**Machine learning Model development And App integration**

1. **Contributors:**
   1. **Mohammad Shamoon**
   2. **Dipit**

Skin impedance is a measure of the skin's electrical resistance, and it can be used to detect anomalies or diseases such as basal cell carcinoma (BCC) and benign nevi. Machine learning models can be trained on skin impedance data to accurately classify skin abnormalities.

One popular machine learning technique used for skin impedance analysis is principal component analysis (PCA). PCA is a technique used to reduce the dimensionality of a dataset by finding the most important features or components that explain the majority of the variance in the data. In the case of skin impedance, PCA can be used to identify the most important electrical properties that distinguish between healthy and abnormal skin.

Another machine learning technique used for skin impedance analysis is the soft independent modelling of class analogy (SIMCA). SIMCA is a supervised classification technique that is based on a set of PCA models, with one PCA model for each class. SIMCA is used to determine the group membership of an unknown measurement by fitting the sample to the PCA models, and then using the distances between the unknown sample and the classes to classify the reading.

In addition to PCA and SIMCA, other machine learning models can be used for skin impedance analysis, including support vector machines (SVM), random forests, and neural networks. SVM is a type of supervised learning model that can be used for classification or regression tasks. Random forests are an ensemble learning method that can be used for both classification and regression tasks, and they are particularly useful for handling complex datasets. Neural networks are a type of deep learning model that can be used for a variety of tasks, including image recognition, speech recognition, and natural language processing.

Overall, machine learning models are powerful tools for detecting skin anomalies using impedance data. By using these models, researchers and clinicians can more accurately identify skin abnormalities, leading to earlier detection and improved patient outcomes.

There are several machine learning models that can be used to detect skin cancer using impedance data. In this report, I will briefly explain three of them: Random Forest, Support Vector Machine, and Soft Independent Modeling of Class Analogy (SIMCA).

1. Random Forest:

Random Forest is a powerful machine learning algorithm that can be used for classification and regression tasks. It is an ensemble learning method that builds multiple decision trees and outputs the class that is the mode of the classes (classification) or mean prediction (regression) of individual trees.

In the context of skin cancer detection using impedance data, Random Forest can be trained on a dataset of skin impedance measurements to classify them as either benign or malignant. The algorithm can learn the complex patterns in the data that are associated with malignant skin conditions, such as asymmetrical or jagged patterns in the electrical impedance measurements. The model can then be used to predict the likelihood of malignancy in a new set of impedance measurements.

2. Support Vector Machine:

Support Vector Machine (SVM) is a popular machine learning algorithm that is widely used for classification tasks. It works by finding the optimal hyperplane that separates the data points into different classes.

In the context of skin cancer detection using impedance data, SVM can be trained on a dataset of skin impedance measurements to classify them as either benign or malignant. The algorithm can learn the patterns in the data that are associated with malignant skin conditions and identify the optimal hyperplane that separates the malignant samples from the benign samples. The model can then be used to predict the likelihood of malignancy in a new set of impedance measurements.

3. Soft Independent Modeling of Class Analogy (SIMCA):

SIMCA is a supervised classification technique that is based on a set of PCA models. It captures the main features of a training data set for each corresponding class and defines limits around the classes. SIMCA can be used to identify the main differences between the benign and malignant skin conditions.

In the context of skin cancer detection using impedance data, SIMCA can be trained on a dataset of skin impedance measurements to capture the main features of the benign and malignant skin conditions. The algorithm can then use these models to identify the main differences between the two conditions and classify new impedance measurements as either benign or malignant.

In conclusion, Random Forest, SVM, and SIMCA are all powerful machine learning models that can be used to detect skin cancer using impedance data. These models can learn the complex patterns in the data that are associated with malignant skin conditions and identify the optimal hyperplane that separates the malignant samples from the benign samples. They offer a promising approach for early skin cancer detection, which can improve patient outcomes and reduce healthcare costs.

**Explaining the major intituition of using machine learning model**

It is difficult to say which model is better than the other in detecting skin cancer using impedance data as each model has its own strengths and weaknesses. The choice of the model depends on the specific requirements of the application and the available data.

