the class distribution
- If A_k is categorical, $P(x_k|C_i)$ is the # of tuples in C_i having value x_k for A_k divided by $|C_{i,D}|$ (# of tuples of C_i in D)
- If A_k is continous-valued, $P(x_k|C_i)$ is usually computed based on Gaussian distribution with a mean μ and standard deviation σ

and $P(x_k|C_i)$ is

$$g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$P(\mathbf{X}|C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i})$$

Naïve Bayes

- **# Importing the libraries**

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

- **# Importing the dataset**

```
dataset =pd.read_csv('EEG_Eye_State_Arff.csv')
X = dataset.iloc[:, 0:14].values
y = dataset.iloc[:, 14].values
```

Naïve Bayes

- **# Splitting the dataset into the Training set and Test set**

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)
```

- **# Feature Scaling**

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Naïve Bayes

- **# Fitting Naïve Bayes to the Training set**

```
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
```

- **# Predicting the Test set results**

```
y_pred = classifier.predict(X_test)
```

- **# Making the Confusion Matrix**

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

Naïve Bayes

Actual class\Predicted class	eye-open=0	eye-closed=1	Total
eye-open=0	**966(TP)**	**698(FN)**	**1664(P)**
eye-closed=1	**478(FP)**	**854(TN)**	**1332(N)**
Total	**1444**	**1552**	**2996**

$$Accuracy\ (or\ Recognition\ Rate) = \frac{TP + TN}{P + N} = \frac{966 + 854}{1664 + 1332} = \frac{1820}{2996} = 0.6075 \approx \mathbf{60.75\%}$$

$$Error\ Rate\ (or\ Misclassification\ Rate) = \frac{FP + FN}{P + N} = \frac{478 + 698}{1664 + 1332} = \frac{1176}{2996} = 0.3925 \approx \mathbf{39.25\%}$$

$$Sensitivity(or\ True\ Positive\ Rate\ or\ Recall) = \frac{TP}{P} = \frac{996}{1664} = 0.5805 \approx \mathbf{58.05\%}$$

$$Specificity\ (True\ negative\ Rate) = \frac{TN}{N} = \frac{854}{1332} = 0.6411 \approx \mathbf{64.11\%}$$

$$Precision = \frac{TP}{TP + FP} = \frac{966}{966 + 478} = \frac{966}{1444} = 0.6690 \approx \mathbf{66.90\%}$$

$$F - Score = \frac{2 \times precision \times recall}{precision + recall} = \frac{2 \times 0.6690 \times 0.5805}{0.6690 + 0.5805} = \frac{0.777}{1.25} = 0.6216 \approx \mathbf{62.16\%}$$

A First Step towards Eye State Prediction Using EEG

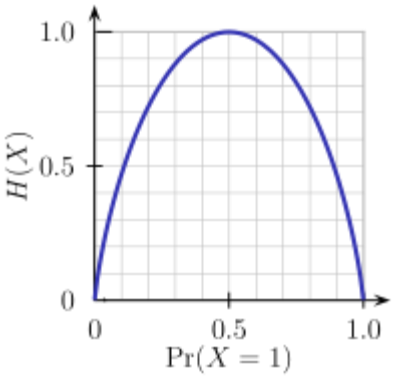
8-Decision Tree Classification

Algorithm for Decision Tree Induction

- Basic algorithm (a greedy algorithm)
  - Tree is constructed in a top-down recursive divide-and-conquer manner
  - At start, all the training examples are at the root
  - Attributes are categorical (if continuous-valued, they are discretized in advance)
  - Examples are partitioned recursively based on selected attributes
  - Test attributes are selected on the basis of a heuristic or statistical measure (e.g., information gain)
- Conditions for stopping partitioning
  - All samples for a given node belong to the same class
  - There are no remaining attributes for further partitioning – majority voting is employed for classifying the leaf
  - There are no samples left

Brief Review of Entropy

- Entropy (Information Theory)
  - A measure of uncertainty associated with a random variable
  - Calculation: For a discrete random variable Y taking m distinct values $\{y_1, \ldots, y_m\}$,
    - $H(Y) = -\sum_{i=1}^{m} p_i \log(p_i)$, where $p_i = P(Y = y_i)$
  - Interpretation:
    - Higher entropy => higher uncertainty
    - Lower entropy => lower uncertainty
- Conditional Entropy
  - $H(Y|X) = \sum_x p(x) H(Y|X = x)$



m = 2

Information Gain

- ID3 uses **information gain as its attribute selection measure**
- The attribute with the highest information gain is chosen as the splitting attribute for node *N.*
- *This attribute minimizes the information needed to classify the tuples in the* resulting partitions and reflects the least randomness or “impurity” in these partitions.

Attribute Selection Measure: Information Gain (ID3/C4.5)

- Select the attribute with the highest information gain
- Let p_i be the probability that an arbitrary tuple in D belongs to class C_i, estimated by $|C_{i,D}|/|D|$

- Expected information (entropy) needed to classify a tuple in D:

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

- Information gained by branching on attribute A

$$Gain(A) = Info(D) - Info_A(D)$$

Decision Tree Classification

- **# Importing the libraries**

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

- **# Importing the dataset**

```
dataset =pd.read_csv('EEG_Eye_State_Arff.csv')
X = dataset.iloc[:, 0:14].values
y = dataset.iloc[:, 14].values
```

Decision Tree Classification

- **# Splitting the dataset into the Training set and Test set**

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)
```

- **# Feature Scaling**

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Decision Tree Classification

- **# Fitting Decision Tree Classification to the Training set**

```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)
```

- **# Predicting the Test set results**

```
y_pred = classifier.predict(X_test)
```

- **# Making the Confusion Matrix**

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

Decision Tree Classification

Actual class\Predicted class	eye-open=0	eye-closed=1	Total
eye-open=0	**1414(TP)**	**250(FN)**	**1664(P)**
eye-closed=1	**223(FP)**	**1109(TN)**	**1332(N)**
Total	**1637**	**1359**	**2996**

$$Accuracy\ (or\ Recognition\ Rate) = \frac{TP + TN}{P + N} = \frac{1414 + 1109}{1664 + 1332} = \frac{2523}{2996} = 0.8421 \approx \mathbf{84.21\%}$$

$$Error\ Rate\ (or\ Misclassification\ Rate) = \frac{FP + FN}{P + N} = \frac{223 + 250}{1664 + 1332} = \frac{473}{2996} = 0.1579 \approx \mathbf{15.79\%}$$

$$Sensitivity(or\ True\ Positive\ Rate\ or\ Recall) = \frac{TP}{P} = \frac{1414}{1664} = 0.8498 \approx \mathbf{84.98\%}$$

$$Specificity\ (True\ negative\ Rate) = \frac{TN}{N} = \frac{1109}{1332} = 0.8326 \approx \mathbf{83.26\%}$$

$$Precision = \frac{TP}{TP + FP} = \frac{1414}{1414 + 223} = \frac{1414}{1637} = 0.8638 \approx \mathbf{86.38\%}$$

$$F - Score = \frac{2 \times precision \times recall}{precision + recall} = \frac{2 \times 08638 \times 0.8498}{0.8638 + 0.8498} = \frac{1.468}{1.714} = 0.8567 \approx \mathbf{85.67\%}$$

A First Step towards Eye State Prediction Using EEG

9-Random Forest Classification

Random Forest Classification

- **# Importing the libraries**

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

- **# Importing the dataset**

```
dataset =pd.read_csv('EEG_Eye_State_Arff.csv')
X = dataset.iloc[:, 0:14].values
y = dataset.iloc[:, 14].values
```

Random Forest Classification

- **# Splitting the dataset into the Training set and Test set**

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)
```

- **# Feature Scaling**

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Random Forest Classification

- **# Fitting Decision Tree Classification to the Training set**

```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)
```

- **# Predicting the Test set results**

