Project Report File

On

“Predicting Life Expectancy Using Machine Learning”

Submitted by:-

Avinash Chaudhary

avinashchaudhary061@gmail.com

ACKNOWLEDGEMENT

I WOULD LIKE TO EXPRESS MY SPECIAL THANKS OF GRATITUDE TO OUR SMARTBRIDGE TEACHER ***MR PRASHANTH V*** SIR AND ***MR MCHARANREDDY*** SIR WHO GAVE ME THE GOLDEN OPPORTUNITY TO DO THIS WONDERFUL PROJECT ON THE TOPIC "***PREDICTING LIFE EXPECTANCY USING MACHINE LEARNING*** " WHICH ALSO HELPED ME IN DOING A LOT OF RESEARCH AND WE COME TO KNOW ABOUT SO MANY NEW THINGS. I AM REALLY THANKFUL TO THEM.

SECONDLY I WOULD LIKE TO THANKS OUR PARENTS AND FRIENDS WHO HELPED ME A LOT IN FINISHING THIS PROJECT WITHIN THE LIMITED TIME.

I AM MAKING THIS PROJECT NOT FOR MARKS BUT TO INCREASE OUR KNOWLEDGE.

THANKS AGAIN TO ALL WHO HELPED ME.

CONTENTS

INTRODUCTION

Life expectancy plays an important role when decisions about the final phase of life need to be made. Good prognostication for example helps to determine the course of treatment and helps to anticipate the procurement of health care services and facilities, or more broadly: facilitates Advance Care Planning. Advance Care Planning (ACP) is the process during which patients make decisions about the health care they wish to receive in the future, in case the patient loses the capacity of making decisions or communicating about them . Successful ACP enhances the quality of life and death for palliative patients, by providing timely palliative care and documenting preferences regarding resuscitation and euthanasia, among other things .

Accurate prognosis of life expectancy is essential for general practitioners (GPs) to decide when to introduce the topic of ACP to the patient, and it is a key determinant in end-of-life decisions . Increasing the accuracy of prognoses has the potential to benefit patients in various ways by enabling more consistent ACP, earlier and better anticipation on palliative needs, and preventing excessive treatment. This study focuses on automatic life expectancy prediction based on medical records.

1.1. Overview

We define the task to solve as follows: predict the life expectancy (in number of months) of a patient at a certain moment in time, given the patient's medical history up to that moment. In order to learn the task automatically from data, we trained an LSTM model on medical records of deceased patients with a recorded date of death, in which the month of death functions as the target to be predicted. We optimized the model architecture and feature set, and tested the performance of several models. The following sections describe:

1. the dataset;
2. the train-validation-test split;
3. our methods for creating the input data for the model;
4. our methods for determining the model architecture;
5. our methods for feature selection;
6. the evaluation protocol.

The project tries to create a model based on data provided by the World Health Organization (WHO) to evaluate the life expectancy for different countries in years. The data offers a timeframe from 2000 to 2015. The data output algorithms have been used to test if they can maintain their accuracy in predicting the life expectancy for data they haven't been trained.

Life expectancy is a statistical measure of the average time a human being is expected to live, Life expectancy depends on various factors: Regional variations, Economic Circumstances, Sex Differences, Mental Illnesses, Physical Illnesses, Education, Year of their birth and other demographic factors. This problem statement provides a way to predict average life expectancy of people living in a country when various factors such as year, GDP, education, alcohol intake of people in the country, expenditure on healthcare system and some specific disease related deaths that happened in the country are given.

1.2. Purpose

The purpose of this study is to train machine learning models using a dataset containing data from over 800 medical examinations, in order to investigate what information can be learned regarding life expectancy. The models are trained as binary classifiers and four different target features are used. The features used are limited to features understandable to the author, who is not a medical professional. Results point to significant features primarily being smoking and body mass index and secondarily alcohol consumption, physical activity and coffee consumption.

In this project I will use machine learning and AutoAI to determine the key factors driving life expectancy in the countries of the world throughout history.

Using these factors, one can predict the life expectancy of a population using health, social, and economic variables. It would be a stretch to use this information to predict the life expectancy of an individual because there are many more life variables involved than the variables presented in this project. This project and data analysis is most useful at predicting life expectancy at the population level.

2. Literature survey

2.1. Existing Problem

Life expectancy plays an important role when decisions about the final phase of life need to be made. Good prognostication for example helps to determine the course of treatment and helps to anticipate the procurement of health care services and facilities, or more broadly: facilitates Advance Care Planning. Advance Care Planning (ACP) is the process during which patients make decisions about the health care they wish to receive in the future, in case the patient loses the capacity of making decisions or communicating about them. Successful ACP enhances the quality of life and death for palliative patients, by providing timely palliative care and documenting preferences regarding resuscitation and euthanasia, among other things .

