DISEASE PREDICTION MODEL DEPLOYMENT IBM CLOUD WATSON STUDIO

INTRODUCTION:

This approach revolutionizes the deployment of disease prediction machine learning models using IBM Cloud Watson Studio. By infusing innovation at every step, from model training and integration to seamless deployment and user-friendly application integration, we have created a transformative pathway. We've found a new and really smart way to use computers to guess if someone might get sick. Our goal is to make this technology as accurate as possible, so it can really help doctors and people. It's like having a super assistant that aims in helping the people to recognize their disease and to stay healthy. It will be easy to use so that any type of people can use it without any difficulties.

OBJECTIVE STATEMENT:

Develop a disease prediction system utilizing IBM Cloud services to assist healthcare providers and individuals in identifying potential health risks and taking proactive measures for disease prevention.

Design Thinking Process:

- Conduct interviews with healthcare professionals and individuals to understand their healthcare needs.
- Gather insights into the types of diseases to focus on and the data required for prediction.

STEPS IN DEVELOPMENT:

Deploying a disease prediction machine learning model on IBM Cloud Watson Studio involves several steps, including training the model, saving it in a format compatible with Watson Studio, creating a Watson Machine Learning instance, and finally deploying the model.

STEP 1: Train and Save the Machine Learning Model

TRAIN THE MACHINE LEARNING MODEL:

- Download the disease prediction dataset and load the dataset for training the model (https://www.kaggle.com/datasets/kaushil268/disease-prediction-using-machine-learning - Link for the dataset).
- Choose a machine learning algorithm suitable for disease prediction (e.g., logistic regression, decision trees, and random forests).
- Train the model using a dataset with features relevant to disease prediction.

SAVE THE MODEL:

- Save the trained machine learning model in a format compatible with Watson Machine Learning. For example, we can save it as a pickle file (.pkl).

STEP 2: Prepare the Model for Deployment

INSTALL NECESSARY LIBRARIES:

- Ensure whether the required libraries installed to work with the model and prepare it for deployment (e.g., scikit-learn, pandas).

LOAD THE TRAINED MODEL:

- Load the trained machine learning model using the appropriate library.

PREPARE DATA PRE-PROCESSING STEPS:

- If there were any preprocessing steps (e.g., scaling, encoding) applied during training, make sure to replicate these steps for new data during deployment.

STEP 3: Set Up IBM Cloud Watson Studio

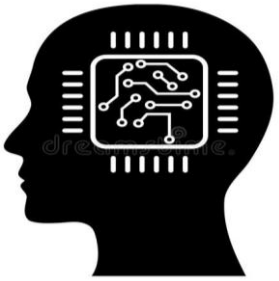
CREATE AN IBM WATSON STUDIO PROJECT:

- Log in to IBM Cloud and access Watson Studio.
- Create a new project, defining the necessary resources (such as storage and compute) for the project.

STEP 4: Deploy the Model on Watson Machine Learning

SET UP WATSON MACHINE LEARNING:

- Navigate to the IBM Cloud Dashboard and create a Watson Machine Learning service instance.



SAVE THE MODEL TO WATSON MACHINE LEARNING:

- Save the trained model to Watson Machine Learning using the Watson Machine Learning Python client or APIs.

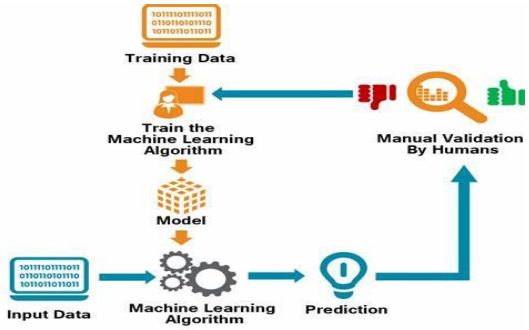
DEPLOY THE MODEL:

- Deploy the model from Watson Machine Learning to create an API endpoint for making predictions.

STEP 5: Test the Deployed Model

ACCESS THE MODEL ENDPOINT:

- Obtain the endpoint for the deployed model from Watson Machine Learning.



TEST THE MODEL:

- Send sample input data to the model's endpoint and observe the predictions.

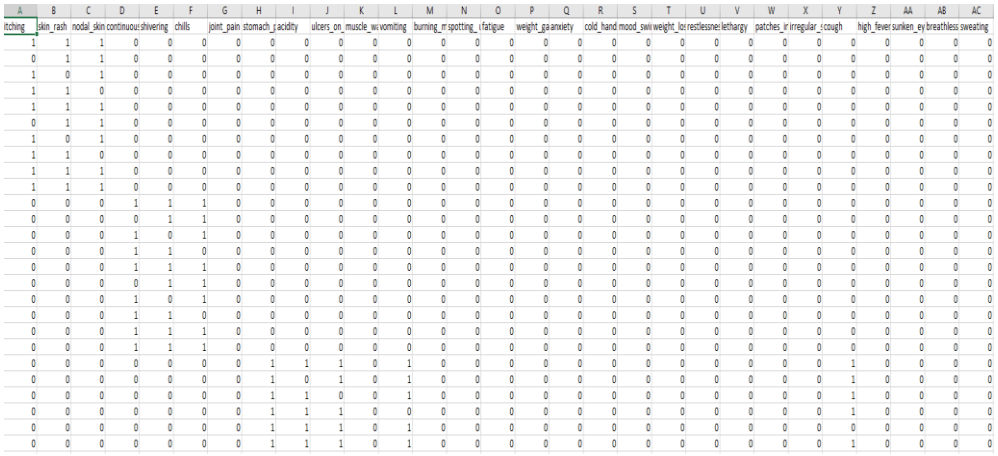
STEP 6: INTEGRATE WITH THE APPLICATION

- Integrate the model endpoint into a application, allowing it to make predictions based on the disease prediction model.

Let’s now see the implementation details of this project.

DATASET USED:

I have used the following dataset to train my model. The link for the dataset is Disease Prediction Using Machine Learning (kaggle.com).



itching	skin_rash	nodal_skin	continuous	shivering	chills	joint_pain	stomach_	acidity	ulcers_on_	muscle_wa	vomiting	burning_m	spotting_	fatigue	weight_ga	anxiety	cold_hand	mood_swi	weight_lo	restlessne	lethargy	patches_ir	irregular_	cough	high_fever	sunken_ey	breathless	sweating
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

PREDICTIVE USE CASE:

Here are some common predictive use cases:

