Report

"Performance of various Machine Learning Algorithms on Electrical Impedance Tomography Images"

By,

Sudhanva Narayana,

Intern, Machine Learning Engineer,

Phone: +91 9632350260

Email: nsudhanva@gmail.com

Faststream Technologies,

6, 2nd Main Rd, Arekere Off Bannerghatta Road, Bengaluru, KA, 560076

Phone: +91 80–4161–3204/3205

Email: info@faststreamtech.com

Table of Contents

1. ABSTRACT

1. Abstract

This report will be of interest for Health Care Data Analysts, Data Scientists, Doctors and Medical researchers. This report provides an overview of current practice of Electrical Impedance Tomography (EIT), its imaging and use-cases. Electrical Impedance Tomography is a non-invasive type of medical imaging. These advances are improving our capacity to treat and even prevent cancers. The full implications of the subject remain to be explored. Examples of research techniques used in this project are detailed.

1.a. Background

Faststream Technologies is a vanguard of technology solutions, specializing in Product & System Engineering, IoT, Big Data, Security, and Application Development with a global footprint across North America, EMEA, and APAC. With over 200+ clients, Faststream Technologies enables Digital Transformation for enterprises by delivering a flawless customer experience, business competence, and deep insights through an integrated set of disruptive technologies and expertise. We are passionate about delivering well-organized, inventive and world-class hardware and software solutions, with a focus on Healthcare, Aerospace, Semiconductors, Automotive, Consumer Electronics, Home Automation, Telecommunications, Security, Retail, and E-Commerce.

Faststream Technologies works at the juncture of business and technology, assisting clients with advancing their product and business performance through sustainable information technology solutions. Faststream Technologies drives innovation to help clients advance their product design, business processes, and application development. Our engineering team's deep expertise in transforming design specs into marketable hardware products — through ASIC design services that include RTL design, design verification and physical design for digital and analogue/mixed-signal semiconductors — is a key differentiator to our suite of application development capabilities.

For today's challenges like embedded processor SoC specifications, Faststream Technologies delivers all of the required firmware/embedded software, positioning us as the turnkey 'concept-to-product' design company. The team is led by a group of focused senior executives and Technologists who complement each other with significant industry experience in building turnkey solutions.

Many of our technologists have multiple patents to their credit in the areas of Analog/Mixed-Signal Design, IoT and embedded systems.

1.b. Project Objectives

The goal of the project is to validate performance of Electrical Impedance Tomography's performance across various Machine Learning – Classification algorithms. Image is read into code in the form of a three-dimensional matrix where in each dimension represents intensities of the respective colour code. This three-dimensional matrix is then converted to two-dimensional matrix (representation of grayscale image) with intensities ranging from 0 to 1. Image is re generated to observe distribution using contour plots. A.

Based on the data obtained and observation from the graphs, random multidimensional matrices are generated. Using radial basis function on these matrices, values ranging from 0 to 1 are created. 1000 random-related images are created based on the matrices and its values. The generated images are read back into code and are plotted to observe the distribution of intensities. Mean intensity ranges are calculated and are assigned labels (colours) correspondingly. The generated images are parsed and respective intensity ranges, its count of pixels and percentages are calculated. A dataset of 8 intensity ranges (columns) and 1000 values (rows) are created. Mean of pixel count of all ranges are taken in consideration and is used as a criterion for assigning targets. Binary targets are generated and are appended to the existing dataset as a target column.

List of classifiers/algorithms used:

- K – Nearest Neighbours
- Decision Tree Classifier
- Kernel Support Vector Machines
- Logistic Regression Classifier
- Naïve Bayes Classifier
- Random Forest Classifier
- Support Vector Machines

The result is interpreted and plotted measuring the performance of the mentioned algorithms above.

2. INTRODUCTION

2. Introduction

Human bodies have electrical properties, specifically the electric conductivity and permittivity. The electric conductivity is a measure of the ease with which a material conducts electricity; the electric permittivity is a measure of how readily the charges within a material separate under an imposed electric field. Highly conductive materials allow both AC and DC currents to pass through them. Highly permissive materials allow only AC current to pass through them. Both of these properties can be used in medical applications as tumours, tissues and other irregularities in human body have different conductive and permissive properties. Other application of EIT include detection of blood clots, pulmonary emboli and gas in human body.

2.a. Scope of the project

The project works across any images in general but is concentrated on images generated by contours and sine, cosine functions. A sample EIT Image is read into the code in the form of a 2-dimensional matrix. This matrix represents intensities of various colour gamut. The project revolves around generating images and reading those images into matrices. A dataset of count of pixels of various intensity ranges are created. A machine learning model is created out the dataset.

2.b. Limitations of the project

The images are generated using numpy's random number generator and mesh-grid technique which uses sine and cosine function to generate contour like matrices. However, these generated images are not real and should be used for experimental purposes only. The accuracy, methodology of the machine learning algorithms is true whereas the images are not. The project is also dependent on certain Python environments and related tools. It is independent of the development environment.

3. MACHINE LEARNING – INTRODUCTION

3. Machine Learning

Machine Learning is the science (and art) of programming computers so they can learn from data. For example, your spam filter is a Machine Learning program that can learn to flag spam given examples of spam emails (e.g., flagged by users) and examples of regular (non-spam) emails. The examples that the system uses to learn are called the training set. Each training example is called a training instance (or sample). In this case, the task T is to flag spam for new emails, the experience E is the training data, and the performance measure P needs to be defined; for example, you can use the ratio of correctly classified emails. This particular performance measure is called accuracy and it is often used in classification tasks.

