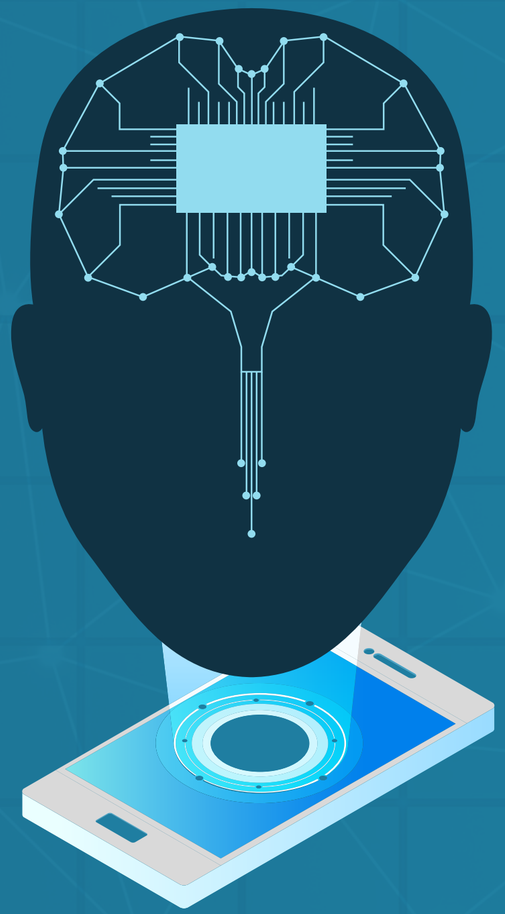
GENDER CLASSIFICATION

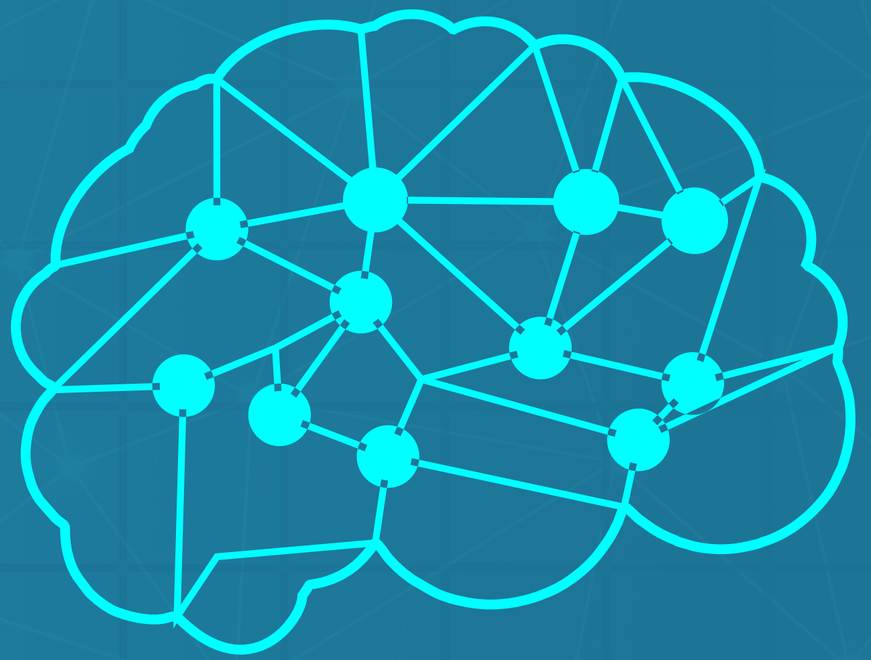


PRESENTED BY:

SULTANH ALJUAID - 1910983
JURI NAJJAR - 1875890
SAMA ALMEHMADI - 1915286
SUHA ALRAJHI - 1912005

INTRODUCTION

Our aim for this project is to use and apply what we learned in this course and prove the practices developed by combining different Machine Learning algorithms to analyze our data.