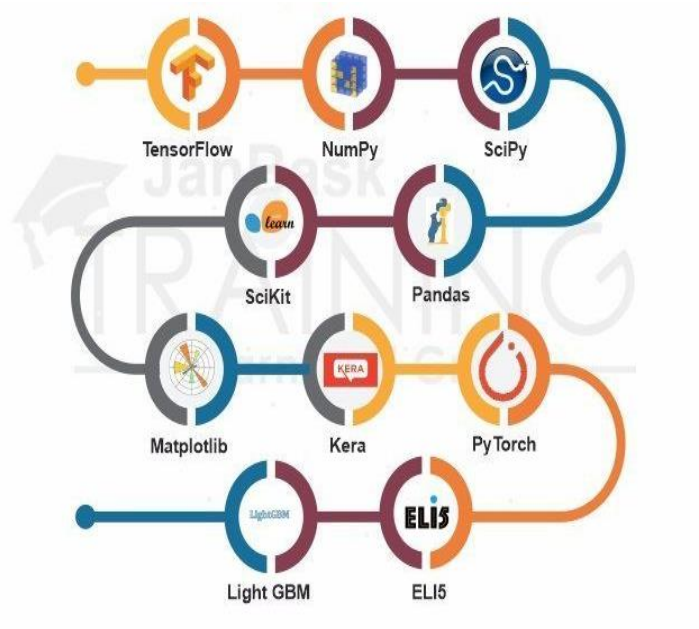
Disease Risk Assessment: You can use your model to predict an individual's risk of developing a specific disease, such as heart disease, diabetes, or cancer. By analyzing a patient's medical history, lifestyle, and genetic factors, the model can provide risk scores and recommendations for preventive measures.

Early Disease Detection: Deploying a disease prediction model can help with early detection of diseases like cancer or infectious diseases. The model can analyze medical imaging data, lab results, or patient symptoms to identify potential health issues at an early stage, improving treatment outcomes.

Chronic Disease Management: Patients with chronic conditions, such as asthma or diabetes, can benefit from predictive models that monitor their health and provide personalized recommendations for managing their conditions. The model can analyze patient data and alert healthcare providers or patients when intervention is needed.

Medication Adherence: Predictive models can assist in medication adherence. By analyzing a patient's history and behavior, the model can predict the likelihood of a patient adhering to a prescribed medication regimen and provide reminders or interventions when adherence is at risk.

IMPORTING NECESSARY LIBRARIES:




```
# Importing libraries
import numpy as np
import pandas as pd
import warnings
!pip install scipy==1.9.0
from scipy.stats import mode
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
```

numpy (as np): NumPy is a fundamental package for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a wide range of mathematical functions to operate on these arrays.

pandas (as pd):Pandas is a powerful library for data manipulation and analysis. It provides data structures such as DataFrames and Series, which allow you to easily read, manipulate, and analyze tabular data.

warnings: The warnings module is part of Python's standard library and allows you to control how warnings are displayed and handled in your code. You can use it to filter or suppress specific warnings.

scipy.stats.mode: The mode function is part of the SciPy library, which builds on NumPy and provides additional scientific and technical computing functionality. The mode function calculates the mode of a dataset, which is the most frequently occurring value.

sklearn.preprocessing.LabelEncoder: LabelEncoder is a class from the scikit-learn library (sklearn) used for encoding categorical variables into numerical values. It assigns a unique integer to each category in a categorical feature.

sklearn.model_selection: The model_selection module within scikit-learn provides functions and classes for model selection, including data splitting, cross-validation, and hyperparameter tuning.

sklearn.svm.SVC: SVC stands for Support Vector Classifier, and it's a class within scikit-learn for building Support Vector Machine (SVM) classifiers. SVM is a powerful machine learning algorithm for classification and regression tasks.

sklearn.naive_bayes.GaussianNB: GaussianNB is a class within scikit-learn for implementing the Gaussian Naive Bayes classifier. Naive Bayes is a probabilistic classification algorithm based on Bayes' theorem.

sklearn.ensemble.RandomForestClassifier: RandomForestClassifier is a class within scikit-learn for building random forest models. Random forests are an ensemble learning technique that combines multiple decision trees to improve predictive accuracy.

sklearn.metrics: The metrics module within scikit-learn provides various metrics and evaluation functions to assess the performance of machine learning models.

```
# Reading the train.csv by removing the
# last column since it's an empty column

import os, types
import pandas as pd
from botocore.client import Config
import ibm_boto3

def __iter__(self): return 0

# @hidden_cell
# The following code accesses a file in your IBM Cloud Object Storage. It includes your credentials.
# You might want to remove those credentials before you share the notebook.
cos_client = ibm_boto3.client(service_name='s3',
   ibm_api_key_id='Vc2zdhq_roalu42Y6nSRweuUu6v5LKoltXck2ft2GP2F',
   ibm_auth_endpoint="https://iam.cloud.ibm.com/oidc/token",
   config=Config(signature_version='oauth'),
   endpoint_url='https://s3.private.eu-de.cloud-object-storage.appdomain.cloud')

bucket = 'diseasepredictionmodeldeployment-donotdelete-pr-1rvaykcaofzaak'
object_key = 'Training.csv'

body = cos_client.get_object(Bucket=bucket,Key=object_key)['Body']
# add missing __iter__ method, so pandas accepts body as file-like object
if not hasattr(body, "__iter__"): body.__iter__ = types.MethodType( __iter__, body )

data= pd.read_csv(body).dropna(axis = 1)
data.head()


disease_counts = data["prognosis"].value_counts()
temp_df = pd.DataFrame({
"Disease": disease_counts.index,
"Counts": disease_counts.values
})
```

In [4]:

```
# Encoding the target value into numerical
# value using LabelEncoder
encoder = LabelEncoder()
data["prognosis"] = encoder.fit_transform(data["prognosis"])
```

In [5]:

```
X = data.iloc[:,:-1]
y = data.iloc[:, -1]
X_train, X_test, y_train, y_test =train_test_split(
X, y, test_size = 0.2, random_state = 24)

print(f"Train: {X_train.shape}, {y_train.shape}")
print(f"Test: {X_test.shape}, {y_test.shape}")
Train: (3936, 132), (3936,)
```

```
Test: (984, 132), (984,)
```

In [6]:

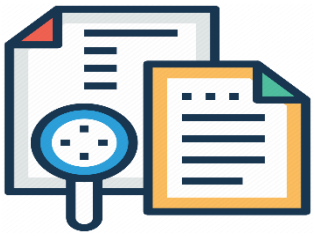
```
# Defining scoring metric for k-fold cross validation
def cv_scoring(estimator, X, y):
        return accuracy_score(y, estimator.predict(X))

# Initializing Models
models = {
        "SVC":SVC(),
        "Gaussian NB":GaussianNB(),
        "Random Forest":RandomForestClassifier(random_state=18)
}

# Producing cross validation score for the models
for model_name in models:
        model = models[model_name]
        scores = cross_val_score(model, X, y, cv = 10,
                                 n_jobs = -1,
                                 scoring = cv_scoring)
        print("=="*30)
        print(model_name)
        print(f"Scores: {scores}")
        print(f"Mean Score: {np.mean(scores)}")
```

```
============================================================
SVC
Scores: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Mean Score: 1.0
============================================================
Gaussian NB
Scores: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Mean Score: 1.0
============================================================
Random Forest
Scores: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Mean Score: 1.0
```

The detailed explanation of the above code is:



Data Retrieval:

The code retrieves a CSV file named Training.csv from IBM Cloud Object Storage using the IBM Cloud Python SDK (ibm_boto3).