<u>Supervised/Unsupervised Learning</u>

Supervised Learning

Machine Learning systems can be classified according to the amount and type of supervision they get during training. There are four major categories: supervised learning, unsupervised learning, semi-supervised learning, and Reinforcement Learning. In supervised learning, the training data you feed to the algorithm includes the desired solutions, called labels. A typical supervised learning task is classification. The spam filter is a good example of this: it is trained with many example emails along with their class (spam or ham), and it must learn how to classify new emails.

Unsupervised learning

In unsupervised learning, as you might guess, the training data is rranged. The system tries to learn without a teacher.

- **Clustering**
    - k-Means
    - Hierarchical Cluster Analysis (HCA)
    - Expectation Maximization
- **Visualization and dimensionality reduction**

- Principal Component Analysis (PCA)
- Kernel PCA
- Locally-Linear Embedding (LLE)
- t-distributed Stochastic Neighbor Embedding (t-SNE)

3.a. Tools and Technologies

1. Python – a general-purpose interpreted, interactive, object-oriented, and high-level programming language
2. Anaconda – a free and open source distribution of the Python and R programming languages for data science and machine learning related applications, that aims to simplify package management and deployment
3. Numpy – the fundamental package for scientific computing with Python.
4. Scipy - a Python-based ecosystem of open-source software for mathematics, science, and engineering
5. Matplotlib – a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms
6. Sci-kit learn – a free software machine learning library for the Python programming language.

3.b. Methodologies

1. Data is checked for validity, accuracy, completeness and consistency
2. Data auditing is made to detect anomalies and contradictions
3. The detection and removal of anomalies is performed by a sequence of operations on the data known as the workflow.
4. After executing the cleansing workflow, the results are inspected to verify correctness. Data that could not be corrected during execution of the workflow is manually corrected.
5. Training set is determined. Here, 75% of the generated dataset is used as training data and the rest 25% is test data.
6. The training set represents the real-world use if the function. A set of input objects is gathered and corresponding outputs are also gathered.
7. The input representation of the learn function is determined and the accuracy of the learned function depends strongly on how the input

object is represented. The input object is transformed into a feature vector, which contains a number of features that are descriptive of the object.

8. The structure of the learned function is determined (any Machine Learning algorithm can be used)
9. The design (model) is completed and I run on the gathered training set. Some of the supervised algorithms require the user to determine certain control parameters. These parameters are adjusted by optimising performance on the test set of the validation set and cross validation is also applied
10. Accuracy of the learned function is determined. After the parameter is adjusted, the performance of the resulting function is measured on the test set that is separate from the training set.

4. IMAGE PROCESSING

4. Image Processing

An image in processed in numpy is basically a three-dimensional matrix where each dimension represents the intensity of the respective colour value. A couple of few basic assumptions and paradigms are made when processing an image. The image is converted to grayscale. It can also be a colour filter or a gradient but to simplify the process, it is converted to grayscale. A grayscale image is basically a two-dimensional matrix where each dimension represents the intensity of either black or white.

```
# Copyright © 2018, Faststream Technologies
# Author: Sudhanva Narayana

import numpy as np
import matplotlib.pyplot as plt
import os

# Import to show plots in © Windows
from Ipython import get_ipython
get_ipython().run_line_magic(‘matplotlib’, ‘qt5’)

# CURR and PARENT directory constants
CURR_DIR = os.path.dirname(os.path.abspath(‘__file__’))
PARENT_DIR = os.path.abspath(os.path.join(CURR_DIR, os.pardir))

# Generate two 1D vectors of Uniform float – range 0 to 1 of size
100
matrix1 = np.random.uniform(low=0, high=1, size=100)
matrix2 = np.random.uniform(low=0, high=1, size=100)

# Create a 2D matrix out of the two 1D vectors
matrix = np.array([matrix1, matrix2])

# Import image – converts image into a 3D numpy array
img = plt.imread(PARENT_DIR + ‘\\assets\\eit_images\\eitcrop.png’)

# Convert the colored-3D image into grayscale-2D
grayscale = img.mean(axis=2)

# Flatten 2D array to 1D array
x = grayscale.ravel()
```

```
y = img.ravel()

# Generate an image based on the ‘grayscale’ 2D matrix
plt.set_cmap(‘gray’)
plt.imshow(grayscale, cmap=’gray’)
plt.axis(‘off’)
plt.show()
```

5. IMAGE GENERATION

5. Image Generation

1000 sample images are generated using the following tools:

Numpy random –

- Returns a sample (or samples) from the “standard normal” distribution.
- If positive, int_like or int-convertible arguments are provided, random generates an array of shape (d0, d1, ..., dn), filled with random floats sampled from a univariate “normal” (Gaussian) distribution of mean 0 and variance 1 (if any of the d_i are floats, they are first converted to integers by truncation). A single float randomly sampled from the distribution is returned if no argument is provided

```
# Generate data:
x, y, z = 10 * np.random.random((3, 50))
```

Numpy meshgrid –

- Return coordinate matrices from coordinate vectors.
- Make N-D coordinate arrays for vectorized evaluations of N-D scalar/vector fields over N-D grids, given one-dimensional coordinate arrays x1, x2,..., xn.

```
# Set up a regular grid of interpolation points
xi, yi = np.linspace(x.min(), x.max(), 100), np.linspace(y.min(),
y.max(), 100)

xi, yi = np.meshgrid(xi, yi)
```

Scipy interpolate

- This sub-package contains spline functions and classes, one-dimensional and multi-dimensional (univariate and multivariate) interpolation classes

```
# Interpolate
rbf = scipy.interpolate.Rbf(x, y, z, function=’linear’)
zi = rbf(xi, yi)
```