```
y_pred = classifier.predict(X_test)
```

- **# Making the Confusion Matrix**

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

Random Forest Classification

Actual class\Predicted class	eye-open=0	eye-closed=1	Total
eye-open=0	**1572(TP)**	**92(FN)**	**1664(P)**
eye-closed=1	**212(FP)**	**1120(TN)**	**1332(N)**
Total	**1784**	**1212**	**2996**

$$Accuracy\ (or\ Recognition\ Rate) = \frac{TP + TN}{P + N} = \frac{1572 + 1120}{1664 + 1332} = \frac{2692}{2996} = 0.8985 \approx \mathbf{89.85\%}$$

$$Error\ Rate\ (or\ Misclassification\ Rate) = \frac{FP + FN}{P + N} = \frac{212 + 92}{1664 + 1332} = \frac{304}{2996} = 0.1015 \approx \mathbf{10.15\%}$$

$$Sensitivity(or\ True\ Positive\ Rate\ or\ Recall) = \frac{TP}{P} = \frac{1572}{1664} = 0.9447 \approx \mathbf{94.47\%}$$

$$Specificity\ (True\ negative\ Rate) = \frac{TN}{N} = \frac{1120}{1332} = 0.8408 \approx \mathbf{84.08\%}$$

$$Precision = \frac{TP}{TP + FP} = \frac{1572}{1572 + 212} = \frac{1572}{1784} = 0.8812 \approx \mathbf{88.12\%}$$