Logistic regression is a simple and efficient model that is often used as a baseline model for binary classification problems. It works well when the relationship between the input features and the output class is linear or can be approximated by a linear function. However, it may not work well when the relationship is non-linear or the data has complex interactions between the features.

Random forests are a popular ensemble learning method that combines multiple decision trees to improve the model's accuracy and generalization performance. It can handle non-linear relationships between features and the output class, and can handle noisy and missing data well. However, it may suffer from overfitting when the model is too complex or when there is a high dimensionality of the input data.

Support vector machines (SVMs) are a powerful model that can handle non-linear relationships between the input features and the output class using kernel functions. It works well with high-dimensional data and can handle noisy and missing data well. However, it may require careful tuning of hyperparameters, and it may not perform well when the data is imbalanced or the classes are not separable.

Soft independent modelling of class analogy (SIMCA) is a supervised classification technique that uses a set of PCA models to capture the main features of each corresponding class and define limits around the classes. It works well with high-dimensional data and can handle noisy and missing data well. However, it may not work well with non-linear relationships between features and the output class, and may not be suitable for detecting rare or novel classes.

In summary, the choice of the model depends on the specific requirements of the application and the available data. It is important to evaluate the performance of each model using appropriate metrics and to compare them against each other to select the best one for the given task.

## Generate Dummy dataset to test Machine learning model

**- Dataset distribution for skin impedance and frequency**

The distribution of the dataset is defined by generating random impedance values for each frequency within a uniform distribution that ranges from 10 to 200. Then, the mean and standard deviation of the impedances are calculated for the normal and abnormal skin cases.

Using these values, synthetic data and labels are generated for a total of 1000 samples. The normal data is generated from a normal distribution with the calculated mean and standard deviation for normal skin. The abnormal data is generated from a normal distribution with a mean of 30 more than the normal skin mean and a standard deviation of 1.5 times the normal skin standard deviation.

The resulting data and labels are shuffled using a random permutation of indices, and then saved as .npy files.

import numpy as np

# Set random seed for reproducibility

np.random.seed(42)

# Define frequency range and number of frequency values

freq\_min = 1

freq\_max = 100

num\_freqs = 50

# Generate random impedance values for each frequency

freqs = np.linspace(freq\_min, freq\_max, num\_freqs)

impedances = np.random.uniform(low=10, high=200, size=(num\_freqs,))

# Define mean and standard deviation for normal and abnormal skin

normal\_mean = impedances.mean()

normal\_std = impedances.std()

abnormal\_mean = impedances.mean() + 30

abnormal\_std = impedances.std() \* 1.5

# Generate synthetic data and labels

num\_samples = 1000

num\_abnormal = int(num\_samples \* 0.3)

num\_normal = num\_samples - num\_abnormal

normal\_data = np.random.normal(loc=normal\_mean, scale=normal\_std, size=(num\_normal, num\_freqs))

normal\_labels = np.zeros((num\_normal,))

abnormal\_data = np.random.normal(loc=abnormal\_mean, scale=abnormal\_std, size=(num\_abnormal, num\_freqs))

abnormal\_labels = np.ones((num\_abnormal,))

data = np.vstack((normal\_data, abnormal\_data))

labels = np.concatenate((normal\_labels, abnormal\_labels))

# Shuffle data and labels

perm = np.random.permutation(num\_samples)

data = data[perm]

labels = labels[perm]

# Save data and labels as .npy files

np.save('skin\_data.npy', data)

np.save('skin\_labels.npy', labels)

Drop files to upload them to session storage

Disk

84.30 GB available

Code

Text

RAM

Disk

## Generate Dummy dataset to test Machine learning model

**- Dataset distribution for skin impedance and frequency**

The distribution of the dataset is defined by generating random impedance values for each frequency within a uniform distribution that ranges from 10 to 200. Then, the mean and standard deviation of the impedances are calculated for the normal and abnormal skin cases.