Image generation

```
# Copyright © 2018, Faststream Technologies
# Author: Sudhanva Narayana

import numpy as np
import matplotlib.pyplot as plt
import scipy.interpolate
import os

# CURR and PARENT directory constants
CURR_DIR = os.path.dirname(os.path.abspath('__file__'))
PARENT_DIR = os.path.abspath(os.path.join(CURR_DIR, os.pardir))

for © in range(1, 1001):
    # Generate data:
    x, y, z = 10 * np.random.random((3, 50))

    # Set up a regular grid of interpolation points
    xi, yi = np.linspace(x.min(), x.max(), 100),
np.linspace(y.min(), y.max(), 100)
    xi, yi = np.meshgrid(xi, yi)

    # Interpolate
    rbf = scipy.interpolate.Rbf(x, y, z, function='linear')
    zi = rbf(xi, yi)

    clrs = ('brown', 'red', 'orange', 'yellow', 'green', 'blue',
'violet', 'gray')
    plt.contourf(zi, colors=clrs)
    plt.axis('off')
    plt.savefig(PARENT_DIR + '\\assets\\eit_images\\' + "eit_" +
str(i) + ".png", bbox_inches='tight')
```

6. IMAGE CLASSIFICATION

6. Image Classification

Intensity range is calculated based on basic resistive values. A dictionary of ‘colours’ is mapped to the respective ‘list’ of intensity values for 1000 images.

```
# Copyright © 2018, Faststream Technologies
# Author: Sudhanva Narayana

import numpy as np
import matplotlib.pyplot as plt
import cv2
import os

# Import to show plots in © Windows
from Ipython import get_ipython
get_ipython().run_line_magic(‘matplotlib’, ‘qt5’)

# CURR and PARENT directory constants
CURR_DIR = os.path.dirname(os.path.abspath(‘__file__’))
PARENT_DIR = os.path.abspath(os.path.join(CURR_DIR, os.pardir))

# Import image - converts image into a 3D numpy array
img = cv2.imread(PARENT_DIR + ‘\\assets\\eit_images\\eitcrop.png’)

# Convert the colored-3D image into grayscale-2D
img_two_d = img.mean(axis=2)

# Flatten 2D array to 1D array
img_one_d = img_two_d.ravel()

# Initial setup of intensity range
colors = [‘brown’, ‘red’, ‘orange’, ‘yellow’, ‘green’, ‘blue’,
‘violet’, ‘gray’]
classify_dict = {}
low = 0.0
high = 1.1
skip = 0.1

# Generate intensity range
```

```
intensity = np.arange(low, high, skip)
color_length = high / len(colors)
color_length_array = np.full((1, len(intensity) - 1),
round(color_length, 2))
color_length_array = np.insert(color_length_array, 0, 0)

# Calculate cumulative sum of average range of intensity
intensity_range = np.cumsum(color_length_array)
intensity_range_strings = []

# Create a classify dict of colors mapping to their datapoints
(array)
for index, color in enumerate(colors):
    # print(intensity_range[index], intensity_range[index + 1])
    intensity_range_strings.append(str(round(intensity_range[index],
2)) + ‘ – ‘ + str(round(intensity_range[index + 1], 2)))
    classify_dict[color] = np.where(np.logical_and(img_one_d >=
intensity_range[index], img_one_d < intensity_range[index + 1]))[0]

# Create a count of classified dict
classify_dict_count = {}

for key, value in classify_dict.items():
    classify_dict_count[key] = len(value)

colors_tuple = tuple(classify_dict_count.keys())
y_pos = np.arange(len(colors_tuple))
pixels = classify_dict_count.values()

# Subplot to map a bar chart and its labels
fig, ax = plt.subplots()
rects = ax.bar(y_pos, pixels, align=’center’, alpha=0.5,
color=colors_tuple)

def autolabel(rects):
    “””
    Attach a text label above each bar displaying its height
    “””
    for rect in rects:
        height = rect.get_height()
        ax.text(rect.get_x() + rect.get_width()/2., height,
                ‘%d’ % int(height),
                ha=’center’, va=’bottom’)

autolabel(rects)

# Plot a bar graph with intensities, colors and count of pixels
plt.bar(y_pos, pixels, align=’center’, alpha=0.5,
color=colors_tuple)
plt.xticks(y_pos, tuple(intensity_range_strings))
plt.ylabel(‘Pixels Count’)
plt.title(‘Pixels vs Colors’)
plt.savefig(PARENT_DIR + ‘\\assets\\plots\\eit_classify_plot.png’)
plt.show()
```