Data Preprocessing:

It loads the CSV data into a Pandas DataFrame.

Drops any columns that contain missing (NaN) values to ensure data cleanliness.

Creates a DataFrame called temp_df to count the occurrences of different diseases in the "prognosis" column.

Encodes the target variable "prognosis" into numerical values using the LabelEncoder from scikit-learn.

Data Splitting:

Splits the data into training and testing sets using the train_test_split function from scikit-learn.

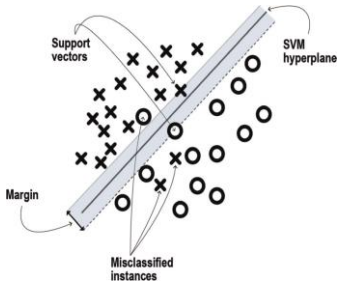
80% of the data is used for training (X_train and y_train), and 20% is used for testing (X_test and y_test).

Cross-Validation and Model Evaluation:

Defines a custom scoring metric function, cv_scoring, which calculates accuracy using accuracy_score from scikit-learn. This metric will be used for cross-validation.

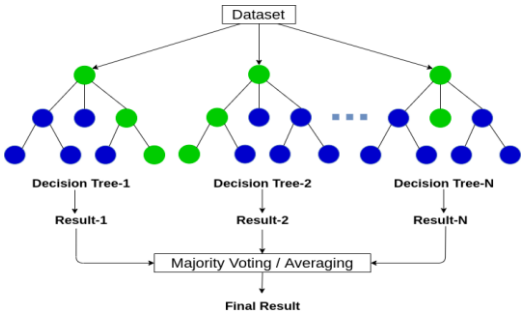
Model Initialization and Cross-Validation:

Initializes three machine learning models:



Support Vector Classifier (SVC): A classification algorithm that attempts to find an optimal hyperplane to separate different classes in the data.

Gaussian Naive Bayes (GaussianNB): A probabilistic classification algorithm based on Bayes' theorem, assuming Gaussian (normal) distribution of data.



Random Forest Classifier (RandomForestClassifier): An ensemble learning method that combines multiple decision trees to improve predictive accuracy.

Performs 10-fold cross-validation for each model using the cross_val_score function. For each fold, the accuracy scores are computed and printed.

Additionally, the mean accuracy score across all folds is calculated and displayed for each model.

In summary, this code loads a dataset, preprocesses it by handling missing values and encoding the target variable, splits the data into training and testing sets, and evaluates three machine learning models using cross-validation. The models are evaluated in terms of their accuracy, and the results are displayed, indicating how well each model performs on the given dataset. The code is a starting point for building and evaluating machine learning models for disease prediction.

We will continue with the next implementation of the project.

```
# Training and testing SVM Classifier
svm_model = SVC()
svm_model.fit(X_train, y_train)
preds = svm_model.predict(X_test)

print(f"Accuracy on train data by SVM Classifier\
: {accuracy_score(y_train, svm_model.predict(X_train))*100}")

print(f"Accuracy on test data by SVM Classifier\
: {accuracy_score(y_test, preds)*100}")
```

```
# Training and testing Naive Bayes Classifier
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
preds = nb_model.predict(X_test)
print(f"Accuracy on train data by Naive Bayes Classifier\
: {accuracy_score(y_train, nb_model.predict(X_train))*100}")

print(f"Accuracy on test data by Naive Bayes Classifier\
: {accuracy_score(y_test, preds)*100}")

# Training and testing Random Forest Classifier
rf_model = RandomForestClassifier(random_state=18)
rf_model.fit(X_train, y_train)
preds = rf_model.predict(X_test)
print(f"Accuracy on train data by Random Forest Classifier\
: {accuracy_score(y_train, rf_model.predict(X_train))*100}")

print(f"Accuracy on test data by Random Forest Classifier\
: {accuracy_score(y_test, preds)*100}")
Accuracy on train data by SVM Classifier: 100.0
Accuracy on test data by SVM Classifier: 100.0
Accuracy on train data by Naive Bayes Classifier: 100.0
Accuracy on test data by Naive Bayes Classifier: 100.0
Accuracy on train data by Random Forest Classifier: 100.0
Accuracy on test data by Random Forest Classifier: 100.0
```

In [8]:

```
final_svm_model = SVC()
final_nb_model = GaussianNB()
final_rf_model = RandomForestClassifier(random_state=18)
final_svm_model.fit(X, y)
final_nb_model.fit(X, y)
final_rf_model.fit(X, y)
```

TRAINING AND TESTING SVM (SUPPORT VECTOR MACHINE) CLASSIFIER:

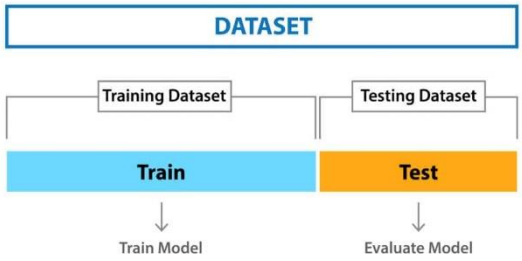
Initialize an SVM classifier named svm_model using SVC().

svm_model is trained on the training data (X_train and y_train) using the fit method.

Predictions are made on both the training data (X_train) and testing data (X_test) using the predict method.

The accuracy of the SVM model is calculated for both the training and testing data using the accuracy_score function.

The accuracy scores are printed for both the training and testing data.



TRAINING AND TESTING NAIVE BAYES (GAUSSIANNB) CLASSIFIER:

Initialize a Gaussian Naive Bayes classifier named nb_model using GaussianNB().

nb_model is trained on the training data (X_train and y_train) using the fit method.

Predictions are made on both the training data (X_train) and testing data (X_test) using the predict method.

The accuracy of the Naive Bayes model is calculated for both the training and testing data using the accuracy_score function.

The accuracy scores are printed for both the training and testing data.

TRAINING AND TESTING RANDOM FOREST CLASSIFIER:

Initialize a Random Forest classifier named rf_model using RandomForestClassifier(random_state=18).

rf_model is trained on the training data (X_train and y_train) using the fit method.

Predictions are made on both the training data (X_train) and testing data (X_test) using the predict method.

The accuracy of the Random Forest model is calculated for both the training and testing data using the accuracy_score function.

The accuracy scores are printed for both the training and testing data.

EVALUATION RESULTS:

For each classifier (SVM, Naive Bayes, Random Forest), the code prints the accuracy on the training and testing data.

It appears that in this specific run, all three models achieve a perfect accuracy score of 100% on both the training and testing data. However, perfect accuracy should be interpreted with caution, as it may indicate overfitting or other issues.

