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](mailto: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](mailto:info@faststreamtech.com)

|  |  |
| --- | --- |
| **Table of Contents** | **Page** |
| **Executive Summary** | **No** |
| 1. **Introduction**   *1.1. Background*  *1.2. Electrical Impedance Tomography*  *1.3. Machine Learning*  *1.4. Project Objectives*  *1.5. Dependencies and Tools*  *1.6. Limitations of the project* | **1**  2  3  4  5  6  6 |
| 1. **Project Methodologies**   *2.1. Understanding EIT Images*  *2.2. Identifying pixels by contours*  *2.3. Applying Machine Learning model to predict intensities* | **7**  10  10  11 |
| 1. **Importing, generating and analysing images**   *3.1. Importing and analysing sample images*  *3.2. Generating Images*  *3.3. Classifying images into intensity-based categories* | **12**  13  14  16 |
| 1. **Preparing Dataset**   *4.1. Reading images and counting pixels*  *4.2. Calculating mean pixel count and percentage* | **19**  20  23 |
| 1. **Applying Machine Learning on Dataset**   5.1. Cross validation set approach  *5.2. KNN Classifier*  *5.3. Decision Tree Classifier*  *5.4. Kernel Support Vector Machines*  *5.5. Logistic Regression*  *5.6. Naïve Bayes Classifier*  *5.7. Random Forest Classifier*  *5.8. Support Vector Machines* | **25**  26  26  31  34  37  40  43  47 |
| 1. **Results**   *6.1. Performance of algorithms*  *6.2. Conclusion* | **51**  52  58 |
| **References** | **59** |

**1. INTRODUCTION**

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.

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.

***1.1. 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.

***1.2. Electrical Impedance Tomography***

Electrical impedance tomography (EIT) is a non-invasive type of medical imaging in which the electrical conductivity, permittivity, and impedance of a part of the body is inferred from surface electrode measurements and used to form a tomographic image of that part. Electrical conductivity varies considerably among various biological tissues or the movement of fluids and gases within tissues. The majority of EIT systems apply small alternating currents at a single frequency, however, some EIT systems use multiple frequencies to better differentiate between normal and suspected abnormal tissue within the same organ. Since free ion content determines tissue and fluid conductivity, muscle and blood will conduct the applied currents better than fat, bone or lung tissue. This property can be used to reconstruct static images by morphological or absolute EIT. However, in contrast to linear x-rays used in Computed Tomography, electric currents travel three dimensionally along the path of least resistivity. This means, that a part of the electric current leaves the transverse plane and results in an impedance transfer. This and other factors are the reason why image reconstruction in absolute EIT is so hard, since there is usually more than just one solution for image reconstruction of a three-dimensional area projected onto a two-dimensional plane.

***1.3. 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.

***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 ranged. 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)
  + Locally-Linear Embedding (LLE)
  + t-distributed Stochastic Neighbor Embedding (t-SNE)

***1.4. 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.

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.

**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.*

***1.5. Dependencies and Tools***

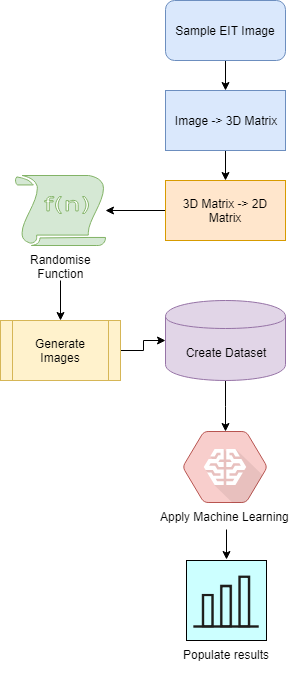
1. Python – a general-purpose interpreted, interactive, object-oriented, and high-level programming language
2. Anaconda – a free and open source distribution of the Python and R programming languages for data science and machine learning related applications, that aims to simplify package management 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. Pandas - pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.
6. Matplotlib – a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms
7. Sci-kit learn – a free software machine learning library for the Python programming language.