7. DATASET GENERATION

7. Dataset Generation

The generated images are parsed and are imported into code. These images are basically matrices as mentioned earlier. The pixel intensity is categorised based on the intensity range. Two datasets are generated. One contains the count of pixels categorised into respective intensity ranges and the other contains percentage of the pixels covered per image.

```
# Copyright © 2018, Faststream Technologies
# Author: Sudhanva Narayana

import numpy as np
import pandas as pd
import cv2
import os

# Import to show plots in © Windows
# from Ipython import get_ipython
# get_ipython().run_line_magic(‘matplotlib’, ‘qt5’)

# CURR and PARENT directory constants
CURR_DIR = os.path.dirname(os.path.abspath(‘__file__’))
PARENT_DIR = os.path.abspath(os.path.join(CURR_DIR, os.pardir))
colors = [‘brown’, ‘red’, ‘orange’, ‘yellow’, ‘green’, ‘blue’,
‘violet’, ‘gray’]
colors = colors[::-1]
colors_p = [© + ‘_%’ for © in colors]

image_files_list = []

for © in range(1, 1001):
    image_files_list.append(‘eit_’ + str(i) + ‘.png’)

# Initial setup of intensity range
low = 0
high = 255
skip = 1

classify_dict = {}
classify_dict_per = {}
```

```
# Generate intensity range
intensity = np.arange(low, high, skip)
color_length = high / len(colors)
color_length_array = np.full((1, len(intensity) - 1),
round(color_length, 2))
color_length_array = np.insert(color_length_array, 0, 0)

# Calculate cumulative sum of average range of intensity
intensity_range = np.cumsum(color_length_array)

for c, p in zip(colors, colors_p):
    classify_dict[c] = []
    classify_dict_per[p] = []

for image_file in image_files_list:
    # Import image - converts image into a 3D numpy array
    # img = cv2.imread(PARENT_DIR + '\\assets\\eit_images\\' +
image_file)

    # Import the colored-3D image into grayscale-2D
    img_two_d = cv2.imread(PARENT_DIR + '\\assets\\eit_images\\' +
image_file, 0)

    # Flatten 2D array to 1D array
    img_one_d = img_two_d.ravel()

    total_length = len(img_one_d)
    intensity_range_strings = []

    # Create a classify dict of colors mapping to their datapoints
(array)
    for index, (color, color_p) in enumerate(zip(colors, colors_p)):
        # print(intensity_range[index], intensity_range[index + 1])

intensity_range_strings.append(str(round(intensity_range[index], 2))
+ ' - ' + str(round(intensity_range[index + 1], 2)))
        intensity_range_length =
len(np.where(np.logical_and(img_one_d >= intensity_range[index],
img_one_d < intensity_range[index + 1]))[0])
        percentage = (intensity_range_length/total_length) * 100
        percentage = round(percentage, 2)
        classify_dict[color].append(intensity_range_length)
        classify_dict_per[color_p].append(percentage)

columns_tuple_list = []
# print(classify_dict)
for color, intensity_range in zip(colors, intensity_range_strings):
    columns_tuple_list.append((color, intensity_range))
    # print(color, intensity_range)

# Created tuples for DataFrames
columns_p_tuple = list(zip(*[iter(colors_p)]*1, [‘100’] * 8))
columns_tuple_list.sort(key=lambda tup: tup[0])

# DataFrame for values
df = pd.DataFrame(classify_dict)
df.columns = pd.MultiIndex.from_tuples(columns_tuple_list)
df = df[colors]
df.to_csv(PARENT_DIR + ‘\\assets\\datasets\\’ + ‘eit.csv’)

# DataFrame for percentages
df_p = pd.DataFrame(classify_dict_per)
df_p.columns = pd.MultiIndex.from_tuples(columns_tuple_list)
df_p = df_p[colors]
df_p.to_csv(PARENT_DIR + ‘\\assets\\datasets\\’ + ‘eit_p.csv’)
```

8. DATASET TARGET GENERATION

8. Dataset Target Generation

The generated dataset is checked for errors and numerical anomalies. Once the data is pre-processed then mean of each column is calculated. These means are made the basic criteria for classification threshold. If the pixel count is more than the mean, the image is categorised as ‘1’. If the pixel count is less than the mean, the image is categorised as ‘0’. Once the classification is done, the ‘target’ column is appended and is attached to the existing dataset. Another dataset is generated and is considered as the final dataset for machine learning.

```
# Copyright © 2018, Faststream Technologies
# Author: Sudhanva Narayana

import numpy as np
import pandas as pd
import os

# Import to show plots in © Windows
# from Ipython import get_ipython
# get_ipython().run_line_magic(‘matplotlib’, ‘qt5’)

# CURR and PARENT directory constants
CURR_DIR = os.path.dirname(os.path.abspath(‘__file__’))
PARENT_DIR = os.path.abspath(os.path.join(CURR_DIR, os.pardir))

# Import dataset ignoring headers
df = pd.read_csv(PARENT_DIR + ‘\\assets\\datasets\\eit.csv’,
index_col=[0], header = [0], skiprows= [1] ,skipinitialspace=True)
df_ranges = pd.read_csv(PARENT_DIR + ‘\\assets\\datasets\\eit.csv’,
index_col=[0], header = [0], skiprows= [0], skipinitialspace=True,
nrows=0)
df_columns_ranges = list(df_ranges.columns)
df_columns_colors = list(df.columns)
df_means = df.mean()

target_series = []

# Create target_series list of  rrange
for ©, color in enumerate(df_columns_colors):
    target_series.append(df[color] > df_means[i])

target = np.array(target_series)
target = np.transpose(target[-4:])

target_bools = []

# Create target_bools which creates the final Series of target
column
for © in range(len(target)):
    if np.sum(target[i]) >= 1:
        target_bools.append(1)
    else:
        target_bools.append(0)

target_bools = pd.Series(target_bools)

columns_tuple_list = []

# Tuple for creating columns for DataFrame
for color, intensity_range in zip(df_columns_colors,
df_columns_ranges):
    columns_tuple_list.append((color, intensity_range))

# Final DataFrame to csv
df.columns = pd.MultiIndex.from_tuples(columns_tuple_list)
df['target'] = target_bools
df.to_csv(PARENT_DIR + '\\assets\\datasets\\' + 'eit_data.csv')
```

9. MACHINE LEARNING – CLASSIFICATION

9. Machine Learning – Classification

- Dataset is split into training set and test set
- Feature scaling is applied, data is fit into various classifiers and models are evaluated for performance
- Confusion matrices and classification reports are generated
- ***List of classifiers/algorithms used:***
    - K – Nearest Neighbours
    - Decision Tree Classifier
    - Kernel Support Vector Machines
    - Logistic Regression Classifier
    - Naïve Bayes Classifier
    - Random Forest Classifier
    - Support Vector Machines