Using these values, synthetic data and labels are generated for a total of 1000 samples. The normal data is generated from a normal distribution with the calculated mean and standard deviation for normal skin. The abnormal data is generated from a normal distribution with a mean of 30 more than the normal skin mean and a standard deviation of 1.5 times the normal skin standard deviation.

The resulting data and labels are shuffled using a random permutation of indices, and then saved as .npy files.

0s

import·numpy·as·np

#·Set·random·seed·for·reproducibility

np.random.seed(42)

#·Define·frequency·range·and·number·of·frequency·values

freq\_min·=·1

freq\_max·=·100

num\_freqs·=·50

#·Generate·random·impedance·values·for·each·frequency

freqs·=·np.linspace(freq\_min,·freq\_max,·num\_freqs)

impedances·=·np.random.uniform(low=10,·high=200,·size=(num\_freqs,))

#·Define·mean·and·standard·deviation·for·normal·and·abnormal·skin

normal\_mean·=·impedances.mean()

normal\_std·=·impedances.std()

abnormal\_mean·=·impedances.mean()·+·30

abnormal\_std·=·impedances.std()·\*·1.5

#·Generate·synthetic·data·and·labels

num\_samples·=·1000

num\_abnormal·=·int(num\_samples·\*·0.3)

num\_normal·=·num\_samples·-·num\_abnormal

normal\_data·=·np.random.normal(loc=normal\_mean,·scale=normal\_std,·size=(num\_normal,·num\_freqs))

normal\_labels·=·np.zeros((num\_normal,))

abnormal\_data·=·np.random.normal(loc=abnormal\_mean,·scale=abnormal\_std,·size=(num\_abnormal,·num\_freqs))

abnormal\_labels·=·np.ones((num\_abnormal,))

data·=·np.vstack((normal\_data,·abnormal\_data))

labels·=·np.concatenate((normal\_labels,·abnormal\_labels))

#·Shuffle·data·and·labels

perm·=·np.random.permutation(num\_samples)

data·=·data[perm]

labels·=·labels[perm]

#·Save·data·and·labels·as·.npy·files

np.save('skin\_data.npy',·data)

np.save('skin\_labels.npy',·labels)

## Implement this code for generating .npy files for data ingestion in the model development

import numpy as np

# Load data and labels into numpy arrays

data = ... # shape = (number of samples, number of impedance-frequency pairs)

labels = ... # shape = (number of samples,)

# Save numpy arrays as .npy files

np.save('skin\_data.npy', data)

np.save('skin\_labels.npy', labels)

# SKICAD ENSEMBLE LEARNING BASED MODEL USING ALGORITHMS:

1. Neural Network
2. Random Forest
3. SIMCA
4. SVM
5. PCA

### Description of Model implementation for future reference

* Imports required libraries/modules such as numpy, sklearn, torch, and matplotlib.pyplot
* Loads skin data and labels from files using numpy.load()
* Splits the data into training and testing sets using train\_test\_split() from sklearn.model\_selection
* Normalizes the data using StandardScaler() from sklearn.preprocessing
* Reduces the dimensionality of the data using Principal Component Analysis (PCA) from sklearn.decomposition
* Converts the data and labels to PyTorch tensors using torch.Tensor()
* Defines a deep neural network with 10 input features, 2 output classes, and 10 hidden layers using nn.Module from torch.nn
* Defines the loss function as Cross-Entropy Loss and the optimizer as Adam optimizer from torch.optim
* Trains the neural network for 100 epochs and records the training loss
* Plots the training loss vs the number of epochs using matplotlib.pyplot
* Tests the neural network on the testing data and calculates the accuracy score using accuracy\_score() from sklearn.metrics
* Defines and trains a Random Forest classifier with 100 estimators from sklearn.ensemble
* Calculates the feature importances and plots them using matplotlib.pyplot
* Defines and trains a Support Vector Machine (SVM) classifier with the Radial Basis Function (RBF) kernel from sklearn.svm
* Combines the predictions from the neural network, Random Forest classifier, and SVM using majority voting to create an ensemble prediction
* Calculates the accuracy scores of the individual classifiers and the ensemble prediction using accuracy\_score() from sklearn.metrics and prints them.