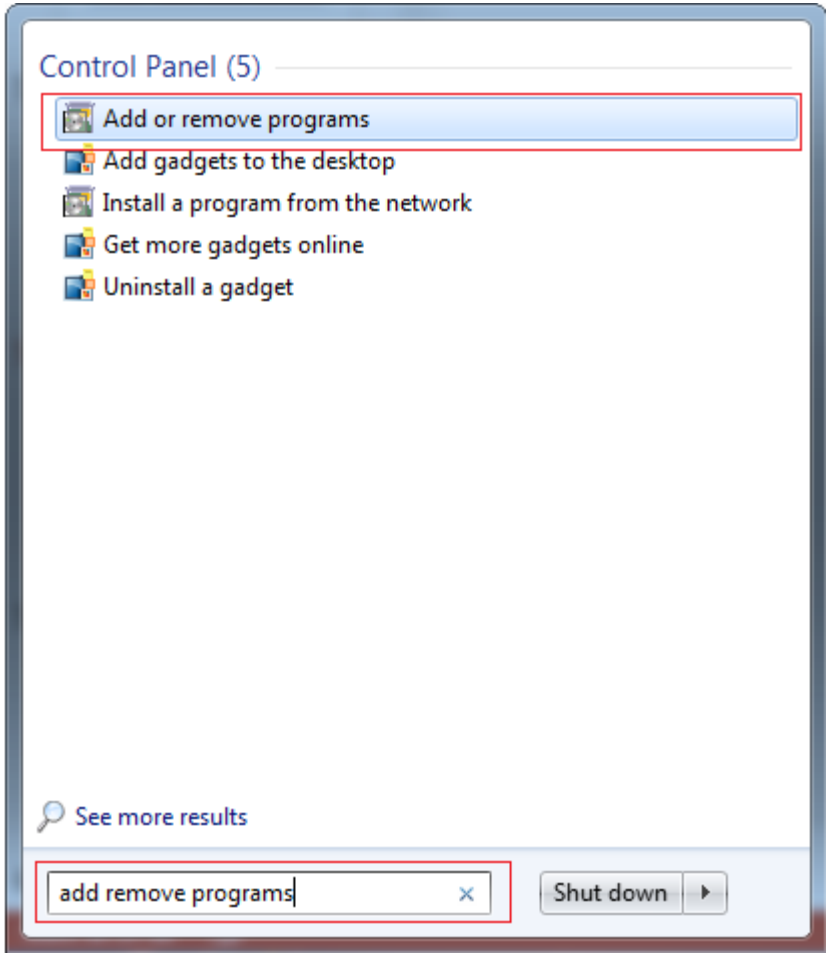
$$F - Score = \frac{2 \times precision \times recall}{precision + recall} = \frac{2 \times 0.8812 \times 0.9447}{0.8812 + 0.9447} = \frac{1.665}{1.826} = 0.9118 \approx \mathbf{91.18\%}$$

A First Step towards Eye State Prediction Using EEG

Install Theano/TensorFlow/Keras

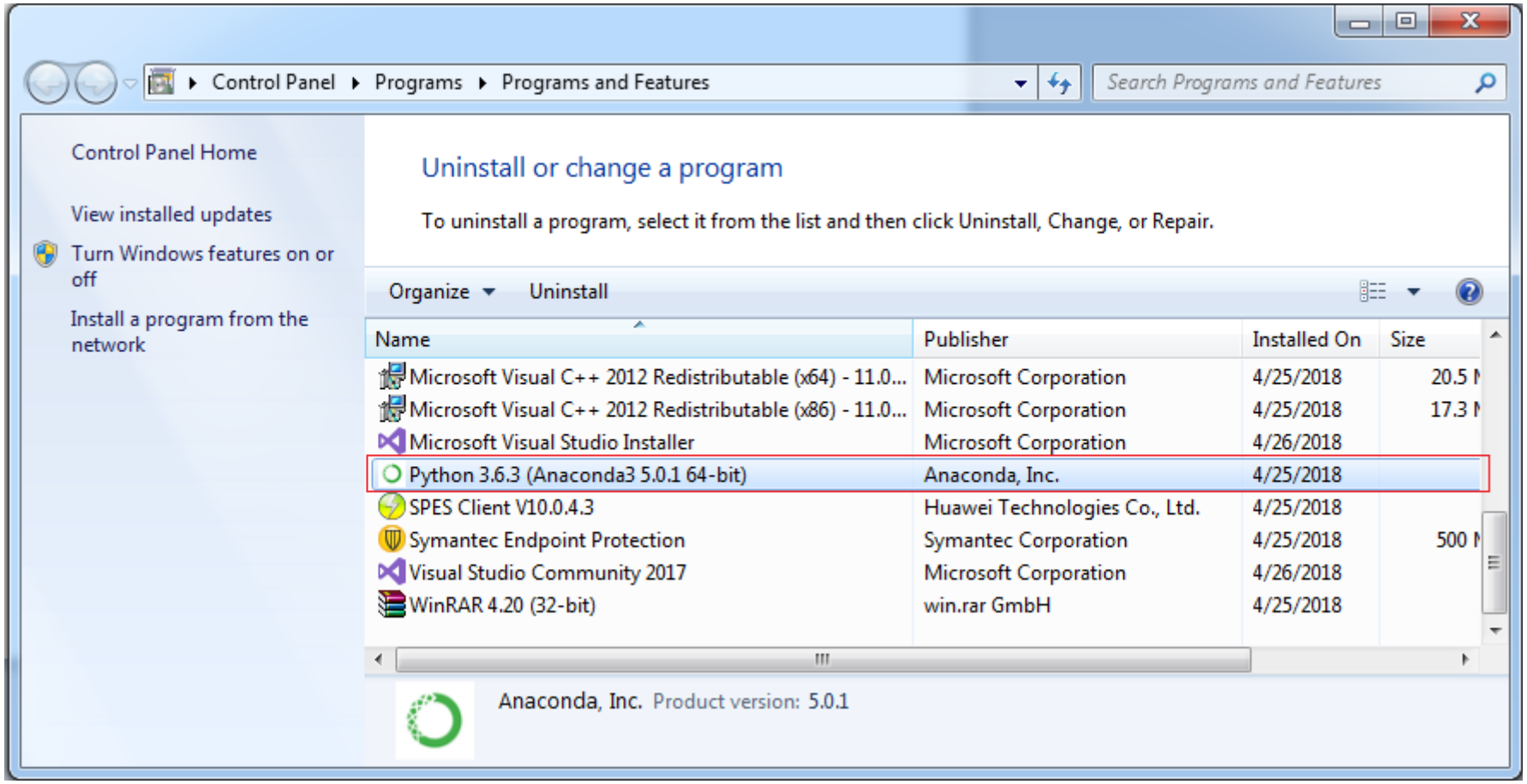
Installation Step-by-Step

- Open add remove programs.



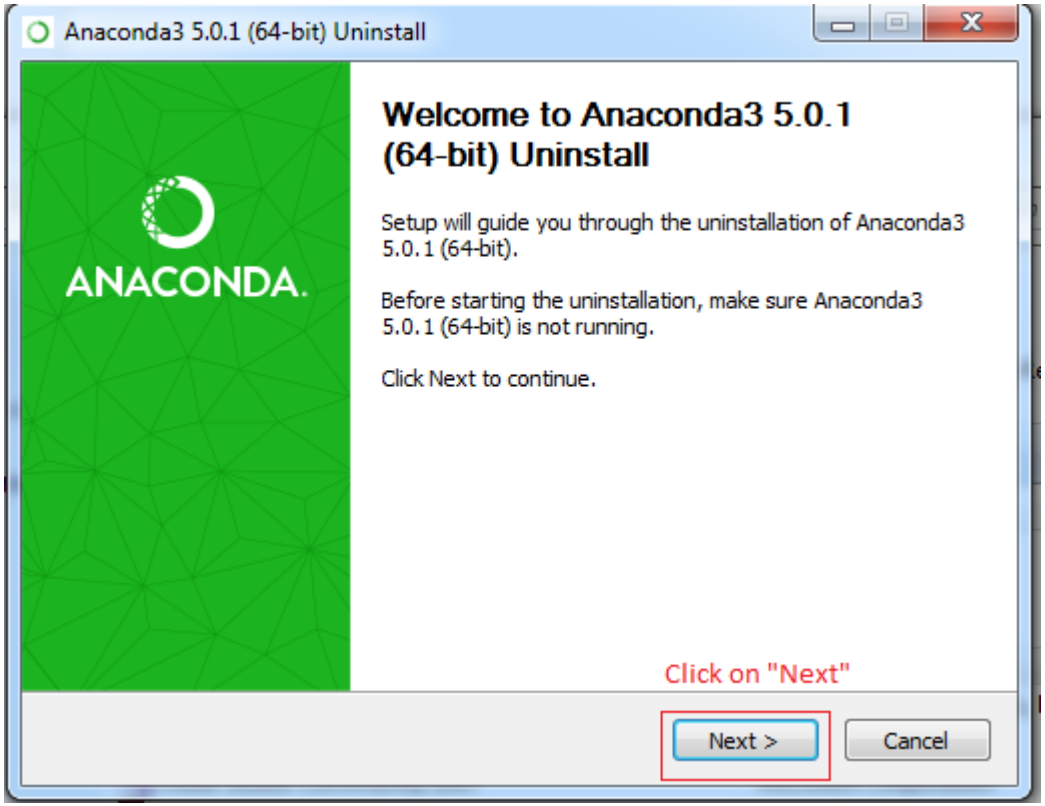
Installation Step-by-Step

- Click on “Python 3.6.3(Anaconda 3 5.0.1 64-bit)”



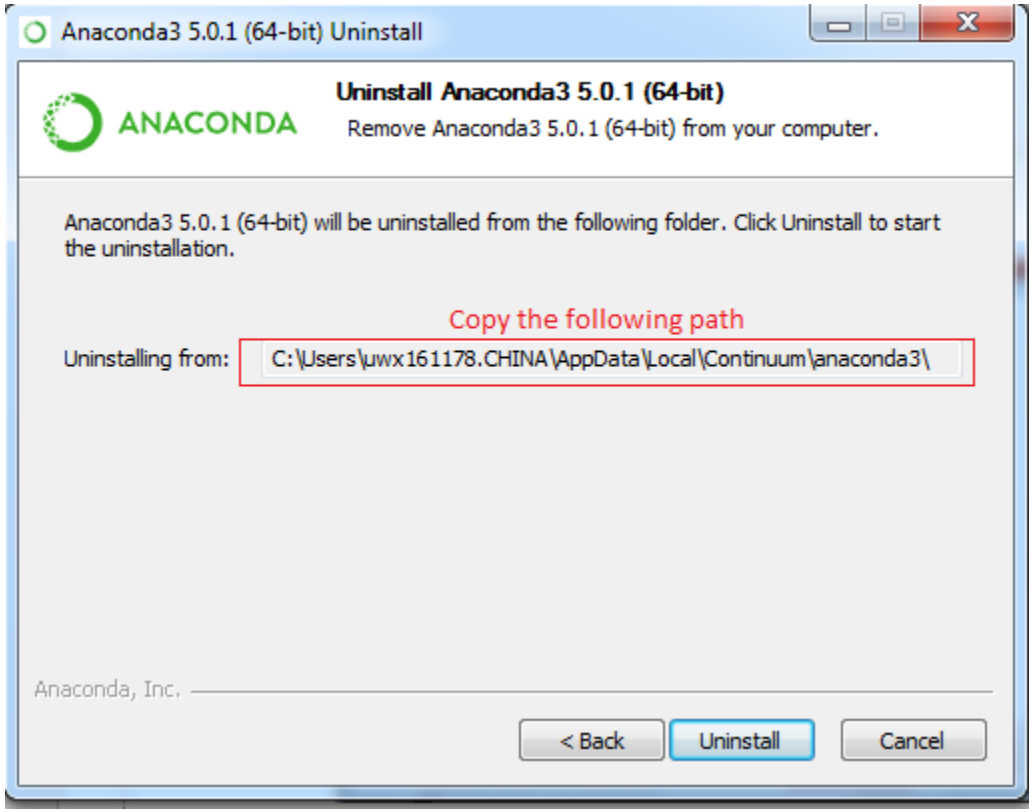
Installation Step-by-Step

- Click on “Next”.



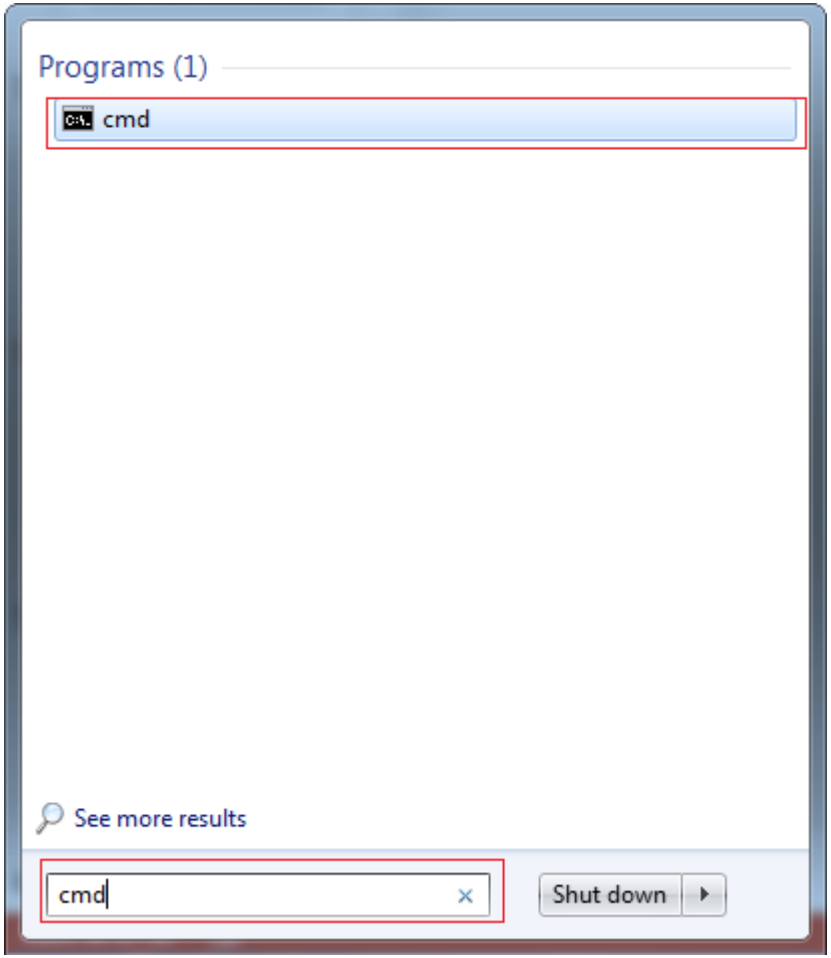
Installation Step-by-Step

- Copy the python installation path;



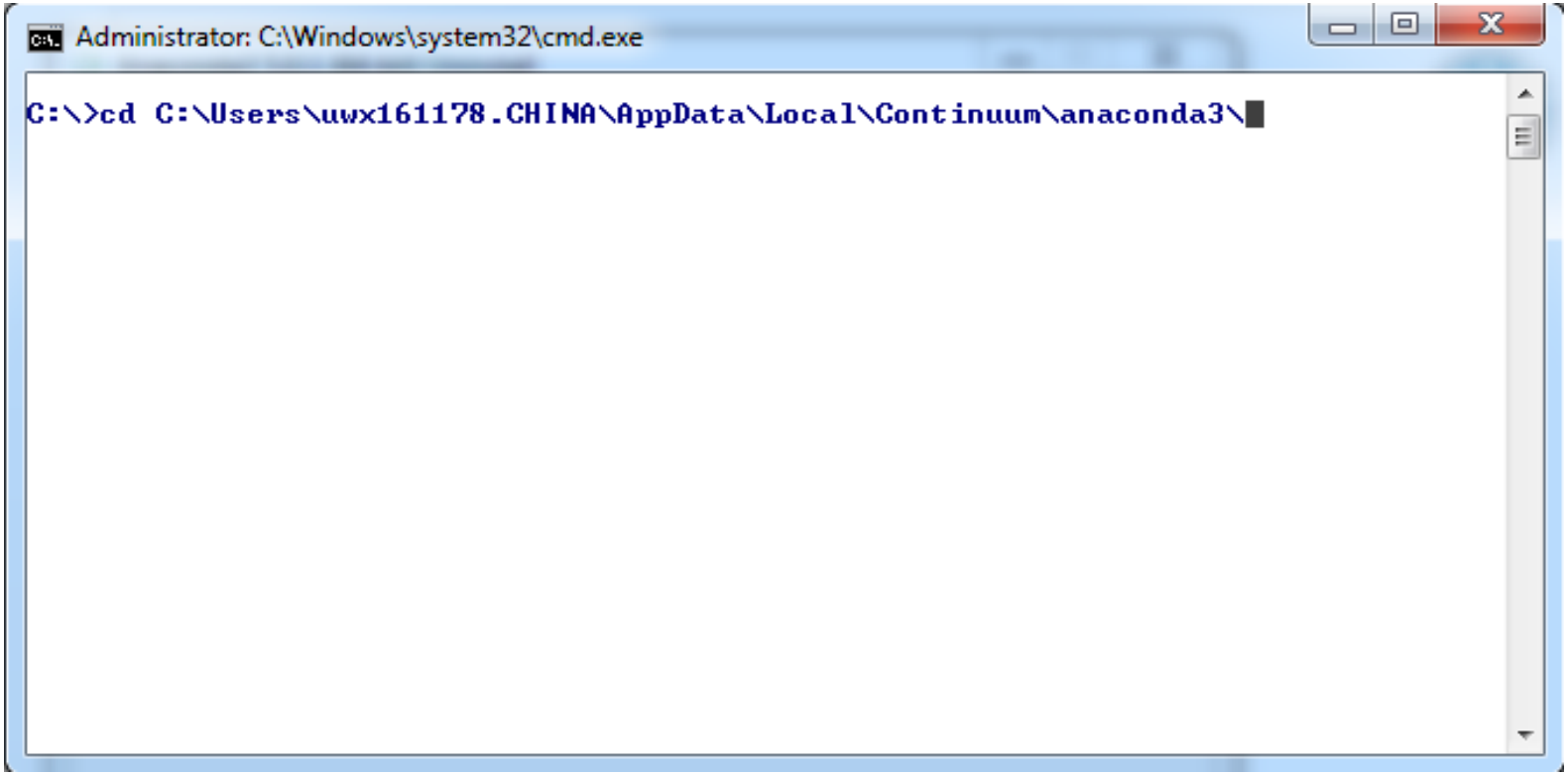
Installation Step-by-Step

- Open cmd.



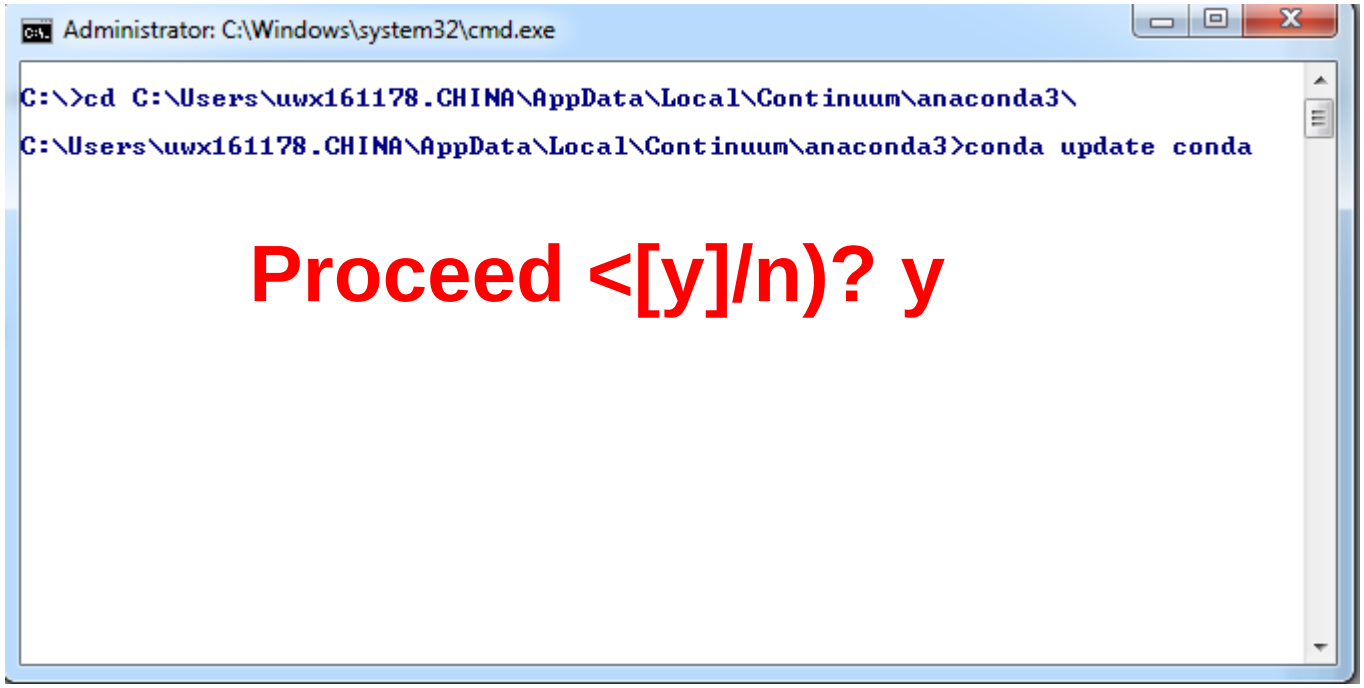
Installation Step-by-Step

- **Command-1**
- Run the following command on cmd;
- cd **C:\Users\uwx161178.CHINA\AppData\Local\Continuum\anaconda3**