K- Nearest Neighbours Classifiers

```
# coding: utf-8

# # Performance validation of Electrical Impedance Tomography Images
using K-Nearest Neighbors Classifier – Machine Learning
#
# ## Copyright © 2018, Faststream Technologies
# ## Author: Sudhanva Narayana

# In[1]:


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
from sklearn.neighbors import KneighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

```
# ### CURR and PARENT directory constants

# In[2]:


CURR_DIR = os.path.dirname(os.path.abspath('__file__'))
PARENT_DIR = os.path.abspath(os.path.join(CURR_DIR, os.pardir))


# ### Import dataset ignoring headers

# In[3]:


df = pd.read_csv(PARENT_DIR + '\\assets\\datasets\\eit_data.csv',
index_col=[0], header = [0], skiprows= [1] ,skipinitialspace=True)


# ### Dataset

# In[4]:


df.head()


# ### Visualise the higher intensities

# In[5]:


plt.scatter(df['brown'], df['orange'], c=['brown', 'orange'])
plt.xlabel("Brown")
plt.ylabel("Orange")
plt.show()


# ### Importing dataset

# In[6]:


X = df.loc[:, ['gray', 'violet', 'blue', 'green', 'yellow',
'orange', 'red', 'brown']].values.astype(float)
```

```
y = df.loc[:, [‘target’]].values
y = y.ravel()


# ### Splitting the dataset into the Training set and Test set (75%,
25%)

# In[7]:


X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=0)


# ### Feature Scaling

# In[8]:


sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)


# ### Fitting classifier to the Training set

# In[9]:


classifier = KneighborsClassifier(n_neighbors=3, metric=’minkowski’,
p=2)
classifier.fit(X_train, y_train)


# ### Predicting the Test set results

# In[10]:


y_pred = classifier.predict(X_test)
print(classifier.score(X_test, y_test))


# ### The Confusion Matrix
# In[11]:


print(confusion_matrix(y_test, y_pred))


# In[12]:


print(classification_report(y_test, y_pred))


# ### Visualising different values of K for Neighbours and
Overfitting/Underfitting

# In[13]:


# Setup arrays to store train and test accuracies
neighbors = np.arange(1, 9)
train_accuracy = np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

# Loop over different values of k
for ©, k in enumerate(neighbors):
    # Setup a k-NN Classifier with k neighbors: knn
    knn = KneighborsClassifier(n_neighbors=k)

    # Fit the classifier to the training data
    knn.fit(X_train, y_train)

    #Compute accuracy on the training set
    train_accuracy[i] = knn.score(X_train, y_train)

    #Compute accuracy on the testing set
    test_accuracy[i] = knn.score(X_test, y_test)

# Generate plot
plt.title('k-NN: Varying Number of Neighbors')
plt.plot(neighbors, test_accuracy, label = 'Testing Accuracy')
plt.plot(neighbors, train_accuracy, label = 'Training Accuracy')
plt.legend()
plt.xlabel(‘Number of Neighbors’)
plt.ylabel(‘Accuracy’)
plt.show()
```

Decision Tree Classifier

```
# coding: utf-8

# # Performance validation of Electrical Impedance Tomography Images
using Decision Tree Classifier – Machine Learning
#
# ## Copyright © 2018, Faststream Technologies
# ## Author: Sudhanva Narayana

# In[1]:


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report


# ### CURR and PARENT directory constants

# In[2]:


CURR_DIR = os.path.dirname(os.path.abspath('__file__'))
PARENT_DIR = os.path.abspath(os.path.join(CURR_DIR, os.pardir))


# ### Import dataset ignoring headers

# In[3]:


df = pd.read_csv(PARENT_DIR + '\\assets\\datasets\\eit_data.csv',
index_col=[0], header = [0], skiprows= [1] ,skipinitialspace=True)
```

```
# ### Dataset

# In[4]:


df.head()


# ### Visualise the higher intensities

# In[5]:


plt.scatter(df[‘brown’], df[‘orange’], c=[‘brown’, ‘orange’])
plt.xlabel(“Brown”)
plt.ylabel(“Orange”)
plt.show()


# ### Importing dataset

# In[6]:


X = df.loc[:, [‘gray’, ‘violet’, ‘blue’, ‘green’, ‘yellow’,
‘orange’, ‘red’, ‘brown’]].values.astype(float)
y = df.loc[:, [‘target’]].values


# ### Splitting the dataset into the Training set and Test set (75%,
25%)

# In[7]:


X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=0)


# In[8]:


y_train = y_train.ravel()
```

```
# ### Feature Scaling

# In[9]:


sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)


# ### Fitting classifier to the Training set

# In[10]:


classifier = DecisionTreeClassifier(criterion='entropy',
random_state=0)
classifier.fit(X_train, y_train)


# ### Predicting the Test set results

# In[11]:


y_pred = classifier.predict(X_test)

print(classifier.score(X_test, y_test))


# ### The Confusion Matrix

# In[12]:


print(confusion_matrix(y_test, y_pred))


# In[13]:


print(classification_report(y_test, y_pred))
```

Kernel Support Vector Machines

```
# coding: utf-8

# # Performance validation of Electrical Impedance Tomography Images
using Kernal Support Vector Machine – Machine Learning
#
# ## Copyright © 2018, Faststream Technologies
# ## Author: Sudhanva Narayana

# In[1]:


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report


# ### CURR and PARENT directory constants

# In[2]:


CURR_DIR = os.path.dirname(os.path.abspath('__file__'))
PARENT_DIR = os.path.abspath(os.path.join(CURR_DIR, os.pardir))


# ### Import dataset ignoring headers

# In[3]:


df = pd.read_csv(PARENT_DIR + '\\assets\\datasets\\eit_data.csv',
index_col=[0], header = [0], skiprows= [1] ,skipinitialspace=True)
```

```
# ### Dataset

# In[4]:


df.head()