TASKS

01 Read the dataset.

02 Explore the dataset.

03 Modify the data.

04 Normalize the data.

05 Split the dataset.

06 ML algorithms.

07 Data tuning to improve the results.

08 Results comparison.

01 READ THE DATASET

- Importing libraries
- Reading (gender_classification) dataset

```
[ ]  # import lib
     import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     %matplotlib inline
```

Double-click (or enter) to edit

Read dataset

```
[ ]  # read the  data set
     data = pd.read_csv("gender_classification_v7.csv")
     data
```

	long_hair	forehead_width_cm	forehead_height_cm	nose_wide	nose_long	lips_thin	distance_nose_to_lip_long	gender
0	1	11.8	6.1	1	0	1	1	Male
1	0	14.0	5.4	0	0	1	0	Female
2	0	11.8	6.3	1	1	1	1	Male
3	0	14.4	6.1	0	1	1	1	Male
4	1	13.5	5.9	0	0	0	0	Female
...	...	...	...	...	...	...	...	...
4996	1	13.6	5.1	0	0	0	0	Female
4997	1	11.9	5.4	0	0	0	0	Female
4998	1	12.9	5.7	0	0	0	0	Female

02 EXPLORE THE DATASET

Exploring the data's information:

- Number and names of features.
- Data types

Explore the data set

```
# Explore the dataset
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5001 entries, 0 to 5000
Data columns (total 8 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   long_hair                  5001 non-null   int64
 1   forehead_width_cm          5001 non-null   float64
 2   forehead_height_cm         5001 non-null   float64
 3   nose_wide                  5001 non-null   int64
 4   nose_long                  5001 non-null   int64
 5   lips_thin                  5001 non-null   int64
 6   distance_nose_to_lip_long  5001 non-null   int64
 7   gender                     5001 non-null   object
dtypes: float64(2), int64(5), object(1)
memory usage: 312.7+ KB
```

02 EXPLORE THE DATASET

```
[ ]  # explore the number of column and rows
     data.shape
```

```
(5001, 8)
```

Checking the data shape of column and row numbers

```
[ ]  #check if there missing data
     data.isnull().sum()
```

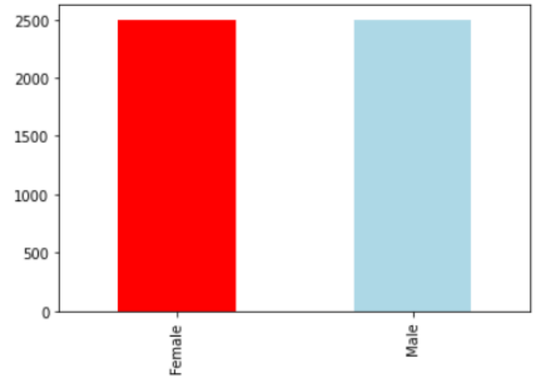
```
long_hair                    0
forehead_width_cm            0
forehead_height_cm           0
nose_wide                    0
nose_long                    0
lips_thin                    0
distance_nose_to_lip_long    0
gender                       0
dtype: int64
```

Check if any of our data is missing/null

02 EXPLORE THE DATASET

- Plotting the numbers of each gender (female - male) to be visually understandable.
- Plotting one of our data's features (Forehead_width_cm)

```
[ ]  data.gender.value_counts().plot(kind="bar", color=["red", "lightblue"]);
```




```
[ ]  plt.scatter(data.gender,data.forehead_width_cm,marker='+',color='red')
     plt.ylabel("forehead_width_cm")
```

```
Text(0, 0.5, 'forehead_width_cm')
```

02 EXPLORE THE DATASET

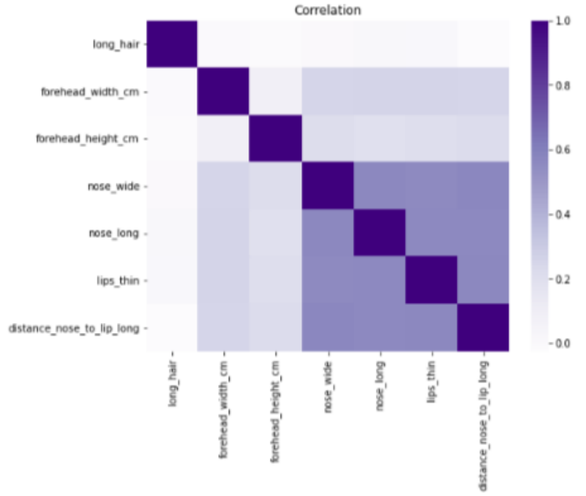
- Plotting one of our data's features (Forehead_height_cm)
- Correlation to visualize our data's best feature (long_hair)

```
[ ] plt.scatter(data.gender,data.forehead_height_cm,marker='+',color='red')
    plt.ylabel("forehead_height_cm")
```

```
Text(0, 0.5, 'forehead_height_cm')
```


```
[ ] import seaborn as sns
    corr= data.corr()
    fig = plt.figure(figsize=(8,6))
    r = sns.heatmap(corr,cmap='Purples')
    r.set_title("Correlation")
```

```
Text(0.5, 1.0, 'Correlation')
```