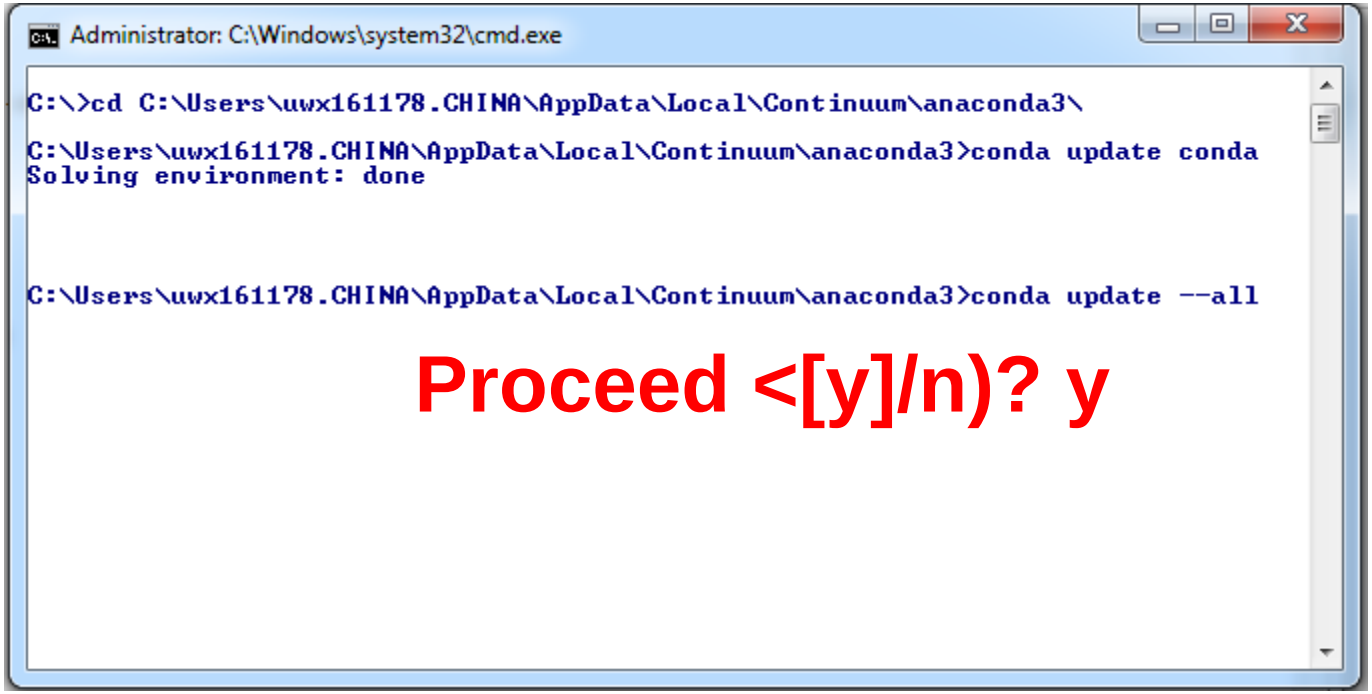
Installation Step-by-Step

- **Command-2**
- Run the following command on cmd;
- **conda update conda**



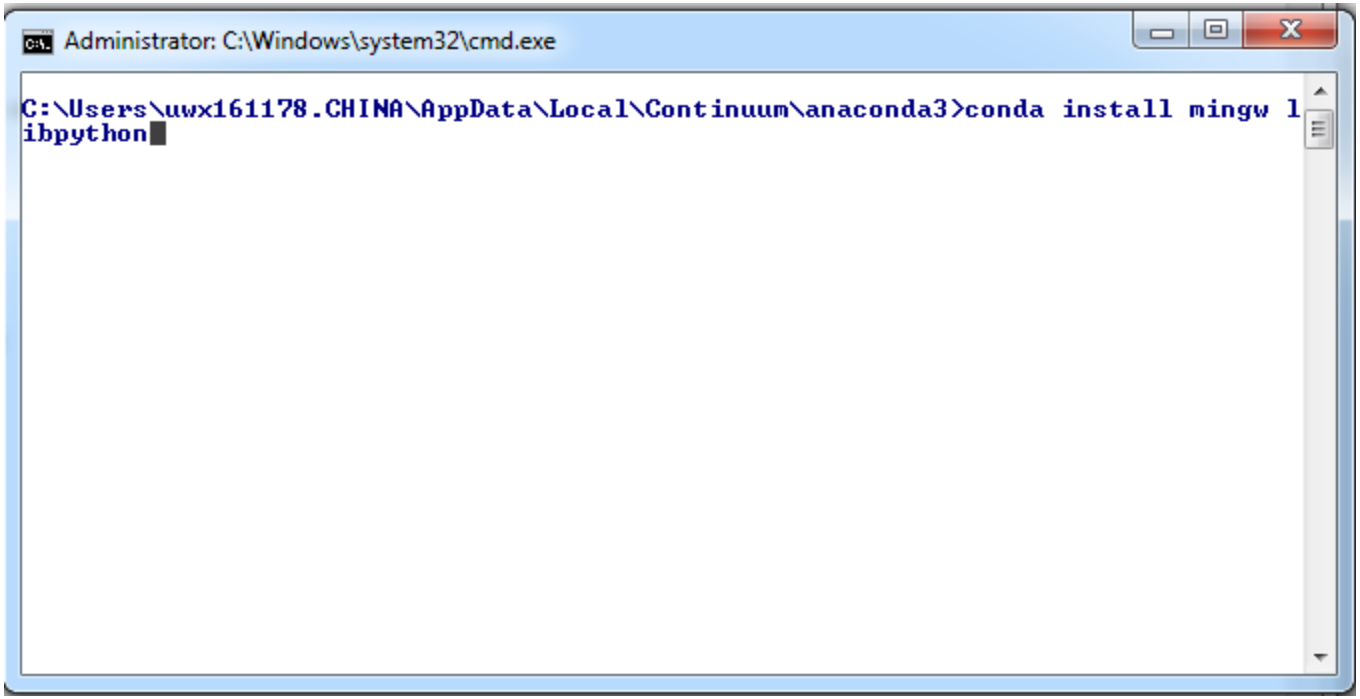
Installation Step-by-Step

- **Command-3**
- Run the following command on cmd;
- **conda update --all**



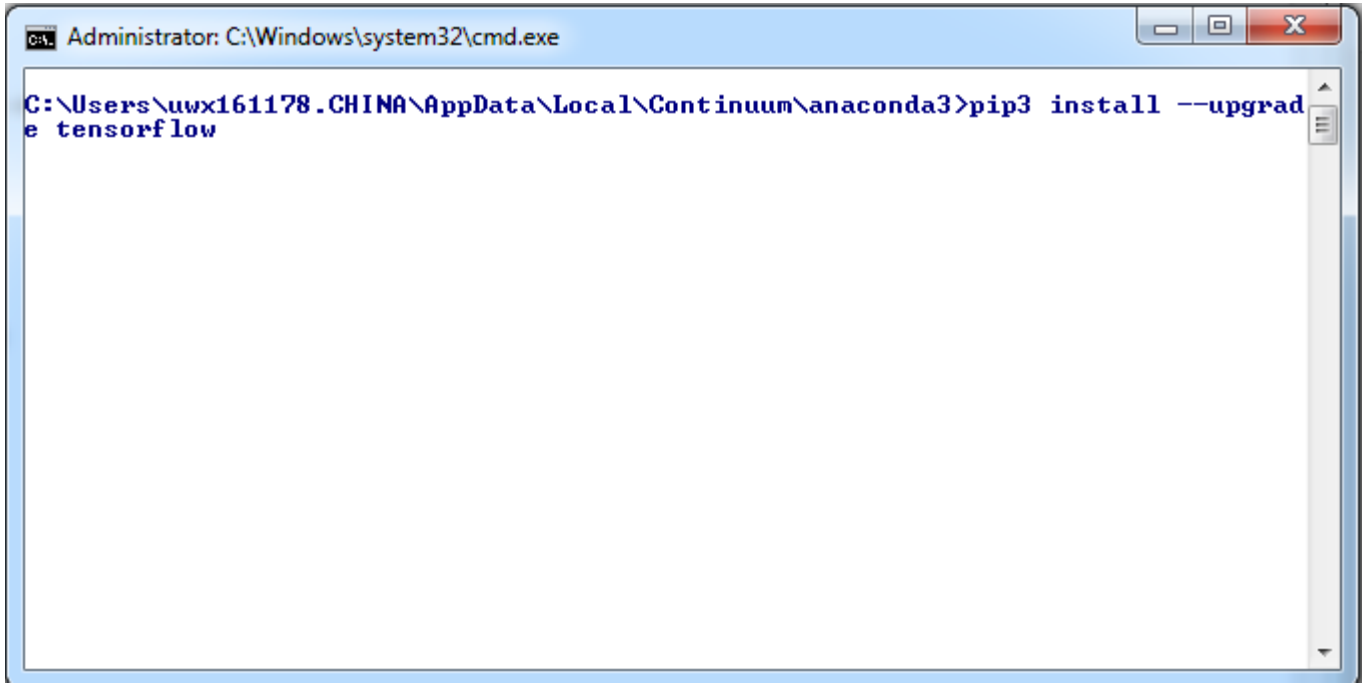
Installation Step-by-Step

- **Command-4**
- Run the following command on cmd;
- **conda install mingw libpython**



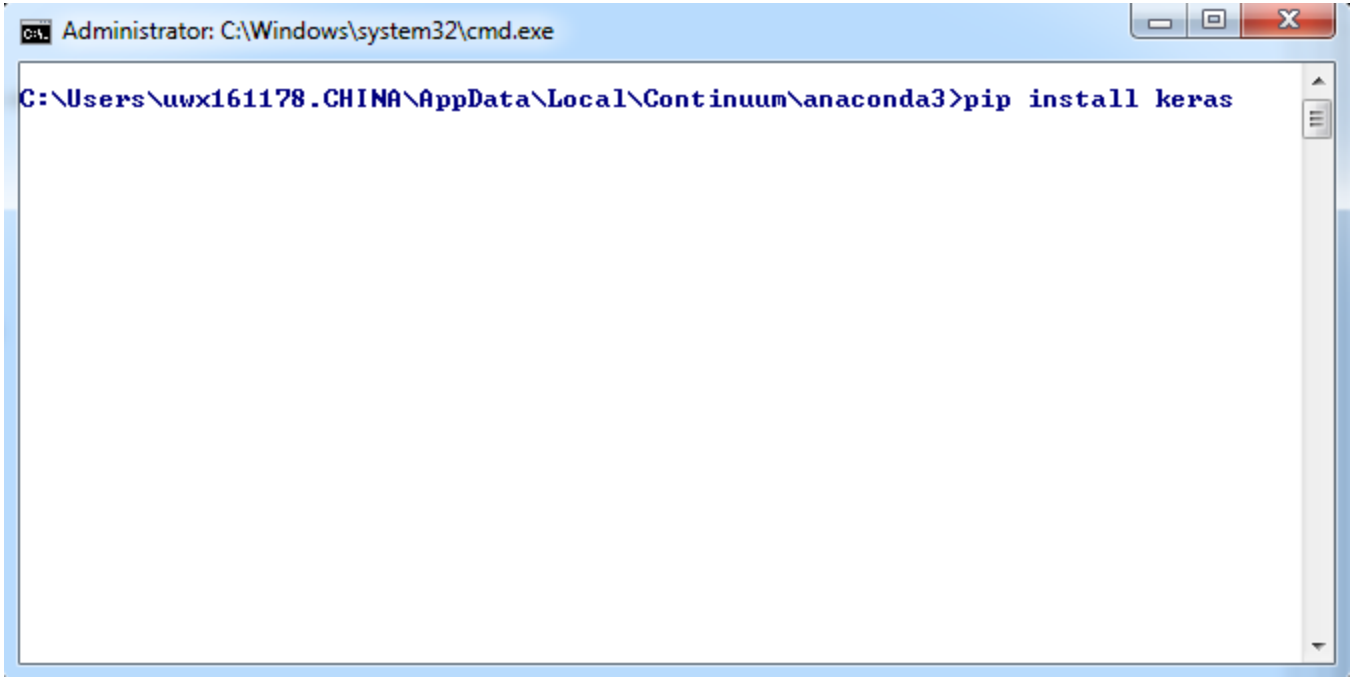
Installation Step-by-Step

- **Command-5**
- Run the following command on cmd;
- **pip3 install --upgrade tensorflow**



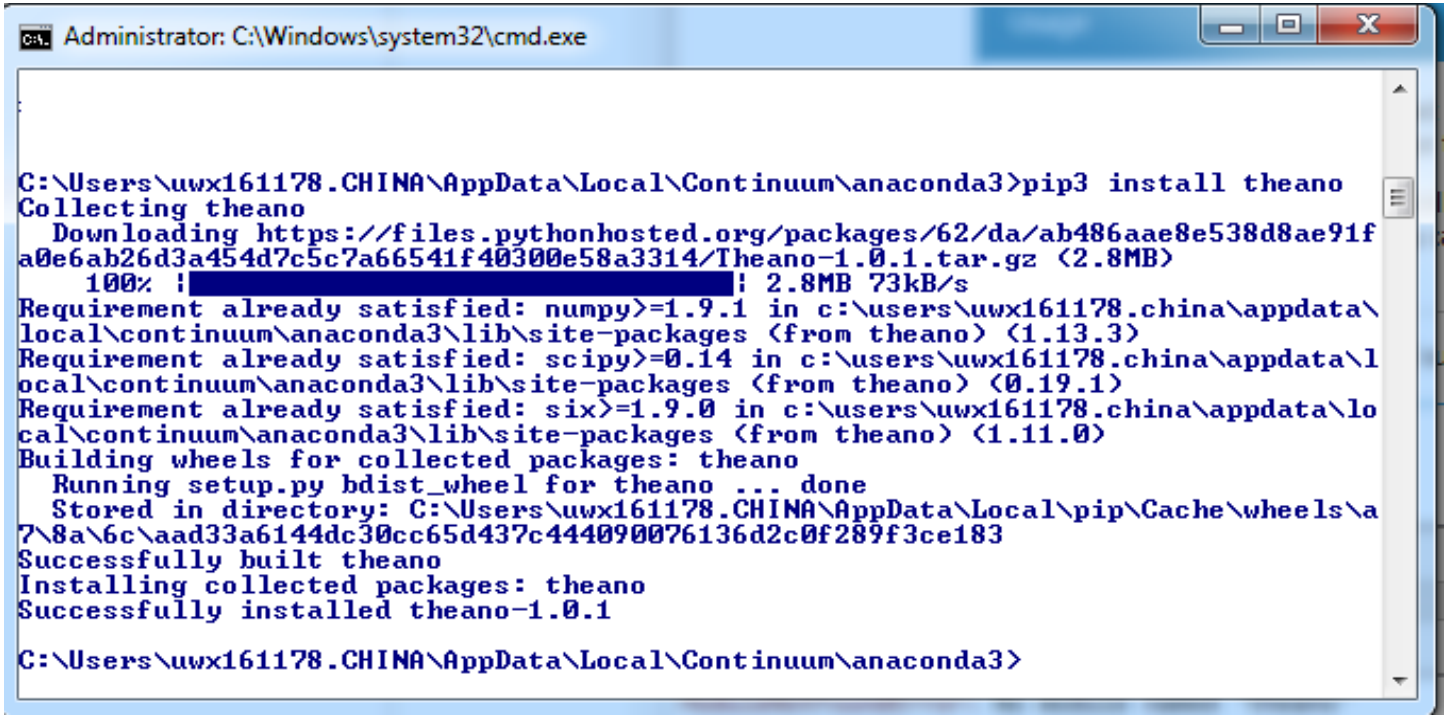
Installation Step-by-Step

- **Command-6**
- Run the following command on cmd;
- **pip install keras**



Installation Step-by-Step

- **Command-7**
- Run the following command on cmd;
- pip install pip3 install theano



Installation Step-by-Step

- Run the following code on Spyder, if no error its mean installation is ok.