# ### Visualise the higher intensities

# In[5]:


plt.scatter(df[‘brown’], df[‘orange’], c=[‘brown’, ‘orange’])
plt.xlabel(“Brown”)
plt.ylabel(“Orange”)
plt.show()


# ### Importing dataset

# In[6]:


X = df.loc[:, [‘gray’, ‘violet’, ‘blue’, ‘green’, ‘yellow’,
‘orange’, ‘red’, ‘brown’]].values.astype(float)
y = df.loc[:, [‘target’]].values


# ### Splitting the dataset into the Training set and Test set (75%,
25%)

# In[7]:


X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=0)


# In[8]:


y_train = y_train.ravel()
```

```
# ### Feature Scaling

# In[9]:


sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)


# ### Fitting classifier to the Training set

# In[10]:


classifier = SVC(kernel='rbf', random_state=0)
classifier.fit(X_train, y_train)


# ### Predicting the Test set results

# In[11]:


y_pred = classifier.predict(X_test)

print(classifier.score(X_test, y_test))


# ### The Confusion Matrix

# In[12]:


print(confusion_matrix(y_test, y_pred))


# In[13]:


print(classification_report(y_test, y_pred))
```

Logistic Regression Classifier

```
# coding: utf-8

# # Performance validation of Electrical Impedance Tomography Images
using Logistic Regression – Machine Learning
#
# ## Copyright © 2018, Faststream Technologies
# ## Author: Sudhanva Narayana

# In[1]:


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report


# ### CURR and PARENT directory constants

# In[2]:


CURR_DIR = os.path.dirname(os.path.abspath('__file__'))
PARENT_DIR = os.path.abspath(os.path.join(CURR_DIR, os.pardir))


# ### Import dataset ignoring headers

# In[3]:


df = pd.read_csv(PARENT_DIR + '\\assets\\datasets\\eit_data.csv',
index_col=[0], header = [0], skiprows= [1] ,skipinitialspace=True)
```

```
# ### Dataset

# In[4]:


df.head()


# ### Visualise the higher intensities

# In[5]:


plt.scatter(df['brown'], df['orange'], c=['brown', 'orange'])
plt.xlabel("Brown")
plt.ylabel("Orange")
plt.show()


# ### Importing dataset

# In[6]:


X = df.loc[:, ['gray', 'violet', 'blue', 'green', 'yellow',
'orange', 'red', 'brown']].values.astype(float)
y = df.loc[:, ['target']].values
y = y.ravel()


# ### Splitting the dataset into the Training set and Test set (75%,
25%)

# In[7]:


X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=0)


# ### Feature Scaling

# In[8]:
```

```
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)


# ### Fitting classifier to the Training set

# In[9]:


classifier = LogisticRegression(random_state=0)
classifier.fit(X_train, y_train)


# ### Predicting the Test set results

# In[10]:


y_pred = classifier.predict(X_test)

print(classifier.score(X_test, y_test))


# ### The Confusion Matrix

# In[11]:


print(confusion_matrix(y_test, y_pred))


# In[12]:


print(classification_report(y_test, y_pred))
```

Naïve Bayes Classifier

```
# coding: utf-8

# # Performance validation of Electrical Impedance Tomography Images
using © Bayes – Machine Learning
#
# ## Copyright © 2018, Faststream Technologies
# ## Author: Sudhanva Narayana

# In[1]:


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report


# ### CURR and PARENT directory constants

# In[2]:


CURR_DIR = os.path.dirname(os.path.abspath('__file__'))
PARENT_DIR = os.path.abspath(os.path.join(CURR_DIR, os.pardir))


# ### Import dataset ignoring headers

# In[3]:


df = pd.read_csv(PARENT_DIR + '\\assets\\datasets\\eit_data.csv',
index_col=[0], header = [0], skiprows= [1] ,skipinitialspace=True)
```

```
# ### Dataset

# In[4]:


df.head()


# ### Visualise the higher intensities

# In[5]:


plt.scatter(df['brown'], df['orange'], c=['brown', 'orange'])
plt.xlabel("Brown")
plt.ylabel("Orange")
plt.show()


# ### Importing dataset

# In[6]:


X = df.loc[:, ['gray', 'violet', 'blue', 'green', 'yellow',
'orange', 'red', 'brown']].values.astype(float)
y = df.loc[:, ['target']].values


# ### Splitting the dataset into the Training set and Test set (75%,
25%)

# In[7]:


X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=0)


# In[8]:


y_train = y_train.ravel()
```

```
# ### Feature Scaling

# In[9]:


sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)


# ### Fitting classifier to the Training set

# In[10]:


classifier = GaussianNB()
classifier.fit(X_train, y_train)


# ### Predicting the Test set results

# In[11]:


y_pred = classifier.predict(X_test)

print(classifier.score(X_test, y_test))


# ### The Confusion Matrix

# In[12]:


print(confusion_matrix(y_test, y_pred))


# In[13]:


print(classification_report(y_test, y_pred))
```

Random Forest Classifier

```
# coding: utf-8

# # Performance validation of Electrical Impedance Tomography Images
using Random Forest Classifier – Machine Learning
#
# ## Copyright © 2018, Faststream Technologies
# ## Author: Sudhanva Narayana

# In[1]:


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report


# ### CURR and PARENT directory constants

# In[2]:


CURR_DIR = os.path.dirname(os.path.abspath('__file__'))
PARENT_DIR = os.path.abspath(os.path.join(CURR_DIR, os.pardir))


# ### Import dataset ignoring headers

# In[3]:


df = pd.read_csv(PARENT_DIR + '\\assets\\datasets\\eit_data.csv',
index_col=[0], header = [0], skiprows= [1] ,skipinitialspace=True)
```