03 MODIFY THE DATA

```
[ ]  #tranform gendr to 0 ,1  male =1 , female = 0
     from sklearn.preprocessing import LabelEncoder
     gender = LabelEncoder()
     data['gender'] = gender.fit_transform(data['gender'])
     data.head()
```

	long_hair	forehead_width_cm	forehead_height_cm	nose_wide	nose_long	lips_thin	distance_nose_to_lip_long	gender
0	1	11.8	6.1	1	0	1	1	1
1	0	14.0	5.4	0	0	1	0	0
2	0	11.8	6.3	1	1	1	1	1
3	0	14.4	6.1	0	1	1	1	1
4	1	13.5	5.9	0	0	0	0	0

Transforming (gender) labels to numbers (0 = female | 1 = male) so its easier for the machine to be trained

04 NORMALIZE THE DATA

Normalizing the data using the Min-Max Scaler so that values are shifted and rescaled ranging from 0 to 1.

```
#normlize the featurs
x = (x-np.min(x))/ (np.max(x)-np.min(x))
x
```

	long_hair	forehead_width_cm	forehead_height_cm	nose_wide	nose_long	lips_thin	distance_nose_to_lip_long
0	1.0	0.097561	0.50	1.0	0.0	1.0	1.0
1	0.0	0.634146	0.15	0.0	0.0	1.0	0.0
2	0.0	0.097561	0.60	1.0	1.0	1.0	1.0
3	0.0	0.731707	0.50	0.0	1.0	1.0	1.0
4	1.0	0.512195	0.40	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...
4996	1.0	0.536585	0.00	0.0	0.0	0.0	0.0
4997	1.0	0.121951	0.15	0.0	0.0	0.0	0.0
4998	1.0	0.365854	0.30	0.0	0.0	0.0	0.0
4999	1.0	0.439024	0.55	0.0	0.0	0.0	0.0
5000	1.0	0.975610	0.15	1.0	1.0	1.0	1.0

5001 rows × 7 columns

05 SPLIT THE DATASET

Split tha data

```
[ ]  # featurs
     x = data.drop('gender', axis='columns')
     x
```

Dropping the column (gender) so the machine doesn't train it

```
# target value
y = data.gender
y
```

Displaying the labels

05 SPLIT THE DATASET

```
#split tha data for train and test
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=1)
```

Splitting the dataset into 80% training and 20% testing set

ML ALGORITHMS USED:

- K Nearest Neighbor (KNN)
- Support Vector Machine (SVM)
- Random Forest Classifier
- Logistic Regression
- Artificial Neural Network (ANN)

KNN

- Importing the KNN model and fitting the data into the model to be trained
- Printing accuracy scores of training and testing data

```
# KNN package
from sklearn.neighbors import KNeighborsClassifier
# Create KNN Classifier
knn = KNeighborsClassifier(n_neighbors = 2 )
```

```
#Train the model using the training sets
knn.fit(X_train, y_train)
```

```
KNeighborsClassifier(n_neighbors=2)
```

```
print("train accuracy:",knn.score(X_train,y_train))
print("test accuracy:",knn.score(X_test,y_test))
```

```
train accuracy: 0.978
test accuracy: 0.958041958041958
```