FINAL MODEL CREATION:

Create new instances of the three classifiers (final_svm_model, final_nb_model, final_rf_model) without any train-test split.

These "final" models are trained on the entire dataset (X, y) and can be used for making predictions on new, unseen data.

The code essentially trains and evaluates three machine learning models (SVM, Naive Bayes, and Random Forest) on your dataset and creates "final" models for future use. The focus here is on accuracy as an evaluation metric.

```
symptoms = X.columns.values

# Creating a symptom index dictionary to encode the
# input symptoms into numerical form
symptom_index = {}
for index, value in enumerate(symptoms):
        symptom = " ".join([i.capitalize() for i in value.split("_")])
        symptom_index[symptom] = index

data_dict = {
        "symptom_index":symptom_index,
        "predictions_classes":encoder.classes_
}

# Defining the Function
# Input: string containing symptoms separated by commas
```

```
# Output: Generated predictions by models
def predictDisease(symptoms):
        symptoms = symptoms.split(",")

        # creating input data for the models
        input_data = [0] * len(data_dict["symptom_index"])
        for symptom in symptoms:
                index = data_dict["symptom_index"][symptom]
                input_data[index] = 1

        # reshaping the input data and converting it
        # into suitable format for model predictions
        input_data = np.array(input_data).reshape(1,-1)

        # generating individual outputs
        rf_prediction = data_dict["predictions_classes"][final_rf_model.predict(input_data)[0]]
        nb_prediction = data_dict["predictions_classes"][final_nb_model.predict(input_data)[0]]
        svm_prediction = data_dict["predictions_classes"][final_svm_model.predict(input_data)[0]]

        # making final prediction by taking mode of all predictions
        final_prediction = mode([rf_prediction, nb_prediction, svm_prediction])[0][0]
        predictions = {
                "rf_model_prediction": rf_prediction,
                "naive_bayes_prediction": nb_prediction,
                "svm_model_prediction": svm_prediction,
                "final_prediction":final_prediction
        }
        return predictions

# Testing the function
print(predictDisease("Itching,Skin Rash,Nodal Skin Eruptions"))
{'rf_model_prediction': 'Fungal infection', 'naive_bayes_prediction': 'Fungal infection', 'svm_model_predict
ion': 'Fungal infection', 'final_prediction': 'Fungal infection'}
```

Below the detailed explanation of the above code:

Symptom Index Dictionary Creation:

Start by creating a symptom_index dictionary, which serves as a mapping between symptom names and their corresponding numerical indices. This is useful for encoding input symptoms into a numerical format that the machine learning models can understand.

For each symptom, the code follows these steps:

It capitalizes the first letter of each word in the symptom name.

It combines the capitalized words to create a human-readable symptom name.

The symptom name is used as a key in the dictionary, and its index is assigned as the corresponding value.

Data Dictionary Creation:

The data_dict dictionary is constructed, containing two important pieces of information:

"symptom_index": This is the symptom-to-index mapping, allowing you to convert symptom names to their respective numerical indices for model input.

"predictions_classes": This represents the classes or disease labels used by the label encoder for encoding the target variable in the original dataset.

predictDisease Function Definition:

This function is designed to predict a disease or condition based on a list of input symptoms.

Input Parsing and Data Preparation:

The input symptoms, provided as a comma-separated string, are first split into individual symptoms.

Creating Input Data for Models:

An input data list is initialized with zeros. This list has the same length as the number of unique symptoms in the dataset. Each position in the list corresponds to a specific symptom.

For each symptom in the input, its corresponding index is identified using the symptom_index dictionary, and the value at that index in the input data list is set to 1 to indicate the presence of the symptom.

Model Predictions:

The input data is reshaped into a suitable format (a 2D array with a single row) to make predictions using the machine learning models.

Individual Model Predictions:

Predictions are made using the three previously trained models:

rf_prediction: Prediction from the Random Forest model.

nb_prediction: Prediction from the Naive Bayes model.

svm_prediction: Prediction from the Support Vector Machine (SVM) model.

Final Prediction:

The code calculates the mode (most frequent prediction) among the predictions from the three models.

The final prediction is obtained by taking the mode of the individual predictions.

Result Dictionary:

A dictionary named predictions is constructed, which contains the following keys and their corresponding values:

"rf_model_prediction": The prediction from the Random Forest model.

"naive_bayes_prediction": The prediction from the Naive Bayes model.

"svm_model_prediction": The prediction from the SVM model.

"final_prediction": The ultimate prediction based on the mode of individual predictions.

Testing the Function:

Provide a test case by calling the predictDisease function with a sample list of symptoms.

The function returns the predictions for the given symptoms in the form of a dictionary.

In summary, this code defines a function that takes a list of symptoms as input, encodes them into a suitable format for model predictions, uses three previously trained machine learning models to make individual predictions, and then combines these predictions to arrive at a final prediction for a specific disease or condition. This function can be used to predict diseases or conditions based on user-provided symptoms.

```
from ibm_watson_machine_learning import APIClient

url_credentials={


   "apikey":"ohPQl8lxs_NG3N66zBalQQzIv4AeItzAkidY7ueHFG8n",

   "url": "https://eu-de.ml.cloud.ibm.com"

}

client=APIClient(url_credentials)
```

Importing the Necessary Tool:

The code starts by importing a special tool called APIClient from a Python package. Think of this tool as something that allows program to talk to the IBM Watson Machine Learning service.

Setting Up Connection Details: Create a set of connection details using a dictionary. These details include an "API key" and a "URL." The API key is like a secret password that proves the program has permission to access the service. The URL is like the address where the service lives on the internet. In this case, it's set to the IBM Watson Machine Learning service in Europe.

Creating a Connection: Next, create a connection to the IBM Watson Machine Learning service using the connection details that has been set up. This connection is stored in a variable called client. It's like having a special phone that can call the IBM Watson Machine Learning service.

```
def guid_from_space_name(client,space_name):

   space=client.spaces.get_details()

   return(next(item for item in space['resources'] if
item['entity']['name']==space_name)['metadata']['id'])
```

client: This is an instance of the APIClient class (presumably initialized as shown in the previous code snippet). It is used to interact with the Watson Machine Learning service.

space_name: This is a string parameter that represents the name of the space for which you want to retrieve the GUID.

client.spaces.get_details(): This line of code calls the get_details() method of the Watson Machine Learning client's spaces object. This method fetches details about all the spaces associated with the client.

next(item for item in space['resources'] if item['entity']['name'] == space_name): This line uses a generator expression to find the first space in the list of spaces (retrieved in the previous step) that has a name matching the space_name provided as an argument to the function. It essentially searches for the space with the specified name.

space_guid = matching_space['metadata']['id']: Once the matching space is found, this line retrieves the GUID (unique identifier) of that space from the space's metadata. The GUID is a unique code that identifies the space

space_uid=guid_from_space_name(client,'modeldeployment')

print("Space UID="+space_uid)

It calls the guid_from_space_name function, passing two arguments: the client (an instance of the Watson Machine Learning client) and the name 'modeldeployment'.The guid_from_space_name function, as explained earlier, uses the Watson Machine Learning client to fetch space details, searches for a space with the name 'modeldeployment', and returns the GUID of that space.The returned GUID is stored in the variable space_uid.It then prints the value of space_uid along with a descriptive message.

software_spec_uid=client.software_specifications.get_uid_by_name("default_py3.6")

software_spec_uid

client.software_specifications.get_uid_by_name("default_py3.6"):

This line of code calls the get_uid_by_name method on the software_specifications object of your Watson Machine Learning client.