```
# ### Dataset

# In[4]:


df.head()


# ### Visualise the higher intensities

# In[5]:


plt.scatter(df[‘brown’], df[‘orange’], c=[‘brown’, ‘orange’])
plt.xlabel(“Brown”)
plt.ylabel(“Orange”)
plt.show()


# ### Importing dataset

# In[6]:


X = df.loc[:, [‘gray’, ‘violet’, ‘blue’, ‘green’, ‘yellow’,
‘orange’, ‘red’, ‘brown’]].values.astype(float)
y = df.loc[:, [‘target’]].values


# ### Splitting the dataset into the Training set and Test set (75%,
25%)

# In[7]:


X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=0)


# In[8]:


y_train = y_train.ravel()
```

```
# ### Feature Scaling

# In[9]:


sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)


# ### Fitting classifier to the Training set

# In[10]:


classifier = RandomForestClassifier(n_estimators=10,
criterion=’entropy’, random_state=0)
classifier.fit(X_train, y_train)


# ### Predicting the Test set results

# In[11]:


y_pred = classifier.predict(X_test)

print(classifier.score(X_test, y_test))


# ### The Confusion Matrix

# In[12]:


print(confusion_matrix(y_test, y_pred))


# In[13]:


print(classification_report(y_test, y_pred))
```

```
# ### Visualising different values of N Estimators (Trees)
Overfitting/Underfitting

# In[14]:


# Setup arrays to store train and test accuracies
estimators = np.arange(1, 20)
train_accuracy = np.empty(len(estimators))
test_accuracy = np.empty(len(estimators))

# Loop over different values of k
for ©, k in enumerate(estimators):
    # Setup a RandomForestClassifier
    random_forest = RandomForestClassifier(n_estimators=k)

    # Fit the classifier to the training data
    random_forest.fit(X_train, y_train)

    #Compute accuracy on the training set
    train_accuracy[i] = random_forest.score(X_train, y_train)

    #Compute accuracy on the testing set
    test_accuracy[i] = random_forest.score(X_test, y_test)

# Generate plot
plt.title('Random Forest: Varying Number of Trees')
plt.plot(estimators, test_accuracy, label = 'Testing Accuracy')
plt.plot(estimators, train_accuracy, label = 'Training Accuracy')
plt.legend()
plt.xlabel('Number of Trees')
plt.ylabel('Accuracy')
plt.show()
```

Support Vector Machines

```
# coding: utf-8

# # Performance validation of Electrical Impedance Tomography Images
using Support Vector Machine – Machine Learning
#
# ## Copyright © 2018, Faststream Technologies
# ## Author: Sudhanva Narayana

# In[1]:


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report


# ### CURR and PARENT directory constants

# In[2]:


CURR_DIR = os.path.dirname(os.path.abspath('__file__'))
PARENT_DIR = os.path.abspath(os.path.join(CURR_DIR, os.pardir))


# ### Import dataset ignoring headers

# In[3]:


df = pd.read_csv(PARENT_DIR + '\\assets\\datasets\\eit_data.csv',
index_col=[0], header = [0], skiprows= [1] ,skipinitialspace=True)
```

```
# ### Dataset

# In[4]:


df.head()


# ### Visualise the higher intensities

# In[5]:


plt.scatter(df[‘brown’], df[‘orange’], c=[‘brown’, ‘orange’])
plt.xlabel(“Brown”)
plt.ylabel(“Orange”)
plt.show()


# ### Importing dataset

# In[6]:


X = df.loc[:, [‘gray’, ‘violet’, ‘blue’, ‘green’, ‘yellow’,
‘orange’, ‘red’, ‘brown’]].values.astype(float)
y = df.loc[:, [‘target’]].values


# ### Splitting the dataset into the Training set and Test set (75%,
25%)

# In[7]:


X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=0)


# In[8]:


y_train = y_train.ravel()
```

```
# ### Feature Scaling

# In[9]:


sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)


# ### Fitting classifier to the Training set

# In[10]:


classifier = SVC(kernel='linear', random_state=0)
classifier.fit(X_train, y_train)


# ### Predicting the Test set results

# In[11]:


y_pred = classifier.predict(X_test)

print(classifier.score(X_test, y_test))


# ### The Confusion Matrix

# In[12]:


print(confusion_matrix(y_test, y_pred))


# In[13]:


print(classification_report(y_test, y_pred))
```

10. RESULTS

10. Results

All algorithms are evaluated based on their performance. The results are interpreted and analysed. (Figure 1)

```
# coding: utf-8

# # Performance of various Machine Learning Algorithms on Electrical
Impedance Tomography Images
#
# ## Copyright (c) 2018, Faststream Technologies
#
# ## Author: Sudhanva Narayana

# In[1]:


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

CURR_DIR = os.path.dirname(os.path.abspath('__file__'))
PARENT_DIR = os.path.abspath(os.path.join(CURR_DIR, os.pardir))