```
import theano
import tensorflow as tf

hello = tf.constant('Hello, TensorFlow!')
sess = tf.Session()
print(sess.run(hello))
```

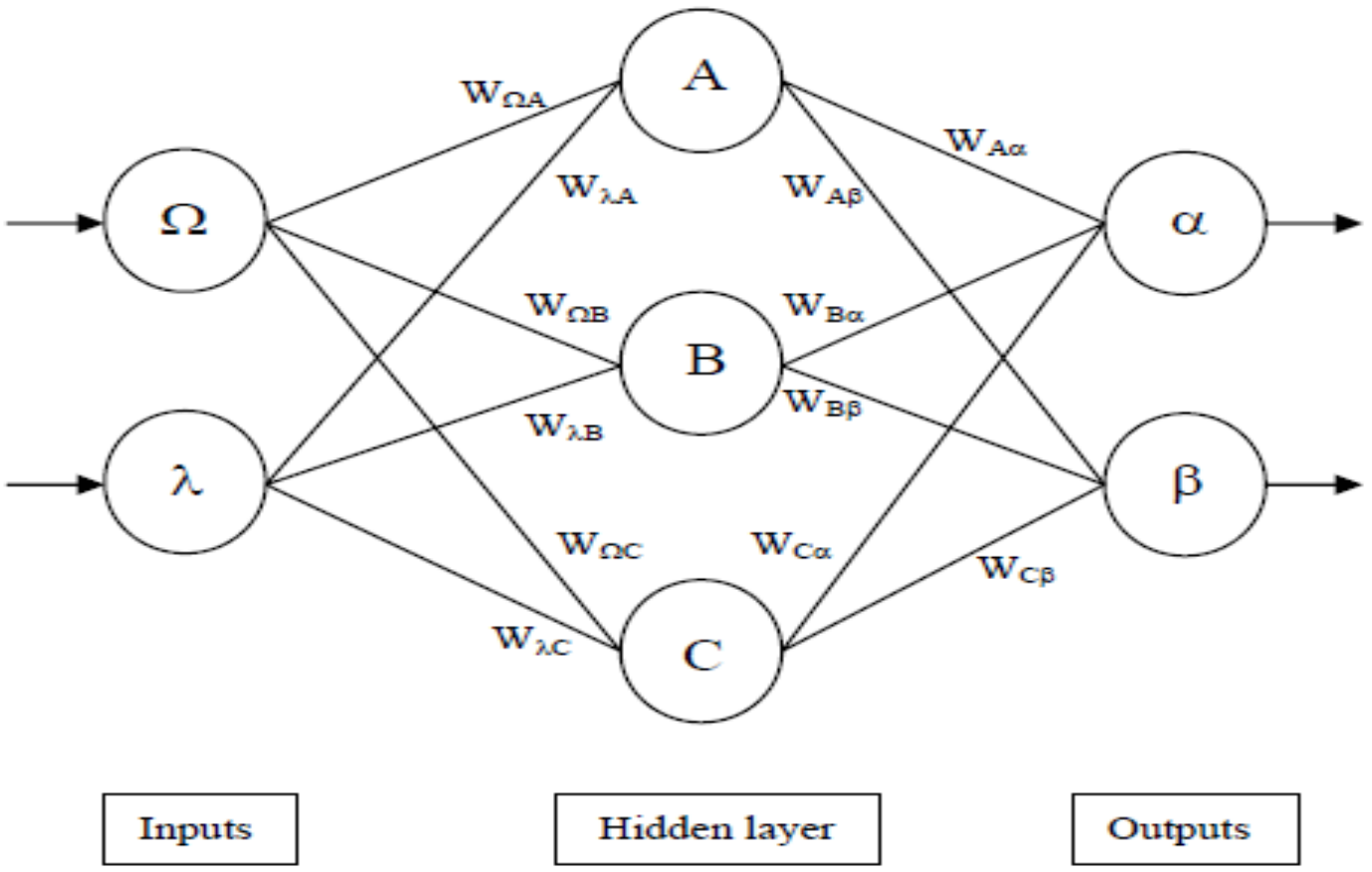
```
In [9]: import theano
   ...: import tensorflow as tf
   ...:
   ...: hello = tf.constant('Hello, TensorFlow!')
   ...: sess = tf.Session()
   ...: print(sess.run(hello))
b'Hello, TensorFlow!'
```

A First Step towards Eye State Prediction Using EEG

Artificial Neural Networks

Artificial Neural Networks

all the calculations for a reverse pass of Back Propagation.



Artificial Neural Networks

1. Calculate errors of output neurons

$$\delta_\alpha = out_\alpha\,(1 - out_\alpha)\,(Target_\alpha - out_\alpha)$$
$$\delta_\beta = out_\beta\,(1 - out_\beta)\,(Target_\beta - out_\beta)$$

2. Change output layer weights

$$W^+_{A\alpha} = W_{A\alpha} + \eta\delta_\alpha\, out_A \qquad W^+_{A\beta} = W_{A\beta} + \eta\delta_\beta\, out_A$$
$$W^+_{B\alpha} = W_{B\alpha} + \eta\delta_\alpha\, out_B \qquad W^+_{B\beta} = W_{B\beta} + \eta\delta_\beta\, out_B$$
$$W^+_{C\alpha} = W_{C\alpha} + \eta\delta_\alpha\, out_C \qquad W^+_{C\beta} = W_{C\beta} + \eta\delta_\beta\, out_C$$

3. Calculate (back-propagate) hidden layer errors

$$\delta_A = out_A\,(1 - out_A)\,(\delta_\alpha W_{A\alpha} + \delta_\beta W_{A\beta})$$
$$\delta_B = out_B\,(1 - out_B)\,(\delta_\alpha W_{B\alpha} + \delta_\beta W_{B\beta})$$
$$\delta_C = out_C\,(1 - out_C)\,(\delta_\alpha W_{C\alpha} + \delta_\beta W_{C\beta})$$

4. Change hidden layer weights

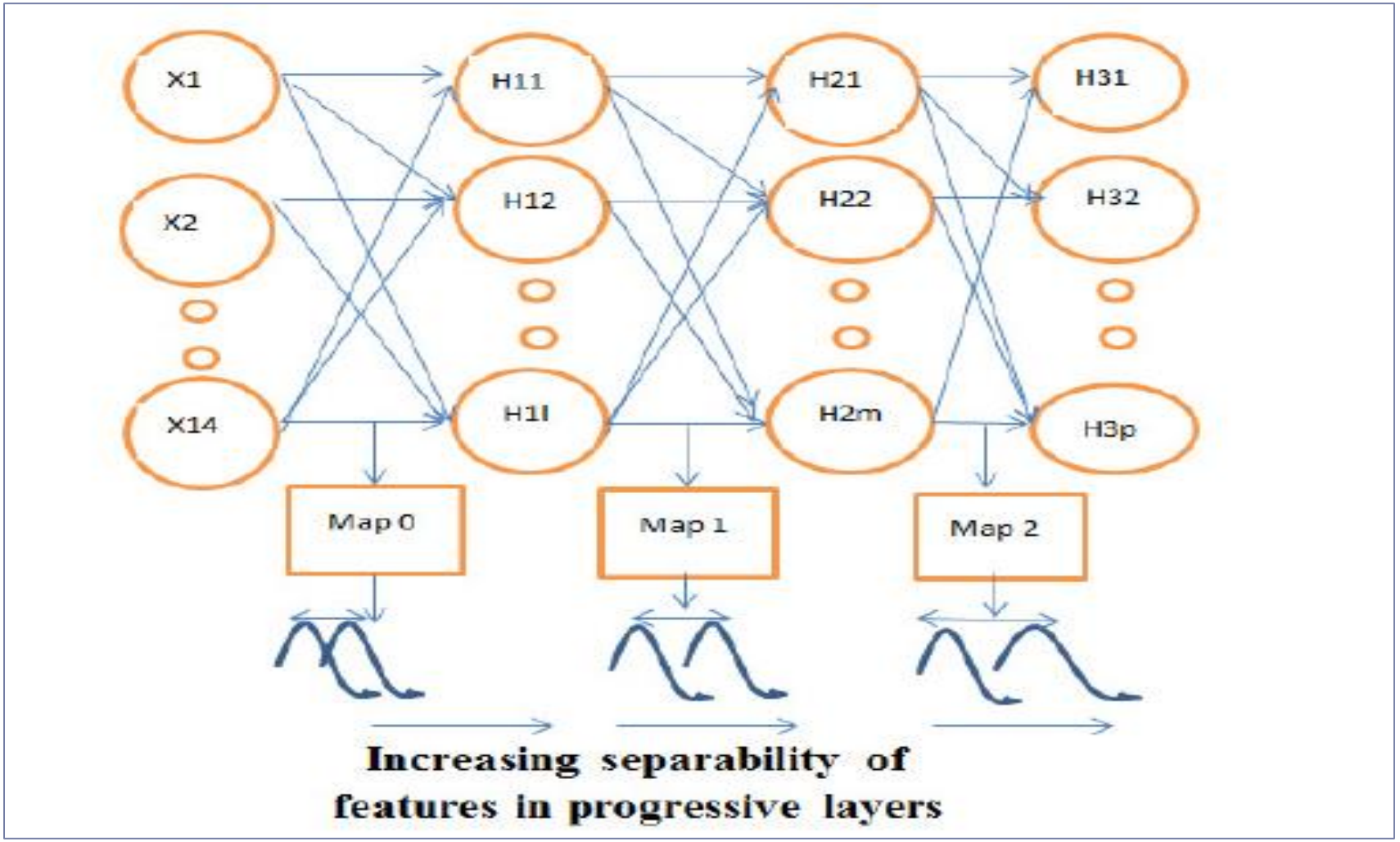
$$W^+_{\lambda A} = W_{\lambda A} + \eta\delta_A\, in_\lambda \qquad W^+_{\Omega A} = W^+_{\Omega A} + \eta\delta_A\, in_\Omega$$
$$W^+_{\lambda B} = W_{\lambda B} + \eta\delta_B\, in_\lambda \qquad W^+_{\Omega B} = W^+_{\Omega B} + \eta\delta_B\, in_\Omega$$
$$W^+_{\lambda C} = W_{\lambda C} + \eta\delta_C\, in_\lambda \qquad W^+_{\Omega C} = W^+_{\Omega C} + \eta\delta_C\, in_\Omega$$

The constant η (called the learning rate, and nominally equal to one) is put in to speed up or slow down the learning if required.

A First Step towards Eye State Prediction Using EEG

10-Artificial Neural Networks(optimizer = 'adam’)

Artificial Neural Networks



Artificial Neural Networks('adam')

- **# Importing the libraries**