07 IMPROVING RESULTS: (KNN)

- Plotting the confusion matrix.
- New accuracy using best features

```
# Plot the confusion matrix.
y_predicted = knn.predict(X_test)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_predicted)
cm
```

```
array([[500,   4],
       [ 38, 459]])
```

```
# Arrange the train and test dataset including the best feature
selector = SelectKBest(f_classif, k = 3)
x_new = selector.fit_transform(X_train, y_train)
x_new_test=selector.fit_transform(X_test,y_test)
names_train = X_train.columns.values[selector.get_support()]
names_test = X_test.columns.values[selector.get_support()]
print("x train features:",names_train)
print("x test features:",names_test)
```

```
x train features: ['nose_wide' 'nose_long' 'lips_thin']
x test features: ['nose_wide' 'nose_long' 'lips_thin']
```

```
#Re-train and re-calculate the model accuracy using the new arrangement of featur
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 2 )
knn.fit(x_new,y_train)
print("train accuracy:",knn.score(x_new,y_train))
print("test accuracy:",knn.score(x_new_test,y_test))
```

```
train accuracy: 0.95925
test accuracy: 0.955044955044955
```

KNN

- Cross-validation method gives us a better understanding of the model performance over the whole dataset instead of just a single train/test split.
- Returns the accuracy for all the assigned folds

```
]  from sklearn.model_selection import cross_val_score
   cross_val_score(knn, x, y, cv=5)
```

```
array([0.95404595, 0.966     , 0.971     , 0.967     , 0.969     ])
```

```
]  cross_val_score(knn, x, y, cv=10)
```

```
array([0.94610778, 0.956     , 0.972     , 0.966     , 0.972     ,
       0.968     , 0.97      , 0.964     , 0.974     , 0.972     ])
```

```
]  np.random.seed(42)

   # Single training and test split score

   # Take the mean of 5-fold cross-validation score
   knn_cross_val_score = np.mean(cross_val_score(knn, x, y, cv=10))

   # Compare the two
   knn_cross_val_score
```

```
0.9660107784431137
```

SVM

- Importing the SVM model and fitting the data into the model to be trained
- Printing accuracy scores of training and testing data

```
[ ]  #importing SVM package
     from sklearn.svm import SVC
     svm = SVC(random_state=1)
```

```
[ ]  #train model using train set of x,y
     svm.fit(X_train,y_train)

     SVC(random_state=1)
```

```
print("train accuracy:",svm.score(X_train,y_train))
print("test accuracy:",svm.score(X_test,y_test))
```

```
train accuracy: 0.97425
test accuracy: 0.971028971028971
```

IMPROVING RESULTS: (SVM)

- Plotting the confusion matrix
- New accuracy using best features

```
y_predicted = svm.predict(X_test)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_predicted)
cm
```

```
array([[497,   7],
       [ 22, 475]])
```

```
# Arrange the train and test dataset
selector = SelectKBest(f_classif, k = 3)
x_new = selector.fit_transform(X_train, y_train)
x_new_test=selector.fit_transform(X_test,y_test)
names_train = X_train.columns.values[selector.get_support()]
names_test = X_test.columns.values[selector.get_support()]
print("x train features:",names_train)
print("x train features:",names_test)
```

```
x train features: ['nose_wide' 'nose_long' 'lips_thin']
x train features: ['nose_wide' 'nose_long' 'lips_thin']
```

```
# Re-train and re-calculate the model accuracy using the new arrangement of features
from sklearn.svm import SVC
svm=SVC(random_state=1)
svm.fit(x_new,y_train)
print("train accuracy:",svm.score(x_new,y_train))
print("test accuracy:",svm.score(x_new_test,y_test))
```

```
train accuracy: 0.95925
test accuracy: 0.955044955044955
```

SVM

- Applying Cross-validation method to give us a better understanding of the model performance

- Returns the accuracy for all the assigned folds

```
[ ] from sklearn.model_selection import cross_val_score
    cross_val_score(svm, x, y, cv=5)
```

```
array([0.95504496, 0.965     , 0.977     , 0.968     , 0.975     ])
```

```
[ ] cross_val_score(svm, x, y, cv=10)
```

```
array([0.94810379, 0.962     , 0.972     , 0.96      , 0.982     ,
       0.976     , 0.976     , 0.964     , 0.978     , 0.97      ])
```

```
[ ] np.random.seed(42)

    # Single training and test split score

    # Take the mean of 5-fold cross-validation score
    svm_cross_val_score = np.mean(cross_val_score(svm, x, y, cv=10))

    # Compare the two
    svm_cross_val_score
```

```
0.968810379241517
```