df = pd.read_csv(PARENT_DIR + '\\assets\\datasets\\eit_data.csv',
index_col=[0], header = [0], skiprows= [1] ,skipinitialspace=True)
```

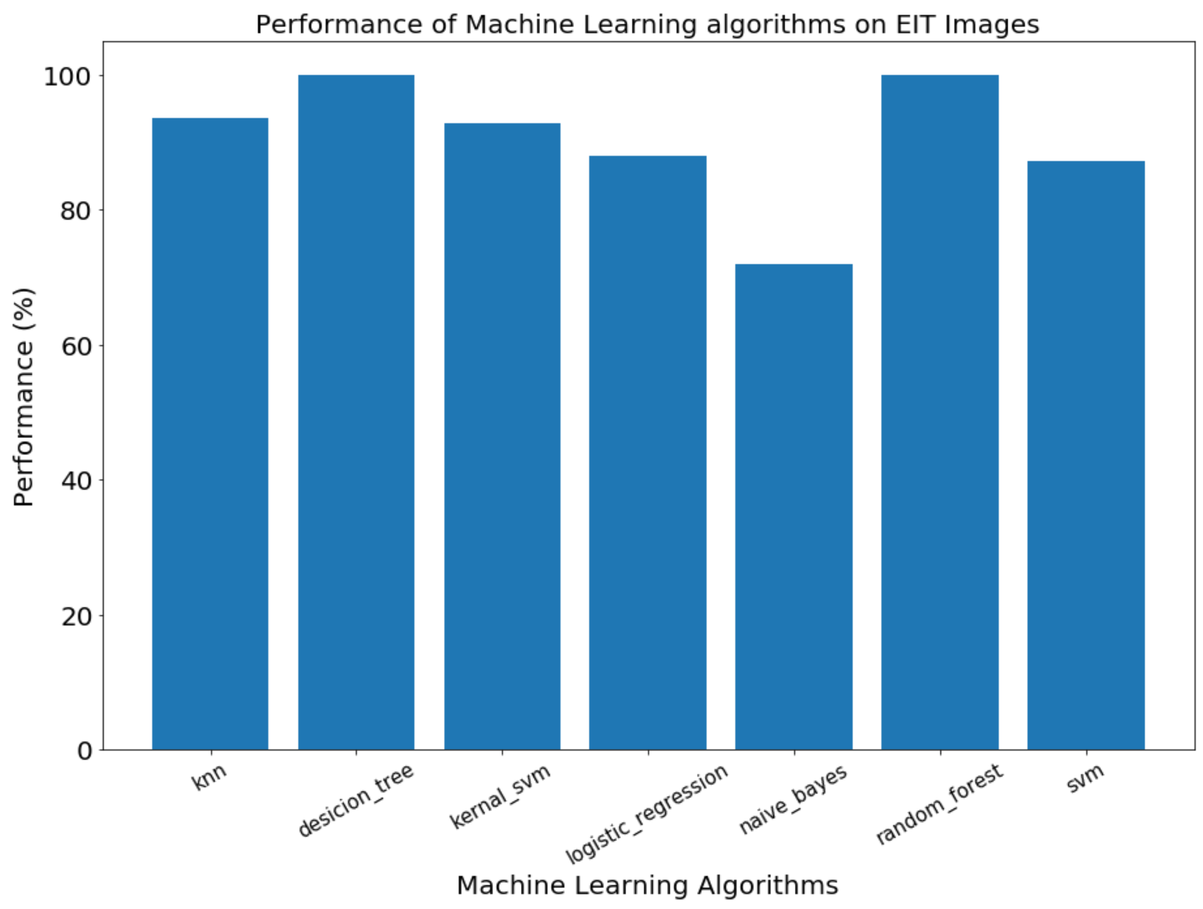
```
X = df.loc[:, ['gray', 'violet', 'blue', 'green', 'yellow',
'orange', 'red', 'brown']].values.astype(float)
y = df.loc[:, ['target']].values

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=0)

y_train = y_train.ravel()

sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)


# ### Classifiers

# In[2]:


classifiers = {}


# ### KNN

# In[3]:


classifier = KNeighborsClassifier(n_neighbors=5, metric='minkowski',
p=2)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

classifiers['knn'] = classifier.score(X_test, y_test)
print(classifier.score(X_test, y_test))


# ### Decision Tree

# In[4]:
```

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

y_pred = classifier.predict(X_test)

classifiers['desicion_tree'] = classifier.score(X_test, y_test)
print(classifier.score(X_test, y_test))


# ### Kernal SVM

# In[5]:


classifier = SVC(kernel='rbf', random_state=0)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

classifiers['kernal_svm'] = classifier.score(X_test, y_test)
print(classifier.score(X_test, y_test))


# ### Logistic Regression

# In[6]:


classifier = LogisticRegression(random_state=0)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

classifiers['logistic_regression'] = classifier.score(X_test,
y_test)
print(classifier.score(X_test, y_test))


# ### Naive Bayes

# In[7]:
```

```
classifier = GaussianNB()
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

classifiers['naive_bayes'] = classifier.score(X_test, y_test)
print(classifier.score(X_test, y_test))


# ### Random Forest

# In[8]:


classifier = RandomForestClassifier(n_estimators=10,
criterion='entropy', random_state=0)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

classifiers['random_forest'] = classifier.score(X_test, y_test)
print(classifier.score(X_test, y_test))


# ### Support Vector Machines

# In[9]:


classifier = SVC(kernel='linear', random_state=0)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

classifiers['svm'] = classifier.score(X_test, y_test)
print(classifier.score(X_test, y_test))


# In[10]:


print(classifiers)
```

```
# In[11]:


values = list(classifiers.values())labels = list(classifiers.keys())

values =  [round(i * 100, 2) for i in values]
# print(values)
# print(labels)

index = np.arange(len(labels))


# In[12]:


plt.figure(figsize=(15,10))
plt.bar(index, values)
plt.xlabel('Machine Learning Algorithms', fontsize=20)
plt.ylabel('Performance (%)', fontsize=20)
plt.xticks(index, labels, rotation=30, fontsize=15)
plt.yticks(fontsize=20)
plt.title('Performance of Machine Learning algorithms on EIT
Images', fontsize=20)
plt.show()
```

Results - Figure 1


Performance of Machine Learning algorithms on EIT Images
Performance (%)
knn
desicion_tree
kernal_svm
logistic_regression
naive_bayes
random_forest
svm
Machine Learning Algorithms