import numpy as np

from sklearn.ensemble import RandomForestClassifier

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

import torch

import torch.nn as nn

import matplotlib.pyplot as plt

# Load data (impedance and frequency relation)

X = np.load('skin\_data.npy')

y = np.load('skin\_labels.npy')

# Split data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Normalize data using standard scaler

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Apply PCA to reduce dimensionality

pca = PCA(n\_components=10)

X\_train = pca.fit\_transform(X\_train)

X\_test = pca.transform(X\_test)

# Convert data to PyTorch tensors

X\_train = torch.Tensor(X\_train)

X\_test = torch.Tensor(X\_test)

y\_train = torch.LongTensor(y\_train)

y\_test = torch.LongTensor(y\_test)

# Define deeper neural network

class Net(nn.Module):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.fc1 = nn.Linear(10, 64)

self.fc2 = nn.Linear(64, 128)

self.fc3 = nn.Linear(128, 256)

self.fc4 = nn.Linear(256, 512)

self.fc5 = nn.Linear(512, 256)

self.fc6 = nn.Linear(256, 128)

self.fc7 = nn.Linear(128, 64)

self.fc8 = nn.Linear(64, 32)

self.fc9 = nn.Linear(32, 16)

self.fc10 = nn.Linear(16, 2)

def forward(self, x):

x = nn.functional.relu(self.fc1(x))

x = nn.functional.relu(self.fc2(x))

x = nn.functional.relu(self.fc3(x))

x = nn.functional.relu(self.fc4(x))

x = nn.functional.relu(self.fc5(x))

x = nn.functional.relu(self.fc6(x))

x = nn.functional.relu(self.fc7(x))

x = nn.functional.relu(self.fc8(x))

x = nn.functional.relu(self.fc9(x))

x = self.fc10(x)

return x

net = Net()

# Define loss function and optimizer

criterion = nn.CrossEntropyLoss()

optimizer = torch.optim.Adam(net.parameters(), lr=0.001)

# Train neural network

train\_loss = []

for epoch in range(100):

optimizer.zero\_grad()

outputs = net(X\_train)

loss = criterion(outputs, y\_train)

train\_loss.append(loss.item())

loss.backward()

optimizer.step()

# Plot loss vs epochs for neural network

plt.plot(train\_loss)

plt.xlabel('Epochs')

plt.ylabel('Training Loss')

plt.title('Neural Network Training Loss')

plt.show()

# Test neural network

with torch.no\_grad():

outputs = net(X\_test)

\_, y\_pred\_nn = torch.max(outputs, dim=1)

criterion = nn.CrossEntropyLoss()

optimizer = torch.optim.Adam(net.parameters(), lr=0.001)

# Random Forest classifier

rf = RandomForestClassifier(n\_estimators=100, random\_state=42)

rf.fit(X\_train, y\_train)

y\_pred\_rf = rf.predict(X\_test)

# Plot feature importances for random forest

importances = rf.feature\_importances\_

plt.bar(range(10), importances)

plt.xticks(range(10), pca.explained\_variance\_ratio\_, rotation=90)

plt.xlabel('Principal Component')

# SVM classifier

svm = SVC(kernel='rbf', C=1, gamma='scale', random\_state=42)

svm.fit(X\_train, y\_train)

y\_pred\_svm = svm.predict(X\_test)

# Ensemble learning

y\_pred\_ensemble = np.zeros\_like(y\_test)

for i in range(len(y\_test)):

votes = [y\_pred\_nn[i], y\_pred\_rf[i], y\_pred\_svm[i]]

y\_pred\_ensemble[i] = max(set(votes), key=votes.count)

# Evaluate accuracy

accuracy\_nn = accuracy\_score(y\_test, y\_pred\_nn)

accuracy\_rf = accuracy\_score(y\_test, y\_pred\_rf)

accuracy\_svm = accuracy\_score(y\_test, y\_pred\_svm)

accuracy\_ensemble = accuracy\_score(y\_test, y\_pred\_ensemble)