***1.6. 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.

**2. PROJECT METHODOLOGIES**

***Steps involved***

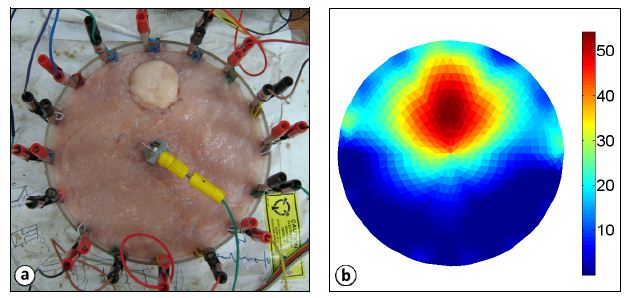


* Image is read into code in the form of a three-dimensional matrix where in each dimension represents intensities of the respective colour code
* This three-dimensional matrix is then converted to two-dimensional matrix (representation of grayscale image) with intensities ranging from 0 to 1
* Image is re generated to observe distribution using contour plots
* 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
* 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

Data is checked for validity, accuracy, completeness and consistency. Data auditing is made to detect anomalies and contradictions. The detection and removal of anomalies is performed by a sequence of operations on the data known as the workflow. 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.

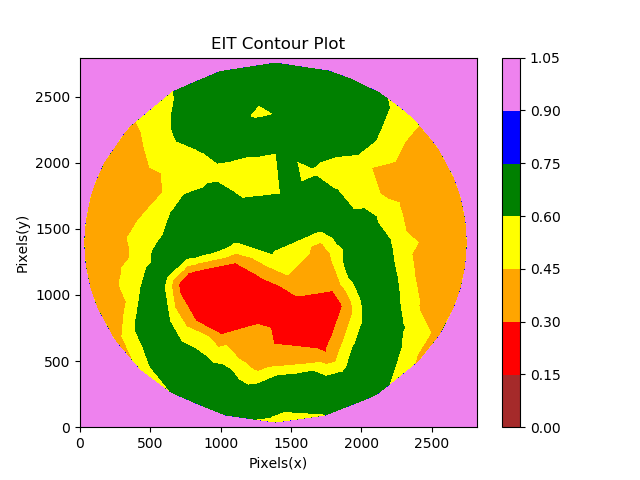
**2.1. Understanding EIT Images**

Electrical Impedance Tomography – images are typically colour based representing the resistive properties on human bodies. The following image (right) is an experimental EIT image generated using electrodes



The resistive properties of the images are assigned colour codes and are plotted.

**2.2. Identifying pixels by contours**

A typical contour image of the same EIT Image looks like this

The colours here are representative and are not related to any colours in resistive properties. The intensity range is from 0 to 1 where 0 being the lowest and 1 being the highest. The x and the y axis are count of pixels

**2.3. Applying Machine Learning model to data**

1. Training set is determined. Here, 75% of the generated dataset is used as training data and the rest 25% is test data.
2. The training set represents the real-world use if the function. A set of input objects is gathered and corresponding outputs are also gathered.
3. 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.
4. The structure of the learned function is determined (any Machine Learning algorithm can be used)
5. 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
6. 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.

**3. IMPORTING, GENERATING AND ANALYSING IMAGES**

**3.1. Importing and analysing sample images**

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()

**3.2. Generating Images**

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’)

**3.3. Classifying images into intensity-based categories**

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()

**4. PREPARING DATASET**

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.

**4.1. Reading images and counting pixels**

# 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’)

**4.2. Calculating mean pixel count and percentage**

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’)

**5. MACHINE LEARNING – APPLICATION**

Cross-validation, sometimes called rotation estimation, or out-of-sample testing is any of various similar model validation techniques for assessing how the results of a statistical analysis will generalize to an independent data set. It is mainly used in the goal prediction, we estimate how accurately a predictive model will perform in practice. In a prediction problem, a model is usually given a dataset of known data on which training is run (training dataset), and a dataset of unknown data (or first seen data) against which the model is tested (called the validation dataset or testing set). The goal of cross-validation is to test the model’s ability to predict new data that were not used in estimating it, in order to flag problems like overfitting and to give an insight on how the model will generalize to an independent dataset (i.e., an unknown dataset, for instance from a real problem).

**5.1. Cross validation set approach**

* 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

***Classification Algorithms***

**5.2. K- Nearest Neighbours Classifier**

The k-nearest-neighbour’s algorithm is a classification algorithm, and it is supervised: it takes a bunch of labelled points and uses them to learn how to label other points. To label a new point, it looks at the labelled points closest to that new point (those are its nearest neighbours), and has those neighbours vote, so whichever label the most of the neighbours have is the label for the new point (the “k” is the number of neighbours it checks).