```
import theano
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

- **# Importing the Keras libraries and packages**

```
import keras
from keras.models import Sequential
from keras.layers import Dense
```

- **# Importing the dataset**

```
dataset = pd.read_csv('EEG_Eye_State_Arff.csv')
x = dataset.iloc[:, 0:14].values
y = dataset.iloc[:, 14].values
```

Artificial Neural Networks ('adam')

- **# Splitting the dataset into the Training set and Test set**

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 0)
```

- **# Feature Scaling**

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

- **# Initialising the ANN**

```
classifier = Sequential()
```

- **# Adding the input layer and the first hidden layer**

```
classifier.add(Dense(output_dim = 7, init = 'uniform', activation = 'relu', input_dim = 14))
```

Artificial Neural Networks (‘adam’)

- **# Adding the second hidden layer**

```
classifier.add(Dense(output_dim = 7, init = 'uniform', activation = 'relu'))
```

- **# Adding the output layer**

```
classifier.add(Dense(output_dim = 1, init = 'uniform', activation = 'sigmoid'))
```

- **# Compiling the ANN**

```
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

- **# Fitting the ANN to the Training set**

```
classifier.fit(x_train, y_train, batch_size = 15, epochs = 150)
```

Artificial Neural Networks ('adam')

- **# Predicting the Test set results**

```
y_pred = classifier.predict(x_test)
y_pred = (y_pred > 0.5)
```

- **# Making the Confusion Matrix**

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

Artificial Neural Networks ('adam')

Actual class\Predicted class	eye-open=0	eye-closed=1	Total
eye-open=0	**1451(TP)**	**213(FN)**	**1664(P)**
eye-closed=1	**222(FP)**	**1110(TN)**	**1332(N)**
Total	**1712**	**1284**	**2996**

$$Accuracy\ (or\ Recognition\ Rate) = \frac{TP + TN}{P + N} = \frac{2561}{2996} = 0.8548 \approx \mathbf{85.48\%}$$

$$Error\ Rate\ (or\ Misclassification\ Rate) = \frac{FP + FN}{P + N} = \frac{435}{2996} = 0.1452 \approx \mathbf{14.52\%}$$

$$Sensitivity(or\ True\ Positive\ Rate\ or\ Recall) = \frac{TP}{P} = \frac{1451}{1664} = 0.8720 \approx \mathbf{87.20\%}$$

$$Specificity\ (True\ negative\ Rate) = \frac{TN}{N} = \frac{1110}{1332} = 0.8333 \approx \mathbf{83.33\%}$$

$$Precision = \frac{TP}{TP + FP} = \frac{1451}{1673} = 0.8673 \approx \mathbf{86.73\%}$$

$$F - Score = \frac{2 \times precision \times recall}{precision + recall} = \frac{1.5126}{1.7393} = 0.8696 \approx \mathbf{86.96\%}$$

A First Step towards Eye State Prediction Using EEG

11-Artificial Neural Networks(optimizer = ‘sgd’)

Artificial Neural Networks (‘sgd’)

- **# Importing the libraries**

```
import theano
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

- **# Importing the Keras libraries and packages**

```
import keras
from keras.models import Sequential
from keras.layers import Dense
```

- **# Importing the dataset**

```
dataset = pd.read_csv('EEG_Eye_State_Arff.csv')
x = dataset.iloc[:, 0:14].values
y = dataset.iloc[:, 14].values
```

Artificial Neural Networks ('sgd')

- **# Splitting the dataset into the Training set and Test set**

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 0)
```

- **# Feature Scaling**

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

- **# Initialising the ANN**

```
classifier = Sequential()
```

- **# Adding the input layer and the first hidden layer**

```
classifier.add(Dense(output_dim = 7, init = 'uniform', activation = 'relu', input_dim = 14))
```

Artificial Neural Networks (‘sgd’)

- **# Adding the second hidden layer**

```
classifier.add(Dense(output_dim = 7, init = 'uniform', activation = 'relu'))
```

- **# Adding the output layer**

```
classifier.add(Dense(output_dim = 1, init = 'uniform', activation = 'sigmoid'))
```

- **# Compiling the ANN**

```
classifier.compile(optimizer = ‘sgd', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

- **# Fitting the ANN to the Training set**

```
classifier.fit(x_train, y_train, batch_size = 15, epochs = 150)
```

Artificial Neural Networks ('sgd')

- **# Predicting the Test set results**

```
y_pred = classifier.predict(x_test)
y_pred = (y_pred > 0.5)
```

- **# Making the Confusion Matrix**

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

Artificial Neural Networks ('sgd')

Actual class\Predicted class	eye-open=0	eye-closed=1	Total
eye-open=0	**1469(TP)**	**195(FN)**	**1664(P)**
eye-closed=1	**210(FP)**	**1122(TN)**	**1332(N)**
Total	**1679**	**1317**	**2996**

$$Accuracy\ (or\ Recognition\ Rate) = \frac{TP + TN}{P + N} = \frac{2591}{2996} = 0.8648 \approx \mathbf{86.48\%}$$

$$Error\ Rate\ (or\ Misclassification\ Rate) = \frac{FP + FN}{P + N} = \frac{405}{2996} = 0.1352 \approx \mathbf{13.52\%}$$

$$Sensitivity(or\ True\ Positive\ Rate\ or\ Recall) = \frac{TP}{P} = \frac{1469}{1664} = 0.8828 \approx \mathbf{88.28\%}$$

$$Specificity\ (True\ negative\ Rate) = \frac{TN}{N} = \frac{1122}{1332} = 0.8423 \approx \mathbf{84.23\%}$$

$$Precision = \frac{TP}{TP + FP} = \frac{1469}{1679} = 0.8749 \approx \mathbf{87.49\%}$$

$$F - Score = \frac{2 \times precision \times recall}{precision + recall} = \frac{1.5448}{1.7577} = 0.8789 \approx \mathbf{87.89\%}$$

A First Step towards Eye State Prediction Using EEG

12-Artificial Neural Networks(optimizer = ‘Adadelta’)

Artificial Neural Networks (‘Adadelta’)

- **# Importing the libraries**

```
import theano
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

- **# Importing the Keras libraries and packages**

```
import keras
from keras.models import Sequential
from keras.layers import Dense
```

- **# Importing the dataset**

```
dataset = pd.read_csv('EEG_Eye_State_Arff.csv')
x = dataset.iloc[:, 0:14].values
y = dataset.iloc[:, 14].values
```

Artificial Neural Networks('Adadelta')

- **# Splitting the dataset into the Training set and Test set**

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 0)
```

- **# Feature Scaling**

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

- **# Initialising the ANN**

```
classifier = Sequential()
```

- **# Adding the input layer and the first hidden layer**

```
classifier.add(Dense(output_dim = 7, init = 'uniform', activation = 'relu', input_dim = 14))
```

Artificial Neural Networks('Adadelta')

- **# Adding the second hidden layer**

```
classifier.add(Dense(output_dim = 7, init = 'uniform', activation = 'relu'))
```

- **# Adding the output layer**

```
classifier.add(Dense(output_dim = 1, init = 'uniform', activation = 'sigmoid'))
```

- **# Compiling the ANN**

```
classifier.compile(optimizer = 'Adadelta', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

- **# Fitting the ANN to the Training set**

```
classifier.fit(x_train, y_train, batch_size = 15, epochs = 150)
```