RANDOM FOREST CLASSIFIER

- Importing the RFC model and fitting the data into the model to be trained
- Printing accuracy scores of training and testing data

```
#import the randomforsetclssifer , crate Rf classifer and train th
#The random forest then takes the prediction from each tree and se
from sklearn.ensemble import RandomForestClassifier
forset = RandomForestClassifier(n_estimators=10)
forset.fit(X_train, y_train)
```

```
RandomForestClassifier(n_estimators=10)
```

```
print("train accuracy:",forset.score(X_train,y_train))
print("test accuracy:",forset.score(X_test,y_test))
```

```
train accuracy: 0.997
test accuracy: 0.968031968031968
```

IMPROVING RESULTS: (RFC)

- Plotting the confusion matrix
- New accuracy using best features

```
# 12.Plot the confusion matrix.
y_predicted = forset.predict(X_test)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_predicted)
cm
```

```
array([[493,  11],
       [ 19, 478]])
```

```
# Arrange the train and test dataset including best features
#LR
selector = SelectKBest(f_classif, k = 3)
x_new_forest= selector.fit_transform(X_train, y_train)
x_new_test_forest=selector. fit_transform(X_test, y_test)
names_train_forest = X_train.columns.values[selector.get_support()]
names_test_forest = X_test.columns.values[selector.get_support() ]
print("x train features:",names_train_forest)
print("x test features:",names_test_forest)
```

```
x train features: ['nose_wide' 'nose_long' 'lips_thin']
x test features: ['nose_wide' 'nose_long' 'lips_thin']
```

```
#Re-train and re-calculate the model accuracy using the new arrangement of featur
#import backage
from sklearn.ensemble import RandomForestClassifier
forset = RandomForestClassifier(n_estimators=10)
forset.fit(x_new_forest, y_train)
#Calc score or (Accuracy)
print("train accuracy: ",forset.score(x_new_forest, y_train))
print("test accuracy: ",forset.score(x_new_test_forest,y_test))
```

```
train accuracy:  0.95925
test accuracy:  0.955044955044955
```

RANDOM FOREST CLASSIFIER

- Applying Cross-validation method to give us a better understanding of the model performance

- Returns the accuracy for all the assigned folds

```
[ ] from sklearn.model_selection import cross_val_score
    cross_val_score(forset, x, y, cv=5)
```

```
array([0.96103896, 0.961     , 0.976     , 0.967     , 0.975     ])
```

```
[ ] np.random.seed(42)

    # Single training and test split score

    # Take the mean of 5-fold cross-validation score
    forest_cross_val_score = np.mean(cross_val_score(forset, x, y, cv=10))

    # Compare the two
    forest_cross_val_score
```

```
0.967810379241517
```

LOGISTIC REGRESSION

- Importing the LR model and fitting the data into the model to be trained
- Printing accuracy scores of training and testing data

```
from sklearn.linear_model import LogisticRegression
# creating linear regression object.
lgrgmodel = LogisticRegression()
```

```
lgrgmodel.fit(X_train, y_train)
```

```
LogisticRegression()
```

```
print("train accuracy:",lgrgmodel.score(X_train,y_train))
print("test accuracy:",lgrgmodel.score(X_test,y_test))
```

```
train accuracy: 0.9685
test accuracy: 0.9660339660339661
```

IMPROVING RESULTS: (LR)

- Plotting the confusion matrix
- Plotting best accuracy

```
# Plot the confusion matrix.
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, lgrgmodel_pred)
cm
```

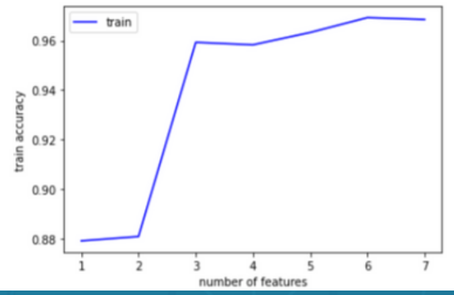
```
array([[485,  19],
       [ 15, 482]])
```