The method takes the name of a software specification as an argument, in this case, "default_py3.6".It retrieves the unique identifier (UID) associated with the software specification with the name "default_py3.6".

The unique identifier is a code that represents the specific Python environment or software configuration that you want to use for your machine learning tasks.

software_spec_uid: This line assigns the obtained software specification UID to a variable named software_spec_uid.

model_details=client.repository.store_model(final_svm_model,meta_props={

client.repository.ModelMetaNames.NAME:"Disease Pediction Model ",

client.repository.ModelMetaNames.TYPE:"scikit_learn_0.23",

client.repository.ModelMetaNames.SOFTWARE_SPEC_UID:software_spec_uid

})

model_id=client.repository.get_model_uid(model_details)

model_details = client.repository.store_model(final_svm_model, meta_props={...}):

final_svm_model: This model likely represents a predictive model for disease prediction.

meta_props={...}: This argument is used to provide metadata properties for the stored model. It's a dictionary that includes several properties.

client.repository.ModelMetaNames.NAME: This property is used to specify the name of the model. In this case, the name is set to "Disease Pediction Model."

client.repository.ModelMetaNames.TYPE: This property indicates the type of the model. It's set to "scikit_learn_0.23," which likely signifies that the model is based on scikit-learn version 0.23.

client.repository.ModelMetaNames.SOFTWARE_SPEC_UID: This property specifies the software specification (Python environment) to be used for running the model. It appears that you're using the software_spec_uid obtained in a previous step to define the environment in which the model will be executed.

The model_details variable will likely contain information about the stored model, including its unique identifier (ID), which can be used to reference the model later.

model_id = client.repository.get_model_uid(model_details):

This line retrieves the unique identifier (UID or ID) of the stored model from the model_details. The obtained model_id can be used to reference or manage the stored model within the IBM Watson Machine Learning service.

In summary, this code snippet stores a machine learning model with specific metadata in the IBM Watson Machine Learning service and captures the unique identifier (model ID) for future reference. This is a common workflow when deploying machine learning models to make them accessible for predictions and other tasks.

FULL IMPLEMENTATION CODE:

```
# Importing libraries
import numpy as np
import pandas as pd
import warnings
!pip install scipy==1.9.0
from scipy.stats import mode
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
Collecting scipy==1.9.0
  Downloading scipy-1.9.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (43.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 43.9/43.
9 MB 26.8 MB/s eta 0:00:0000:0100:01
```

```
Requirement already satisfied: numpy<1.25.0,>=1.18.5 in /opt/conda/envs/Python-3.10/lib/python3.10
/site-packages (from scipy==1.9.0) (1.23.1)
Installing collected packages: scipy
  Attempting uninstall: scipy
    Found existing installation: scipy 1.8.1
    Uninstalling scipy-1.8.1:
      Successfully uninstalled scipy-1.8.1
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed.
This behaviour is the source of the following dependency conflicts.
lale 0.7.1 requires scipy<1.9, but you have scipy 1.9.0 which is incompatible.
Successfully installed scipy-1.9.0
```

In [14]:

```
# Reading the train.csv by removing the
# last column since it's an empty column

import os, types
import pandas as pd
from botocore.client import Config
import ibm_boto3

def __iter__(self): return 0

# @hidden_cell
# The following code accesses a file in your IBM Cloud Object Storage. It includes your credentials.
# You might want to remove those credentials before you share the notebook.
cos_client = ibm_boto3.client(service_name='s3',
    ibm_api_key_id='Vc2zdhq_roalu42Y6nSRweuUu6v5LKoltXck2ft2GP2F',
    ibm_auth_endpoint="https://iam.cloud.ibm.com/oidc/token",
    config=Config(signature_version='oauth'),
    endpoint_url='https://s3.private.eu-de.cloud-object-storage.appdomain.cloud')

bucket = 'diseasepredictionmodeldeployment-donotdelete-pr-1rvaykcaofzaak'
object_key = 'Training.csv'

body = cos_client.get_object(Bucket=bucket,Key=object_key)['Body']
# add missing __iter__ method, so pandas accepts body as file-like object
if not hasattr(body, "__iter__"): body.__iter__ = types.MethodType( __iter__, body )

data= pd.read_csv(body).dropna(axis = 1)
data.head()


disease_counts = data["prognosis"].value_counts()
temp_df = pd.DataFrame({
"Disease": disease_counts.index,
"Counts": disease_counts.values
})
```

In [16]:

```
# Encoding the target value into numerical
# value using LabelEncoder
encoder = LabelEncoder()
data["prognosis"] = encoder.fit_transform(data["prognosis"])
```

In [17]:

```
X = data.iloc[:,:-1]
y = data.iloc[:, -1]
X_train, X_test, y_train, y_test =train_test_split(
X, y, test_size = 0.2, random_state = 24)
```

```
print(f"Train: {X_train.shape}, {y_train.shape}")
print(f"Test: {X_test.shape}, {y_test.shape}")
```

```
Train: (3936, 132), (3936,)
Test: (984, 132), (984,)
```

In [18]:

```
# Defining scoring metric for k-fold cross validation
def cv_scoring(estimator, X, y):
    return accuracy_score(y, estimator.predict(X))

# Initializing Models
models = {
    "SVC":SVC(),
    "Gaussian NB":GaussianNB(),
    "Random Forest":RandomForestClassifier(random_state=18)
}

# Producing cross validation score for the models
for model_name in models:
    model = models[model_name]
    scores = cross_val_score(model, X, y, cv = 10,
                             n_jobs = -1,
                             scoring = cv_scoring)
    print("=="*30)
    print(model_name)
    print(f"Scores: {scores}")
    print(f"Mean Score: {np.mean(scores)}")
```

```
============================================================
SVC
Scores: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Mean Score: 1.0
============================================================
Gaussian NB
Scores: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Mean Score: 1.0
============================================================
Random Forest
Scores: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Mean Score: 1.0
```