Artificial Neural Networks('Adadelta')

- **# Predicting the Test set results**

```
y_pred = classifier.predict(x_test)
y_pred = (y_pred > 0.5)
```

- **# Making the Confusion Matrix**

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

Artificial Neural Networks ('Adadelta')

Actual class\Predicted class	eye-open=0	eye-closed=1	Total
eye-open=0	**1472(TP)**	**192(FN)**	**1664(P)**
eye-closed=1	**234(FP)**	**1098(TN)**	**1332(N)**
Total	**1706**	**1290**	**2996**

$$Accuracy\ (or\ Recognition\ Rate) = \frac{TP + TN}{P + N} = \frac{2570}{2996} = 0.8578 \approx \mathbf{85.78\%}$$

$$Error\ Rate\ (or\ Misclassification\ Rate) = \frac{FP + FN}{P + N} = \frac{426}{2996} = 0.1422 \approx \mathbf{14.22\%}$$

$$Sensitivity(or\ True\ Positive\ Rate\ or\ Recall) = \frac{TP}{P} = \frac{1472}{1664} = 0.8846 \approx \mathbf{88.46\%}$$

$$Specificity\ (True\ negative\ Rate) = \frac{TN}{N} = \frac{1098}{1332} = 0.8243 \approx \mathbf{82.43\%}$$

$$Precision = \frac{TP}{TP + FP} = \frac{1472}{1706} = 0.8628 \approx \mathbf{86.28\%}$$

$$F - Score = \frac{2 \times precision \times recall}{precision + recall} = \frac{1.5266}{1.7475} = 0.8736 \approx \mathbf{87.36\%}$$

A First Step towards Eye State Prediction Using EEG

13-Artificial Neural Networks(optimizer = ‘Adamax’)

Artificial Neural Networks('Adamax')

- **# Importing the libraries**

```
import theano
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

- **# Importing the Keras libraries and packages**

```
import keras
from keras.models import Sequential
from keras.layers import Dense
```

- **# Importing the dataset**

```
dataset = pd.read_csv('EEG_Eye_State_Arff.csv')
x = dataset.iloc[:, 0:14].values
y = dataset.iloc[:, 14].values
```

Artificial Neural Networks('Adamax')

- **# Splitting the dataset into the Training set and Test set**

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 0)
```

- **# Feature Scaling**

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

- **# Initialising the ANN**

```
classifier = Sequential()
```

- **# Adding the input layer and the first hidden layer**

```
classifier.add(Dense(output_dim = 7, init = 'uniform', activation = 'relu', input_dim = 14))
```

Artificial Neural Networks(‘Adamax’)

- **# Adding the second hidden layer**

```
classifier.add(Dense(output_dim = 7, init = 'uniform', activation = 'relu'))
```

- **# Adding the output layer**

```
classifier.add(Dense(output_dim = 1, init = 'uniform', activation = 'sigmoid'))
```

- **# Compiling the ANN**

```
classifier.compile(optimizer = ‘Adamax', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

- **# Fitting the ANN to the Training set**

```
classifier.fit(x_train, y_train, batch_size = 15, epochs = 150)
```

Artificial Neural Networks('Adamax')

- **# Predicting the Test set results**

```
y_pred = classifier.predict(x_test)
y_pred = (y_pred > 0.5)
```

- **# Making the Confusion Matrix**

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

Artificial Neural Networks('Adamax')

Actual class\Predicted class	eye-open=0	eye-closed=1	Total
eye-open=0	**1482(TP)**	**182(FN)**	**1664(P)**
eye-closed=1	**285(FP)**	**1047(TN)**	**1332(N)**
Total	**1767**	**1229**	**2996**

$$Accuracy\ (or\ Recognition\ Rate) = \frac{TP + TN}{P + N} = \frac{2529}{2996} = 0.8441 \approx \mathbf{84.41\%}$$

$$Error\ Rate\ (or\ Misclassification\ Rate) = \frac{FP + FN}{P + N} = \frac{467}{2996} = 0.1559 \approx \mathbf{15.59\%}$$

$$Sensitivity(or\ True\ Positive\ Rate\ or\ Recall) = \frac{TP}{P} = \frac{1482}{1664} = 0.8906 \approx \mathbf{89.06\%}$$

$$Specificity\ (True\ negative\ Rate) = \frac{TN}{N} = \frac{1047}{1332} = 0.7860 \approx \mathbf{78.60\%}$$

$$Precision = \frac{TP}{TP + FP} = \frac{1482}{1767} = 0.8387 \approx \mathbf{83.87\%}$$

$$F - Score = \frac{2 \times precision \times recall}{precision + recall} = \frac{1.4940}{1.7293} = 0.8639 \approx \mathbf{86.39\%}$$

A First Step towards Eye State Prediction Using EEG

14-Artificial Neural Networks(optimizer = ‘RMSprop’)

Artificial Neural Networks('RMSprop')

- **# Importing the libraries**

```
import theano
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

- **# Importing the Keras libraries and packages**

```
import keras
from keras.models import Sequential
from keras.layers import Dense
```

- **# Importing the dataset**

```
dataset = pd.read_csv('EEG_Eye_State_Arff.csv')
x = dataset.iloc[:, 0:14].values
y = dataset.iloc[:, 14].values
```

Artificial Neural Networks ('RMSprop')

- **# Splitting the dataset into the Training set and Test set**

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 0)
```

- **# Feature Scaling**

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

- **# Initialising the ANN**

```
classifier = Sequential()
```

- **# Adding the input layer and the first hidden layer**

```
classifier.add(Dense(output_dim = 7, init = 'uniform', activation = 'relu', input_dim = 14))
```

Artificial Neural Networks (‘RMSprop’)

- **# Adding the second hidden layer**

```
classifier.add(Dense(output_dim = 7, init = 'uniform', activation = 'relu'))
```

- **# Adding the output layer**

```
classifier.add(Dense(output_dim = 1, init = 'uniform', activation = 'sigmoid'))
```

- **# Compiling the ANN**

```
classifier.compile(optimizer = ‘RMSprop', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

- **# Fitting the ANN to the Training set**

```
classifier.fit(x_train, y_train, batch_size = 15, epochs = 150)
```

Artificial Neural Networks ('RMSprop')

- **# Predicting the Test set results**

```
y_pred = classifier.predict(x_test)
y_pred = (y_pred > 0.5)
```

- **# Making the Confusion Matrix**

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

Artificial Neural Networks ('RMSprop')

Actual class\Predicted class	eye-open=0	eye-closed=1	Total
eye-open=0	**1507(TP)**	**157(FN)**	**1664(P)**
eye-closed=1	**255(FP)**	**1077(TN)**	**1332(N)**
Total	**1762**	**1234**	**2996**

$$Accuracy\ (or\ Recognition\ Rate) = \frac{TP + TN}{P + N} = \frac{2584}{2996} = 0.8625 \approx \mathbf{86.25\%}$$

$$Error\ Rate\ (or\ Misclassification\ Rate) = \frac{FP + FN}{P + N} = \frac{412}{2996} = 0.1375 \approx \mathbf{13.75\%}$$

$$Sensitivity(or\ True\ Positive\ Rate\ or\ Recall) = \frac{TP}{P} = \frac{1507}{1664} = 0.9056 \approx \mathbf{90.56\%}$$

$$Specificity\ (True\ negative\ Rate) = \frac{TN}{N} = \frac{1077}{1332} = 0.8086 \approx \mathbf{80.86\%}$$

$$Precision = \frac{TP}{TP + FP} = \frac{1507}{1762} = 0.8553 \approx \mathbf{85.53\%}$$

$$F - Score = \frac{2 \times precision \times recall}{precision + recall} = \frac{1.5492}{1.7609} = 0.8797 \approx \mathbf{87.97\%}$$

A First Step towards Eye State Prediction Using EEG

15-Artificial Neural Networks(optimizer = ‘Adagrad’)

Artificial Neural Networks (‘Adagrad’)

- **# Importing the libraries**

```
import theano
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

- **# Importing the Keras libraries and packages**

```
import keras
from keras.models import Sequential
from keras.layers import Dense
```

- **# Importing the dataset**

```
dataset = pd.read_csv('EEG_Eye_State_Arff.csv')
x = dataset.iloc[:, 0:14].values
y = dataset.iloc[:, 14].values
```

Artificial Neural Networks ('Adagrad')

- **# Splitting the dataset into the Training set and Test set**

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 0)
```

- **# Feature Scaling**

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

- **# Initialising the ANN**

```
classifier = Sequential()
```

- **# Adding the input layer and the first hidden layer**

```
classifier.add(Dense(output_dim = 7, init = 'uniform', activation = 'relu', input_dim = 14))
```

Artificial Neural Networks (‘Adagrad’)

- **# Adding the second hidden layer**

```
classifier.add(Dense(output_dim = 7, init = 'uniform', activation = 'relu'))
```

- **# Adding the output layer**

```
classifier.add(Dense(output_dim = 1, init = 'uniform', activation = 'sigmoid'))
```

- **# Compiling the ANN**

```
classifier.compile(optimizer = ‘Adagrad', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

- **# Fitting the ANN to the Training set**

```
classifier.fit(x_train, y_train, batch_size = 15, epochs = 150)
```

Artificial Neural Networks (‘Adagrad’)

- **# Predicting the Test set results**

```
y_pred = classifier.predict(x_test)
y_pred = (y_pred > 0.5)
```

- **# Making the Confusion Matrix**

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

Artificial Neural Networks ('Adagrad')

Actual class\Predicted class	eye-open=0	eye-closed=1	Total
eye-open=0	**1417(TP)**	**247(FN)**	**1664(P)**
eye-closed=1	**366(FP)**	**966(TN)**	**1332(N)**
Total	**1783**	**1213**	**2996**

$$Accuracy\ (or\ Recognition\ Rate) = \frac{TP + TN}{P + N} = \frac{2383}{2996} = 0.7954 \approx \mathbf{85.48\%}$$

$$Error\ Rate\ (or\ Misclassification\ Rate) = \frac{FP + FN}{P + N} = \frac{613}{2996} = 0.2046 \approx \mathbf{14.52\%}$$

$$Sensitivity(or\ True\ Positive\ Rate\ or\ Recall) = \frac{TP}{P} = \frac{1417}{1664} = 0.8516 \approx \mathbf{85.16\%}$$

$$Specificity\ (True\ negative\ Rate) = \frac{TN}{N} = \frac{966}{1332} = 0.7252 \approx \mathbf{72.52\%}$$

$$Precision = \frac{TP}{TP + FP} = \frac{1417}{1783} = 0.7947 \approx \mathbf{79.47\%}$$

$$F - Score = \frac{2 \times precision \times recall}{precision + recall} = \frac{1.3535}{1.6463} = 0.8222 \approx \mathbf{82.22\%}$$

A First Step towards Eye State Prediction Using EEG

16-Artificial Neural Networks(optimizer = ‘Nadam’)

Artificial Neural Networks (‘Nadam’)

- **# Importing the libraries**