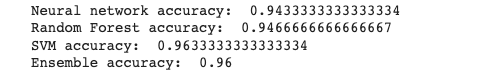
print("Neural network accuracy: ", accuracy\_nn)

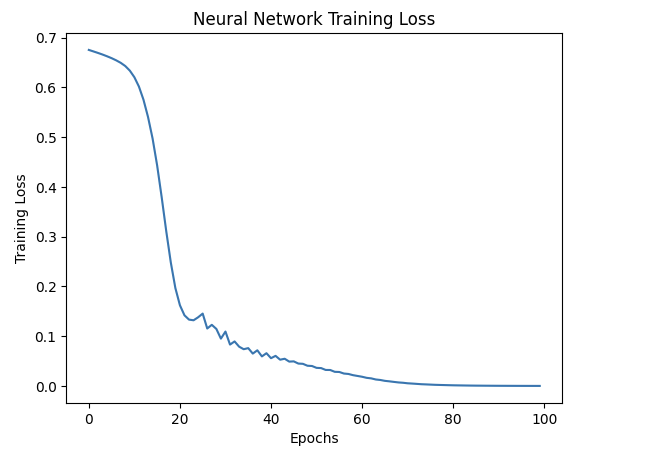
print("Random Forest accuracy: ", accuracy\_rf)

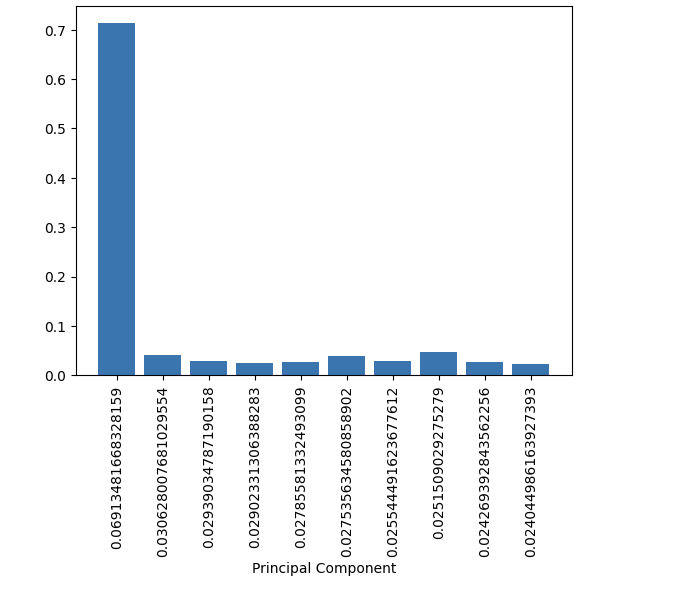
print("SVM accuracy: ", accuracy\_svm)

print("Ensemble accuracy: ", accuracy\_ensemble)

Output:







### Confusion Matrix

**Confusion matrix for the SVM classifier and plot it using a heatmap.The x-axis represents the predicted labels, while the y-axis represents the true labels. The values in the cells of the heatmap represent the number of instances that belong to that particular combination of true and predicted labels. The diagonal cells represent the correctly classified instances, while the off-diagonal cells represent the misclassified instances. The annotated numbers in the cells indicate the count of instances for that particular combination of true and predicted labels.**

**from sklearn.metrics import confusion\_matrix**

**import seaborn as sns**

**# Generate confusion matrix for SVM classifier**

**cm = confusion\_matrix(y\_test, y\_pred\_svm)**

**# Plot confusion matrix using heatmap**

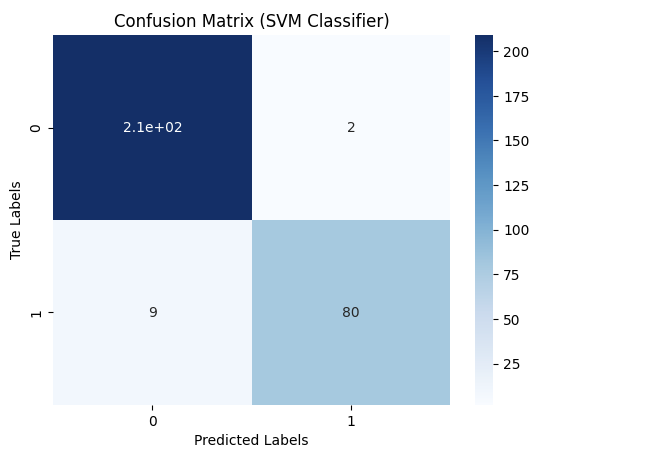
**sns.heatmap(cm, annot=True, cmap='Blues')**