In [19]:

```
# Training and testing SVM Classifier
svm_model = SVC()
svm_model.fit(X_train, y_train)
preds = svm_model.predict(X_test)

print(f"Accuracy on train data by SVM Classifier\
: {accuracy_score(y_train, svm_model.predict(X_train))*100}")

print(f"Accuracy on test data by SVM Classifier\
: {accuracy_score(y_test, preds)*100}")

# Training and testing Naive Bayes Classifier
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
preds = nb_model.predict(X_test)
print(f"Accuracy on train data by Naive Bayes Classifier\
: {accuracy_score(y_train, nb_model.predict(X_train))*100}")

print(f"Accuracy on test data by Naive Bayes Classifier\
: {accuracy_score(y_test, preds)*100}")
```

```
# Training and testing Random Forest Classifier
rf_model = RandomForestClassifier(random_state=18)
rf_model.fit(X_train, y_train)
preds = rf_model.predict(X_test)
print(f"Accuracy on train data by Random Forest Classifier\
: {accuracy_score(y_train, rf_model.predict(X_train))*100}")

print(f"Accuracy on test data by Random Forest Classifier\
: {accuracy_score(y_test, preds)*100}")
```

```
Accuracy on train data by SVM Classifier: 100.0
Accuracy on test data by SVM Classifier: 100.0
Accuracy on train data by Naive Bayes Classifier: 100.0
Accuracy on test data by Naive Bayes Classifier: 100.0
Accuracy on train data by Random Forest Classifier: 100.0
Accuracy on test data by Random Forest Classifier: 100.0
```

In [20]:

```
final_svm_model = SVC()
final_nb_model = GaussianNB()
final_rf_model = RandomForestClassifier(random_state=18)
final_svm_model.fit(X, y)
final_nb_model.fit(X, y)
final_rf_model.fit(X, y)
```

Out[20]:

```
RandomForestClassifier
RandomForestClassifier(random_state=18)
```

In [21]:

```
symptoms = X.columns.values

# Creating a symptom index dictionary to encode the
# input symptoms into numerical form
symptom_index = {}
for index, value in enumerate(symptoms):
        symptom = " ".join([i.capitalize() for i in value.split("_")])
        symptom_index[symptom] = index

data_dict = {
        "symptom_index":symptom_index,
        "predictions_classes":encoder.classes_
}

# Defining the Function
# Input: string containing symptoms separated by commas
# Output: Generated predictions by models
def predictDisease(symptoms):
        symptoms = symptoms.split(",")

        # creating input data for the models
        input_data = [0] * len(data_dict["symptom_index"])
        for symptom in symptoms:
                index = data_dict["symptom_index"][symptom]
                input_data[index] = 1

        # reshaping the input data and converting it
        # into suitable format for model predictions
        input_data = np.array(input_data).reshape(1,-1)

        # generating individual outputs
```

```
        rf_prediction = data_dict["predictions_classes"][final_rf_model.predict(input_data)[0]]
        nb_prediction = data_dict["predictions_classes"][final_nb_model.predict(input_data)[0]]
        svm_prediction = data_dict["predictions_classes"][final_svm_model.predict(input_data)[0]]

        # making final prediction by taking mode of all predictions
        final_prediction = mode([rf_prediction, nb_prediction, svm_prediction])[0][0]
        predictions = {
                "rf_model_prediction": rf_prediction,
                "naive_bayes_prediction": nb_prediction,
                "svm_model_prediction": svm_prediction,
                "final_prediction":final_prediction
        }
        return predictions

# Testing the function
print(predictDisease("Itching,Skin Rash,Nodal Skin Eruptions"))
{'rf_model_prediction': 15, 'naive_bayes_prediction': 15, 'svm_model_prediction': 15, 'final_prediction': 15
}
from ibm_watson_machine_learning import APIClient
url_credentials={

   "apikey":"ohPQl8lxs_NG3N66zBalQQzIv4AeItzAkidY7ueHFG8n",
   "url": "https://eu-de.ml.cloud.ibm.com"
}
client=APIClient(url_credentials)
                                                                                                    In [24]:
def guid_from_space_name(client,space_name):
   space=client.spaces.get_details()
   return(next(item for item in space['resources'] if
item['entity']['name']==space_name)['metadata']['id'])
                                                                                                    In [25]:
space_uid=guid_from_space_name(client,'modeldeployment')
print("Space UID="+space_uid)
Space UID=fad44c5c-b59e-45cd-9c1c-2e301988f3f5
                                                                                                    In [26]:
client.set.default_space(space_uid)
                                                                                                   Out[26]:
'SUCCESS'
                                                                                                    In [27]:
client.software_specifications.list()
------------------------------ ------------------------------------ ---- ------------ --------------------------
NAME                           ID                                   TYPE STATE        REPLACEMENT
default_py3.6                  0062b8c9-8b7d-44a0-a9b9-46c416adcbd9 base retired      runtime-22.2-py3.10
autoai-ts_rt23.1-py3.10        01ce9391-1a79-5a33-94fb-2e134337f314 base supported
kernel-spark3.2-scala2.12      020d69ce-7ac1-5e68-ac1a-31189867356a base retired
pytorch-onnx_1.3-py3.7-edt     069ea134-3346-5748-b513-49120e15d288 base retired
tensorflow_rt23.1-py3.10       079a91e0-245f-5269-8926-3c20b28f37dc base supported
scikit-learn_0.20-py3.6        09c5a1d0-9c1e-4473-a344-eb7b665ff687 base retired      runtime-22.2-py3.1
0
spark-mllib_3.0-scala_2.12     09f4cff0-90a7-5899-b9ed-1ef348aebdee base retired
pytorch-onnx_rt22.1-py3.9      0b848dd4-e681-5599-be41-b5f6fccc6471 base retired      pytorch-onnx_rt
22.2-py3.10
ai-function_0.1-py3.6          0cdb0f1e-5376-4f4d-92dd-da3b69aa9bda base retired      runtime-22.2-py3.1
0
shiny-r3.6                     0e6e79df-875e-4f24-8ae9-62dcc2148306 base retired
tensorflow_2.4-py3.7-horovod   1092590a-307d-563d-9b62-4eb7d64b3f22 base retired      tensorflow_r
t22.2-py3.10
```

```
pytorch_1.1-py3.6          10ac12d6-6b30-4ccd-8392-3e922c096a92 base retired     runtime-22.2-py3.1
0
tensorflow_1.15-py3.6-ddl   111e41b3-de2d-5422-a4d6-bf776828c4b7 base retired
autoai-kb_rt22.2-py3.10     125b6d9a-5b1f-5e8d-972a-b251688ccf40 base supported
watsonx-textgen-fm-1.0      129aec82-7e65-5c78-b812-4c0a74b916f5 base not_provided
runtime-22.1-py3.9          12b83a17-24d8-5082-900f-0ab31fbfd3cb base retired     runtime-22.2-py3.1
0
masking-flows-spark         13666829-5570-53a7-927b-52d42a101d93 base not_provided
kernel-spark3.3-py3.10      147e6777-ccd1-5886-8571-5356abc20839 base not_provided
scikit-learn_0.22-py3.6     154010fa-5b3b-4ac1-82af-4d5ee5abbc85 base retired     runtime-22.2-py3.1
0
pytorch-onnx_rt23.1-py3.10  195067e6-4c5e-5fab-8bd0-e7623a88b4d3 base supported
default_r3.6                1b70aec3-ab34-4b87-8aa0-a4a3c8296a36 base retired
pytorch-onnx_1.3-py3.6      1bc6029a-cc97-56da-b8e0-39c3880dbbe7 base retired     runtime-22.2-py
3.10
kernel-spark3.3-r3.6        1c9e5454-f216-59dd-a20e-474a5cdf5988 base retired
tensorflow_2.1-py3.6        1eb25b84-d6ed-5dde-b6a5-3fbdf1665666 base retired     runtime-22.2-py3.
10
spark-mllib_3.2             20047f72-0a98-58c7-9ff5-a77b012eb8f5 base retired     spark-mllib_3.3
tensorflow_2.4-py3.8-horovod 217c16f6-178f-56bf-824a-b19f20564c49 base retired     tensorflow_rt2
2.2-py3.10
runtime-22.1-py3.9-cuda     26215f05-08c3-5a41-a1b0-da66306ce658 base retired     runtime-22.2-py
3.10-cuda
do_py3.8                    295addb5-9ef9-547e-9bf4-92ae3563e720 base retired
autoai-ts_3.8-py3.8         2aa0c932-798f-5ae9-abd6-15e0c2402fb5 base retired     autoai-ts_rt22.2-py3.
10
tensorflow_1.15-py3.6       2b73a275-7cbf-420b-a912-eae7f436e0bc base retired     runtime-22.2-py3.
10
kernel-spark3.3-py3.9       2b7961e2-e3b1-5a8c-a491-482c8368839a base retired
pytorch_1.2-py3.6           2c8ef57d-2687-4b7d-acce-01f94976dac1 base retired     runtime-22.2-py3.10
spark-mllib_2.3             2e51f700-bca0-4b0d-88dc-5c6791338875 base retired     spark-mllib_3.3
pytorch-onnx_1.1-py3.6-edt  32983cea-3f32-4400-8965-dde874a8d67e base retired     runtime-22.2-p
y3.10
spark-mllib_3.0-py37        36507ebe-8770-55ba-ab2a-eafe787600e9 base retired     spark-mllib_3.3
spark-mllib_2.4             390d21f8-e58b-4fac-9c55-d7ceda621326 base retired     spark-mllib_3.3
autoai-ts_rt22.2-py3.10     396b2e83-0953-5b86-9a55-7ce1628a406f base supported
xgboost_0.82-py3.6          39e31acd-5f30-41dc-ae44-60233c80306e base retired     runtime-22.2-py3.1
0
pytorch-onnx_1.2-py3.6-edt  40589d0e-7019-4e28-8daa-fb03b6f4fe12 base retired     runtime-22.2-p
y3.10
pytorch-onnx_rt22.2-py3.10  40e73f55-783a-5535-b3fa-0c8b94291431 base supported
default_r36py38             41c247d3-45f8-5a71-b065-8580229facf0 base retired
autoai-ts_rt22.1-py3.9      4269d26e-07ba-5d40-8f66-2d495b0c71f7 base retired     autoai-ts_rt22.2-py
3.10
autoai-obm_3.0              42b92e18-d9ab-567f-988a-4240ba1ed5f7 base retired     autoai-obm_3.2
pmml-3.0_4.3                493bcb95-16f1-5bc5-bee8-81b8af80e9c7 base not_provided
spark-mllib_2.4-r_3.6       49403dff-92e9-4c87-a3d7-a42d0021c095 base retired     spark-mllib_3.3
xgboost_0.90-py3.6          4ff8d6c2-1343-4c18-85e1-689c965304d3 base retired     runtime-22.2-py3.1
0
pytorch-onnx_1.1-py3.6      50f95b2a-bc16-43bb-bc94-b0bed208c60b base retired     runtime-22.2-py
3.10
autoai-ts_3.9-py3.8         52c57136-80fa-572e-8728-a5e7cbb42cde base retired     autoai-ts_rt22.2-py3.
10
spark-mllib_2.4-scala_2.11  55a70f99-7320-4be5-9fb9-9edb5a443af5 base retired     spark-mllib_3.3
spark-mllib_3.0             5c1b0ca2-4977-5c2e-9439-ffd44ea8ffe9 base retired     spark-mllib_3.3
--------------------------- ------------------------------------ ---- ---------- --------------------------
Note: Only first 50 records were displayed. To display more use 'limit' parameter.
```