```
import theano
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

- **# Importing the Keras libraries and packages**

```
import keras
from keras.models import Sequential
from keras.layers import Dense
```

- **# Importing the dataset**

```
dataset = pd.read_csv('EEG_Eye_State_Arff.csv')
x = dataset.iloc[:, 0:14].values
y = dataset.iloc[:, 14].values
```

Artificial Neural Networks ('Nadam')

- **# Splitting the dataset into the Training set and Test set**

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 0)
```

- **# Feature Scaling**

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

- **# Initialising the ANN**

```
classifier = Sequential()
```

- **# Adding the input layer and the first hidden layer**

```
classifier.add(Dense(output_dim = 7, init = 'uniform', activation = 'relu', input_dim = 14))
```

Artificial Neural Networks (‘Nadam’)

- **# Adding the second hidden layer**

```
classifier.add(Dense(output_dim = 7, init = 'uniform', activation = 'relu'))
```

- **# Adding the output layer**

```
classifier.add(Dense(output_dim = 1, init = 'uniform', activation = 'sigmoid'))
```

- **# Compiling the ANN**

```
classifier.compile(optimizer = ‘Nadam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

- **# Fitting the ANN to the Training set**

```
classifier.fit(x_train, y_train, batch_size = 15, epochs = 150)
```

Artificial Neural Networks ('Nadam')

- **# Predicting the Test set results**

```
y_pred = classifier.predict(x_test)
y_pred = (y_pred > 0.5)
```

- **# Making the Confusion Matrix**

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

Artificial Neural Networks ('Nadam')

Actual class\Predicted class	eye-open=0	eye-closed=1	Total
eye-open=0	**1483(TP)**	**181(FN)**	**1664(P)**
eye-closed=1	**206(FP)**	**1126(TN)**	**1332(N)**
Total	**1689**	**1307**	**2996**

$$Accuracy\ (or\ Recognition\ Rate) = \frac{TP + TN}{P + N} = \frac{2609}{2996} = 0.8708 \approx \mathbf{87.08\%}$$

$$Error\ Rate\ (or\ Misclassification\ Rate) = \frac{FP + FN}{P + N} = \frac{387}{2996} = 0.1292 \approx \mathbf{12.92\%}$$

$$Sensitivity(or\ True\ Positive\ Rate\ or\ Recall) = \frac{TP}{P} = \frac{1483}{1664} = 0.8912 \approx \mathbf{89.12\%}$$

$$Specificity\ (True\ negative\ Rate) = \frac{TN}{N} = \frac{1126}{1332} = 0.8453 \approx \mathbf{84.53\%}$$

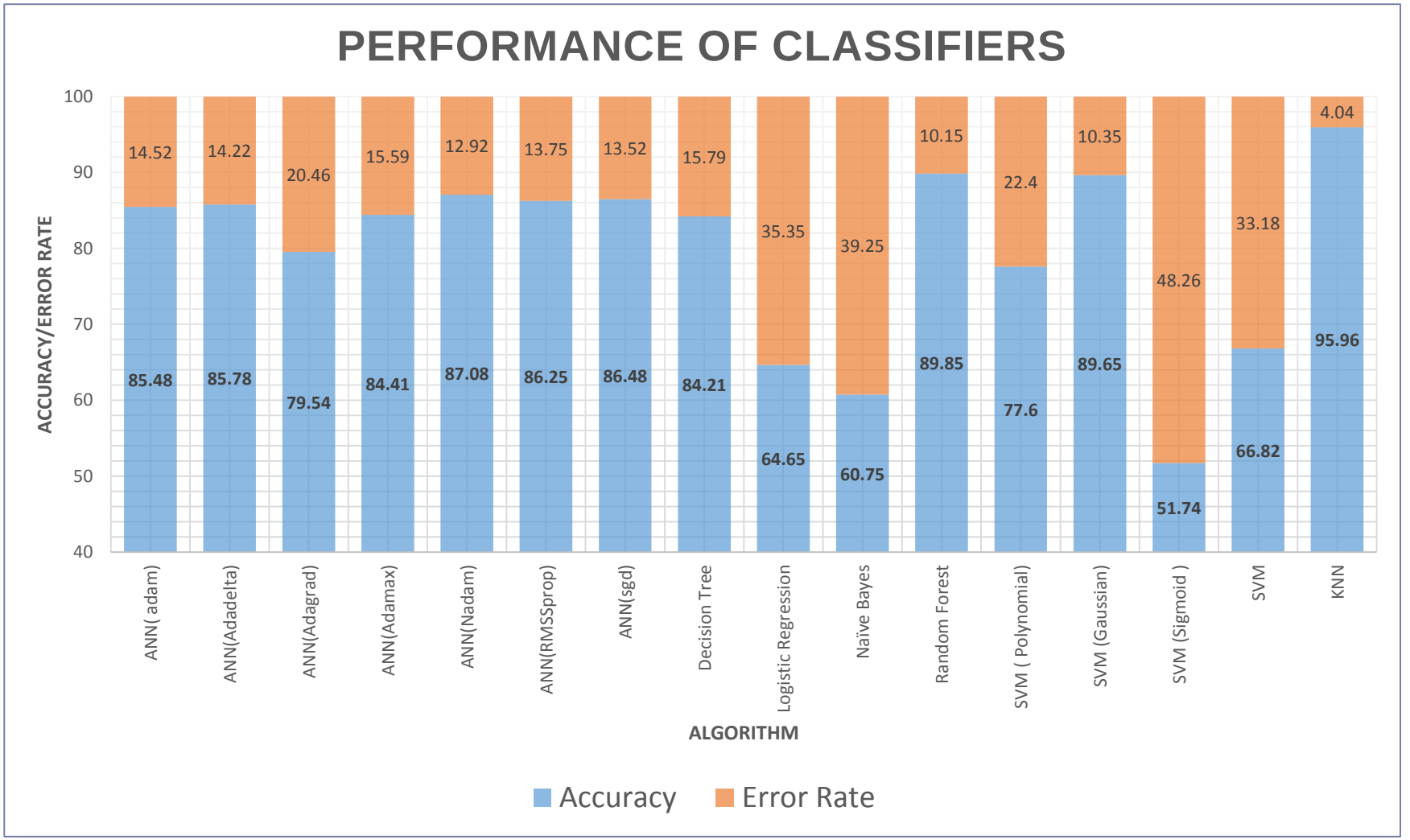
$$Precision = \frac{TP}{TP + FP} = \frac{1483}{1689} = 0.8780 \approx \mathbf{87.80\%}$$

$$F - Score = \frac{2 \times precision \times recall}{precision + recall} = \frac{1.5651}{1.7693} = 0.8846 \approx \mathbf{88.46\%}$$

Performance of the Classifiers

Sr No	Algorithm	Accuracy	Error Rate	Sensitivity	Specificity	Precision	F-Score
1	Logistic Regression	64.65	35.35	77.22	48.95	65.39	70.82
2	KNN	95.96	4.04	96.75	94.97	96.00	96.38
3	SVM	66.82	33.18	82.33	47.45	66.18	73.38
4	SVM (Gaussian)	89.65	10.35	92.73	85.81	89.09	90.87
5	SVM (Polynomial)	77.60	22.40	98.38	51.65	71.77	82.99
6	SVM (Sigmoid)	51.74	48.26	54.69	48.05	56.80	55.73
7	Naïve Bayes	60.75	39.25	58.05	64.11	66.90	62.16
8	Decision Tree	84.21	15.79	84.98	83.26	86.38	85.67
9	Random Forest	89.85	10.15	94.47	84.08	88.12	91.18
10	ANN(adam)	85.48	14.52	87.20	83.33	86.73	86.96
11	ANN(sgd)	86.48	13.52	88.28	84.23	87.49	87.89
12	ANN(Adadelta)	85.78	14.22	88.46	82.43	86.28	87.36
13	ANN(Adamax)	84.41	15.59	89.06	78.60	83.87	86.39
14	ANN(RMSSprop)	86.25	13.75	90.56	80.86	85.53	87.97
15	ANN(Adagrad)	79.54	20.46	85.16	72.52	79.47	82.22
16	ANN(Nadam)	87.08	12.92	89.12	84.53	87.80	88.46

Performance of the Classifiers



Performance of the Classifiers

- K Nearest Neighbors (K-NN) achieved the clearly best performance with a classification error rate of merely 4.04%.
- Decision tree , ANN(Adamax), ANN(adam), ANN(Adadelta), ANN(sgd), ANN(RMSSprop), ANN(Nadam), SVM (Gaussian) and Random Forest performed much better (about 10%-15% classification error)
- Standard classifiers such as na¨ıve Bayes, logistic regression with a proven track of high classification performance produced rather poor results on this task (over 35% classification error)

Reference

- 1- P. Pour, T. Gulrez, O. AlZoubi, G. Gargiulo, and R. Calvo, “Brain-Computer Interface: Next Generation Thought Controlled Distributed Video Game Development Platform,” in Proc: of the CIG, Perth, Australia, 2008.
- 2- T. Pham and D. Tran, “Emotion recognition using the emotiv epoc device,” Lecture Notes in Computer Science, vol. 7667, 2012
- 3-O. Ossmy, O. Tam, R. Puzis, L. Rokach, O. Inbar, and Y. Elovici, “MindDesktop - Computer Accessibility for Severely Handicapped,” in Proc. of the ICEIS, Beijing, China, 2011.
- 4- J. van Erp, S. Reschke, M. Grootjen, and A.-M. Brouwer, “Brain Performance Enhancement for Military Operators,” in Proc. of the HFM, Sofia, Bulgaria, 2009.
- 5- L. Li, L. Xiao, and L. Chen, “Differences of EEG between Eyes-Open and Eyes-Closed States Based on Autoregressive Method,” Journal Of Electronic Science And Technology Of China,, vol. 7, no. 2, 2009.
- 6- B. Chambayil, R. Singla, and R. Jha, “EEG Eye Blink Classification Using Neural Network,” in Proc: of the World Congress on Engineering, London, UK, 2010.
- 7- R. Thuraisingham, Y. Tran, P. Boord, and A. Craig, “Analysis of Eyes Open, Eye Closed EEG Signals Using Second-Order Difference Plot,” Medical and Biological Engineering and Computing, vol. 45, no. 12, 2007.

Reference

- 8- http://archive.ics.uci.edu/ml/datasets/EEG+Eye+State
- 9- http://scikit-learn.org/stable/
- 10- https://github.com/jaysonfrancis/machinelearning
- 11- https://data-flair.training/blogs/svm-kernel-functions/
- 12- https://www.youtube.com/watch?v=HcW0DeWRggs
- 13- https://data-flair.training/blogs/svm-kernel-functions/
- 14- http://www.statsoft.com/Textbook/Support-Vector-Machines