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif

#improve the accuracy of prediction.#LR
print("==================================#LR")
accuracy_list_train_LR = []
number_of_features_LR = np.arange(1,8,1)
for each in number_of_features_LR:
 x_new_LR = SelectKBest(f_classif, k = each).fit_transform(X_train, y_train)
 lgrgmodel.fit(x_new_LR, y_train)
 accuracy_list_train_LR.append(lgrgmodel.score(x_new_LR, y_train))

plt.plot(number_of_features_LR,accuracy_list_train_LR,color="blue", label="train")
plt.xlabel("number of features")
plt.ylabel("train accuracy")
plt.legend()
plt.show()
```

```
==================================#LR
```



LOGISTIC REGRESSION

- New accuracy using best features
- Applying Cross-validation method to give us a better understanding of the model performance

```
# Arrange the train and test dataset including best features
#LR
selector = SelectKBest(f_classif, k = 3)
x_new_LR = selector.fit_transform(X_train, y_train)
x_new_test_LR=selector. fit_transform(X_test, y_test)
names_train_LR = X_train.columns.values[selector.get_support()]
names_test_LR = X_test.columns.values[selector.get_support() ]
print("x train features:",names_train_LR)
print("x test features:",names_test_LR)
```

```
x train features: ['nose_wide' 'nose_long' 'lips_thin']
x test features: ['nose_wide' 'nose_long' 'lips_thin']
```

```
#Re-train and re-calculate the model accuracy using the new arrangement of featur
#LR
#import backage
from sklearn.linear_model import LogisticRegression
lgrgmodel = LogisticRegression()
lgrgmodel.fit(x_new_LR, y_train)
#Calc score or (Accuracy)
print("train accuracy: ",lgrgmodel.score(x_new_LR, y_train))
print("test accuracy: ",lgrgmodel.score(x_new_test_LR,y_test))
```

```
train accuracy:  0.95925
test accuracy:  0.955044955044955
```

```
] from sklearn.model_selection import cross_val_score
  cross_val_score(lgrgmodel, x, y, cv=5)
```

```
array([0.96103896, 0.961     , 0.976     , 0.967     , 0.975     ])
```

```
] cross_val_score(lgrgmodel, x, y, cv=10)
```

```
array([0.94810379, 0.974     , 0.966     , 0.958     , 0.974     ,
       0.976     , 0.968     , 0.966     , 0.976     , 0.972     ])
```

```
] np.random.seed(42)

  # Single training and test split score

  # Take the mean of 5-fold cross-validation score
  lgrgmodel_cross_val_score = np.mean(cross_val_score(lgrgmodel, x, y, cv=10))

  # Compare the two
  lgrgmodel_cross_val_score
```

```
0.967810379241517
```

ARTIFICIAL NEURAL NETWORK

- Importing tenserflow
- Defining the number of layers that matches our data's features (7)
- Deviding the dataset into (10) epochs

```
] import tensorflow as tf
  from tensorflow import keras

] X_train.shape

  (4000, 7)

] y_train.shape

  (4000,)

] #Defining the model and adding the layers
  model = keras.Sequential([
      keras.layers.Dense(10,input_shape=(7,), activation='sigmoid')
  ])

] model.compile(optimizer='adam',
                loss='sparse_categorical_crossentropy',
                metrics=['accuracy'])

] model.fit(X_train,y_train , epochs=10)

  Epoch 1/10
  125/125 [==============================] - 1s 4ms/step - loss: 0.6311 - accuracy: 0.8537
  Epoch 2/10
  125/125 [==============================] - 1s 6ms/step - loss: 0.5423 - accuracy: 0.8827
  Epoch 3/10
  125/125 [==============================] - 1s 7ms/step - loss: 0.4716 - accuracy: 0.9122
  Epoch 4/10
  125/125 [==============================] - 0s 2ms/step - loss: 0.4151 - accuracy: 0.9323
  Epoch 5/10
  125/125 [==============================] - 0s 2ms/step - loss: 0.3697 - accuracy: 0.9405
```

ARTIFICIAL NEURAL NETWORK