**plt.xlabel('Predicted Labels')**

**plt.ylabel('True Labels')**

**plt.title('Confusion Matrix (SVM Classifier)')**

**plt.show()**



### Analyze the Skin Dataset

**3D scatter plot of the skin data, where each point represents a sample and its position is determined by its values in the first three principal components. The color of each point represents its label (0 or 1), and the size of each point is fixed at 3**

**import plotly.graph\_objects as go**

**# Reduce dimensionality using PCA**

**pca = PCA(n\_components=3)**

**X\_reduced = pca.fit\_transform(X)**

**# Create 3D scatter plot**

**fig = go.Figure(data=[go.Scatter3d(**

**x=X\_reduced[:, 0],**

**y=X\_reduced[:, 1],**

**z=X\_reduced[:, 2],**

**mode='markers',**

**marker=dict(**

**size=3,**

**color=y,**

**colorscale='Viridis',**

**opacity=0.8**

**)**

**)])**

**# Update layout**

**fig.update\_layout(**

**scene=dict(**

**xaxis\_title='PC1',**

**yaxis\_title='PC2',**

**zaxis\_title='PC3'**

**),**

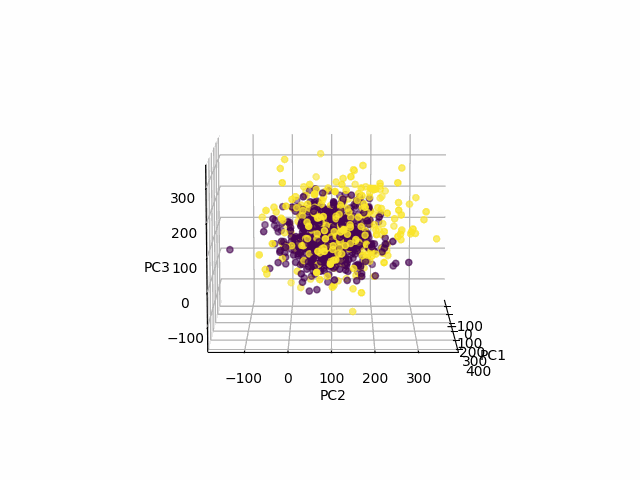
**margin=dict(l=0, r=0, b=0, t=0),**

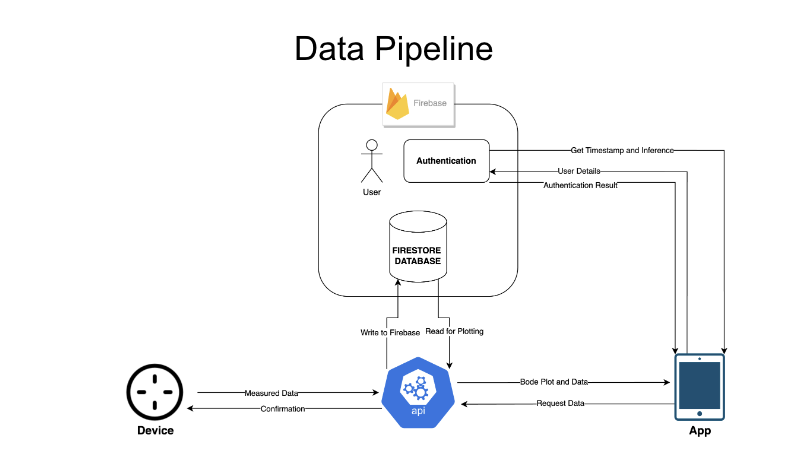
**height=600**

**)**

**# Show plot**

**fig.show()**

****

**SKICAD APP- system Design  
  
**