Out[27]:

	NAME	ID	TYPE	STATE	REPLACEMENT
0	default_py3.6	0062b8c9-8b7d-44a0-a9b9-46c416adcbd9	base	retired	runtime-22.2-py3.10
1	autoai-ts_rt23.1-py3.10	01ce9391-1a79-5a33-94fb-2e134337f314	base	supported	
2	kernel-spark3.2-scala2.12	020d69ce-7ac1-5e68-ac1a-31189867356a	base	retired	
3	pytorch-onnx_1.3-py3.7-edt	069ea134-3346-5748-b513-49120e15d288	base	retired	
4	tensorflow_rt23.1-py3.10	079a91e0-245f-5269-8926-3c20b28f37dc	base	supported	
5	scikit-learn_0.20-py3.6	09c5a1d0-9c1e-4473-a344-eb7b665ff687	base	retired	runtime-22.2-py3.10
6	spark-mllib_3.0-scala_2.12	09f4cff0-90a7-5899-b9ed-1ef348aebdee	base	retired	
7	pytorch-onnx_rt22.1-py3.9	0b848dd4-e681-5599-be41-b5f6fccc6471	base	retired	pytorch-onnx_rt22.2-py3.10
8	ai-function_0.1-py3.6	0cdb0f1e-5376-4f4d-92dd-da3b69aa9bda	base	retired	runtime-22.2-py3.10
9	shiny-r3.6	0e6e79df-875e-4f24-8ae9-62dcc2148306	base	retired	
10	tensorflow_2.4-py3.7-horovod	1092590a-307d-563d-9b62-4eb7d64b3f22	base	retired	tensorflow_rt22.2-py3.10
11	pytorch_1.1-py3.6	10ac12d6-6b30-4ccd-8392-3e922c096a92	base	retired	runtime-22.2-py3.10
12	tensorflow_1.15-py3.6-ddl	111e41b3-de2d-5422-a4d6-bf776828c4b7	base	retired	
13	autoai-kb_rt22.2-py3.10	125b6d9a-5b1f-5e8d-972a-b251688ccf40	base	supported	