```
125/125 [==============================] - 0s 2ms/step - loss: 0.3330 - accuracy: 0.9442
Epoch 7/10
125/125 [==============================] - 0s 2ms/step - loss: 0.3031 - accuracy: 0.9460
Epoch 8/10
125/125 [==============================] - 0s 2ms/step - loss: 0.2784 - accuracy: 0.9470
Epoch 9/10
125/125 [==============================] - 0s 2ms/step - loss: 0.2577 - accuracy: 0.9475
Epoch 10/10
125/125 [==============================] - 0s 2ms/step - loss: 0.2403 - accuracy: 0.9475
<keras.callbacks.History at 0x7ff8e57bdb10>
```

```
] model.evaluate(X_test,y_test)
```

```
32/32 [==============================] - 0s 2ms/step - loss: 0.2352 - accuracy: 0.9481
[0.23524464666843414, 0.948051929473877]
```

Model evaluation of the test set:

- Data loss
- Data accuarcy

08 COMPARING RESULTS

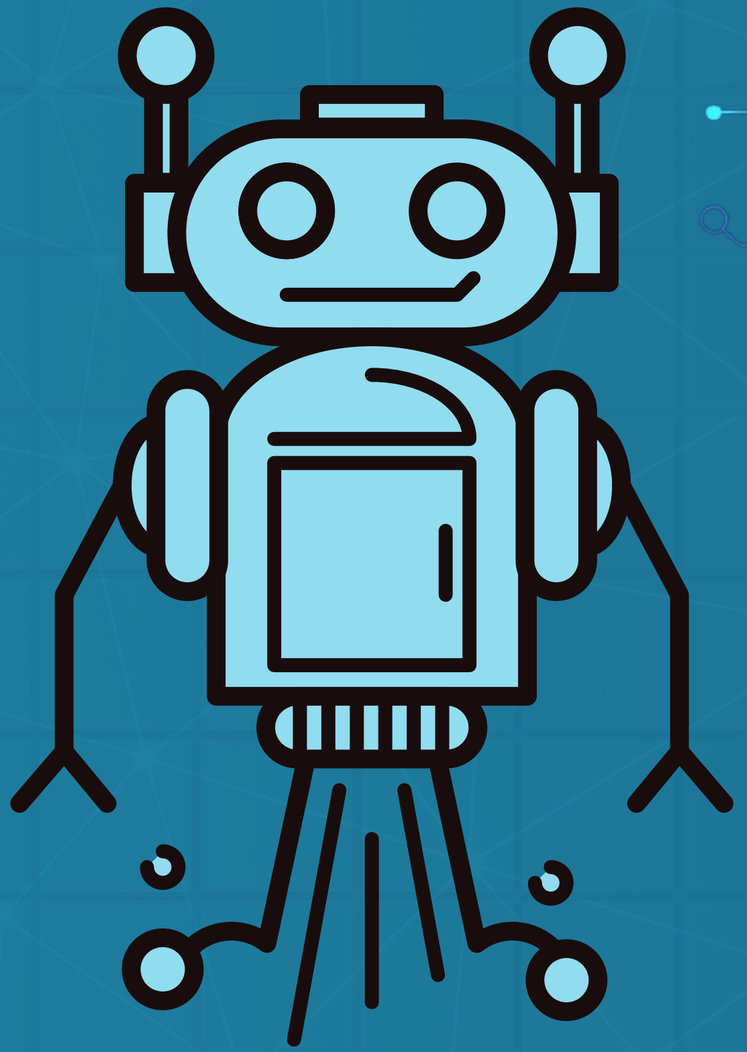
Status	SVM	KNN	LogisticRegression	RandomForestClassifier	neural network
test accuracy	0.971	0.958	0.966	0.968	0.948
After improve	0.955	0.955	0.95	0.955	
cross-validation score	0.968	0.966	0.967	0.967	

We use Cross-validation score to choose the best algorithm. It has more accurate estimate of out-of-sample accuracy and has more efficient use of data as every observation is used for both training and testing.

So based on the Cross-validation score, SVM has the best accuracy by 0.968 Therefore, it is the best algorithm applied to our data.

THANKS!

Feel free to ask questions :)



The link of the project :
https://colab.research.google.com/drive/1-xBJ8_Q8JT_eey3TD5uotr7tb1UeGeQi?usp=sharing