***Performance: Random Forest showed an accuracy of 93.6%***

# 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()

**5.3. Decision Tree Classifier**

Decision tree builds classification or regression models in the form of a tree structure. It breaks down a data set into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node has two or more branches and a leaf node represents a classification or decision.

***Performance: Decision Tress showed an accuracy of 99.5%***

# 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))

**5.4. Kernel Support Vector Machines**

Kernel support vector machines (SVMs, also support vector networks are supervised learning models with associated learning algorithms that analyse data used for classification and regression analysis.

***Performance: SVM showed an accuracy of 92.8%***

# 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))

**5.5. Logistic Regression Classifier**

It is a statistical method for analysing a dataset in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes). The goal of logistic regression is to find the best fitting model to describe the relationship between the dichotomous characteristic of interest (dependent variable = response or outcome variable) and a set of independent (predictor or explanatory) variables.

***Performance: Logistic Regression showed an accuracy of 88%***

# 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))

**5.6. Naïve Bayes Classifier**

It is a classification technique based on Bayes’ Theorem with an assumption of independence among predictors. Naive Bayes model is easy to build and particularly useful for very large datasets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

***Performance: Naïve Bayes showed an accuracy of 72%***

# 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))

**5.7. Random Forest Classifier**

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees’ habit of overfitting to their training set.

***Performance: Random Forest showed an accuracy of 99.5%***

# 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()

**5.8. Support Vector Machines**

Support vector machines (SVMs, also support vector networks are supervised learning models with associated learning algorithms that analyse data used for classification and regression analysis.

***Performance: SVM showed an accuracy of 87.2%***

# 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))

**6. RESULTS**

**6.1. Performance of algorithms**

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



**Figure 1**

*The results are tabulated below*

|  |  |  |
| --- | --- | --- |
| **No** | **Algorithms** | **Accuracy (Percentage)** |
| 1 | K Nearest Neighbours | 93.6% |
| 2 | Decision Tree Classification | 98.8% |
| 3 | Kernel Support Vector Machines | 94% |
| 4 | Logistic Regression | 88.4% |
| 5 | Naive Bayes | 92.8% |
| 6 | Random Forest Classification | 99.2% |
| 7 | Support Vector Machines | 88% |

# 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()

**6.2. Conclusion**

The performance data is collected based on the confusion matrix produced by the algorithms. In the field of machine learning and specifically the problem of statistical classification, a confusion matrix, also known as an error matrix, is a specific table layout that allows visualization of the performance of an algorithm.

Kernel SVM and K Nearest Neighbours tabulated similar results. Logistic Regression and SVM showcased a significant improvement in the results. However, Decision Tree and Random Forest classifiers performed better than linear classifiers. Random Forest and Decision Tree both topped at 99.5% making them the best algorithm to use for this kind of dataset. Overall the results are similar in nature. We come to the conclusion that for a dataset of 1,000 rows, there are major differences in accuracy on machine learning models.

**References**

1. Aurelien Geron (2017) Hands–On Machine Learning with Scikit–Learn and TensorFlow – 2nd ed.
2. Guoqiang Zhang and B. Eddy Patuwo, Michael Y. Hu (1997), ‘Forecasting with artificial neural networks: The state of the art’, Graduate School of Management, Kent State University, Kent, Ohio 44242-0001, USA
3. S. B. Kotsiantis (2007), ‘Supervised Machine Learning: A Review of Classification Techniques’, Department of Computer Science and Technology, University of Peloponnese, Greece, End of Karaiskaki, 22100, Tripolis GR.
4. Amento, B., Hill, W., and Terveen, L. The sound of one hand: a wrist-mounted bio-acoustic fingertip gesture interface.In Proc. CHI '02, 724-725.
5. Amma, C. Krings, T., Böer, J. and Schultz, T. Advancing Muscle-Computer Interfaces with High-Density. Electromyography. In Proc. CHI '15, 929-938.
6. Beck, M. S., Dyakowski, T and William, R. A. Process, Tomography–The State of Art. Transactions of the Institute of Measurement and Control. 20(4), 163-177. Oct. 1998
7. Robert W. Stacey, Electrical Impedance Tomography, SGP-TR-182, Stanford University (2016)
8. Grzegorz Klosowski, Tomasz Rymarczyk, Using neural networks and deep learning algorithms in electrical impedance tomography, Lublin University of Technology, Lublin (2017)
9. Eoghan Dunne and Adam Santorelli, Supervised Learning Classifiers for Electrical Impedance-based BladderState Detection (2018)