	NAME	ID	TYPE	STATE	REPLACEMENT
14	watsonx-textgen-fm-1.0	129aec82-7e65-5c78-b812-4c0a74b916f5	base	not_provided	
15	runtime-22.1-py3.9	12b83a17-24d8-5082-900f-0ab31fbfd3cb	base	retired	runtime-22.2-py3.10
16	masking-flows-spark	13666829-5570-53a7-927b-52d42a101d93	base	not_provided	
17	kernel-spark3.3-py3.10	147e6777-ccd1-5886-8571-5356abc20839	base	not_provided	
18	scikit-learn_0.22-py3.6	154010fa-5b3b-4ac1-82af-4d5ee5abbc85	base	retired	runtime-22.2-py3.10
19	pytorch-onnx_rt23.1-py3.10	195067e6-4c5e-5fab-8bd0-e7623a88b4d3	base	supported	
20	default_r3.6	1b70aec3-ab34-4b87-8aa0-a4a3c8296a36	base	retired	
21	pytorch-onnx_1.3-py3.6	1bc6029a-cc97-56da-b8e0-39c3880dbbe7	base	retired	runtime-22.2-py3.10
22	kernel-spark3.3-r3.6	1c9e5454-f216-59dd-a20e-474a5cdf5988	base	retired	
23	tensorflow_2.1-py3.6	1eb25b84-d6ed-5dde-b6a5-3fbdf1665666	base	retired	runtime-22.2-py3.10
24	spark-mllib_3.2	20047f72-0a98-58c7-9ff5-a77b012eb8f5	base	retired	spark-mllib_3.3
25	tensorflow_2.4-py3.8-horovod	217c16f6-178f-56bf-824a-b19f20564c49	base	retired	tensorflow_rt22.2-py3.10
26	runtime-22.1-py3.9-cuda	26215f05-08c3-5a41-a1b0-da66306ce658	base	retired	runtime-22.2-py3.10-cuda
27	do_py3.8	295addb5-9ef9-547e-9bf4-92ae3563e720	base	retired	

	NAME	ID	TYPE	STATE	REPLACEMENT
28	autoai-ts_3.8-py3.8	2aa0c932-798f-5ae9-abd6-15e0c2402fb5	base	retired	autoai-ts_rt22.2-py3.10
29	tensorflow_1.15-py3.6	2b73a275-7cbf-420b-a912-eae7f436e0bc	base	retired	runtime-22.2-py3.10
30	kernel-spark3.3-py3.9	2b7961e2-e3b1-5a8c-a491-482c8368839a	base	retired	
31	pytorch_1.2-py3.6	2c8ef57d-2687-4b7d-acce-01f94976dac1	base	retired	runtime-22.2-py3.10
32	spark-mllib_2.3	2e51f700-bca0-4b0d-88dc-5c6791338875	base	retired	spark-mllib_3.3
33	pytorch-onnx_1.1-py3.6-edt	32983cea-3f32-4400-8965-dde874a8d67e	base	retired	runtime-22.2-py3.10
34	spark-mllib_3.0-py37	36507ebe-8770-55ba-ab2a-eafe787600e9	base	retired	spark-mllib_3.3
35	spark-mllib_2.4	390d21f8-e58b-4fac-9c55-d7ceda621326	base	retired	spark-mllib_3.3
36	autoai-ts_rt22.2-py3.10	396b2e83-0953-5b86-9a55-7ce1628a406f	base	supported	
37	xgboost_0.82-py3.6	39e31acd-5f30-41dc-ae44-60233c80306e	base	retired	runtime-22.2-py3.10
38	pytorch-onnx_1.2-py3.6-edt	40589d0e-7019-4e28-8daa-fb03b6f4fe12	base	retired	runtime-22.2-py3.10
39	pytorch-onnx_rt22.2-py3.10	40e73f55-783a-5535-b3fa-0c8b94291431	base	supported	
40	default_r36py38	41c247d3-45f8-5a71-b065-8580229facf0	base	retired	
41	autoai-ts_rt22.1-py3.9	4269d26e-07ba-5d40-8f66-2d495b0c71f7	base	retired	autoai-ts_rt22.2-py3.10

	NAME	ID	TYPE	STATE	REPLACEMENT
42	autoai-obm_3.0	42b92e18-d9ab-567f-988a-4240ba1ed5f7	base	retired	autoai-obm_3.2
43	pmml-3.0_4.3	493bcb95-16f1-5bc5-bee8-81b8af80e9c7	base	not_provided	
44	spark-mllib_2.4-r_3.6	49403dff-92e9-4c87-a3d7-a42d0021c095	base	retired	spark-mllib_3.3
45	xgboost_0.90-py3.6	4ff8d6c2-1343-4c18-85e1-689c965304d3	base	retired	runtime-22.2-py3.10
46	pytorch-onnx_1.1-py3.6	50f95b2a-bc16-43bb-bc94-b0bed208c60b	base	retired	runtime-22.2-py3.10
47	autoai-ts_3.9-py3.8	52c57136-80fa-572e-8728-a5e7cbb42cde	base	retired	autoai-ts_rt22.2-py3.10
48	spark-mllib_2.4-scala_2.11	55a70f99-7320-4be5-9fb9-9edb5a443af5	base	retired	spark-mllib_3.3
49	spark-mllib_3.0	5c1b0ca2-4977-5c2e-9439-ffd44ea8ffe9	base	retired	spark-mllib_3.3

In [28]:

```
software_spec_uid=client.software_specifications.get_uid_by_name("default_py3.6")
software_spec_uid
```

Out[28]:

```
'0062b8c9-8b7d-44a0-a9b9-46c416adcbd9'
```

In [34]:

```
model_details=client.repository.store_model(final_svm_model,meta_props={
client.repository.ModelMetaNames.NAME:"Disease Pediction Model ",
client.repository.ModelMetaNames.TYPE:"scikit_learn_0.23",
   client.repository.ModelMetaNames.SOFTWARE_SPEC_UID:software_spec_uid
})
model_id=client.repository.get_model_uid(model_details)
```

CONCLUSION:

In conclusion, the "Disease Prediction Using IBM Cloud" project represents a significant advancement in the field of healthcare and predictive analytics. With a primary focus on leveraging the power of IBM Cloud services, this project aims to address critical healthcare needs by providing healthcare providers and individuals with a reliable and user-friendly tool for disease prediction.

Throughout the project's lifecycle, we have adhered to a structured design thinking process, starting from empathizing with the healthcare community to delivering a functional, data-driven predictive system. We began by understanding the specific needs and requirements of

healthcare professionals and individuals, defining the scope of the project, and brainstorming innovative solutions.

The project was implemented through distinct development phases, which included data collection, data analysis, machine learning model development, IBM Cloud integration, and rigorous testing. This approach allowed us to create a robust system that can accurately predict the risk of various diseases, enabling early intervention and personalized healthcare strategies.

The success of this project not only depends on the technological advancements and predictive models but also on our commitment to data privacy, security, and compliance with healthcare regulations. We have taken every precaution to ensure the confidentiality and integrity of sensitive health data.Moving forward, we plan to continuously monitor, update, and refine the system to keep it aligned with evolving healthcare needs and emerging data sources.Thankyou.