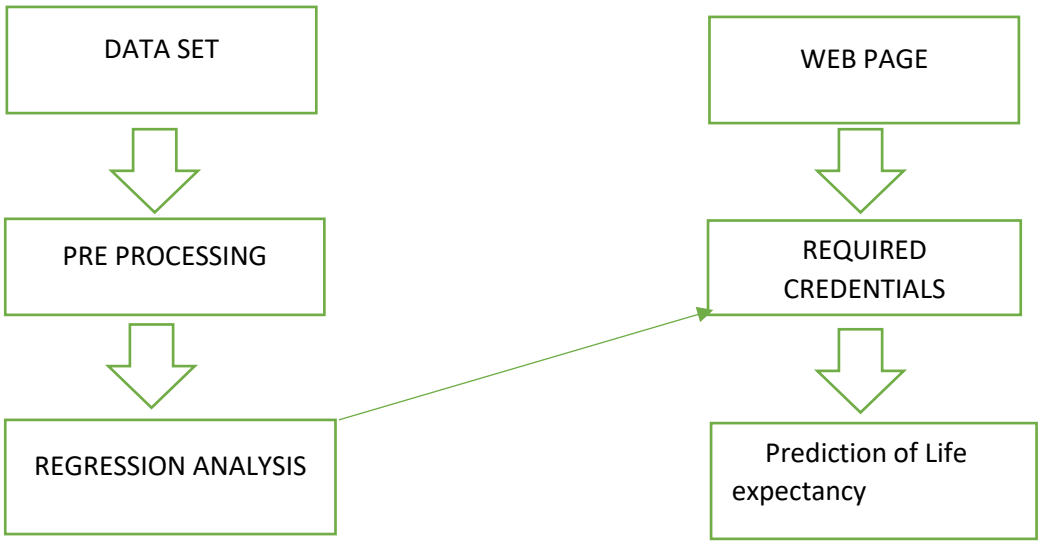
Accurate prognosis of life expectancy is essential for general practitioners (GPs) to decide when to introduce the topic of ACP to the patient, and it is a key determinant in end-of-life decisions. Increasing the accuracy of prognoses has the potential to benefit patients in various ways by enabling more consistent ACP, earlier and better anticipation on palliative needs, and preventing excessive treatment. This study focuses on automatic life expectancy prediction based on medical records.

2.2. Proposed Solution

The project tries to create a model based on data provided by the World Health Organization (WHO) to evaluate the life expectancy for different countries in years. The data offers a timeframe from 2000 to 2015. The data originates from here: https://www.kaggle.com/kumarajarshi/life-expectancy-who/data. The output algorithms have been used to test if they can maintain their accuracy in predicting the life expectancy for data they haven't been trained.

3. Theoretical Analysis

3.1. Block Diagram



3.2. Hardware/Software designing

3.2.1. IBM Cloud Platform

IBM cloud computing is a set of cloud computing services for business offered by the information technology company IBM. IBM Cloud includes infrastructure as a service (IaaS), software as a service (SaaS) and platform as a service (PaaS) offered through public, private and hybrid cloud delivery models, in addition to the components that make up those clouds.

IBM offers three hardware platforms for cloud computing. These platforms offer built-in support for virtualization. For virtualization IBM offers IBM Websphere application infrastructure that supports programming models and open standards for virtualization.

The management layer of the IBM cloud framework includes IBM Tivoli middleware. Management tools provide capabilities to regulate images with automated provisioning and de-provisioning, monitor operations and meter usage while tracking costs and allocating billing. The last layer of the framework provides integrated workload tools. Workloads for cloud computing are services or instances of code that can be executed to meet specific business needs. IBM offers tools for cloud based collaboration, development and test, application development, analytics, business-to-business integration, and security.

3.2.2. IBM Node-Red Service

Browser-based flow editing

Node-RED provides a browser-based flow editor that makes it easy to wire together flows using the wide range of nodes in the palette. Flows can be then deployed to the runtime in a single-click.

JavaScript functions can be created within the editor using a rich text editor. A built-in library allows you to save useful functions, templates or flows for re-use.

Built on Node.js

The light-weight runtime is built on Node.js, taking full advantage of its event-driven, non-blocking model. This makes it ideal to run at the edge of the network on low-cost hardware such as the Raspberry Pi as well as in the cloud.

With over 225,000 modules in Node's package repository, it is easy to extend the range of palette nodes to add new capabilities.

Social Development

The flows created in Node-RED are stored using JSON which can be easily imported and exported for sharing with others.

An online flow library allows you to share your best flows with the world.

3.2.3. IBM Watson Service

Watson was created as a question answering (QA) computing system that IBM built to apply advanced natural language processing, information retrieval, knowledge representation , automated reasoning, and machine learning technologies to the field of open domain question answering.

The key difference between QA technology and document search is that document search takes a keyword query and returns a list of documents, ranked in order of relevance to the query (often based on popularity and page ranking), while QA technology takes a question expressed in natural language, seeks to understand it in much greater detail, and returns a precise answer to the question.

When created, IBM stated that, “more than 100 different techniques are used to analyze natural language, identify sources, find and generate hypotheses, find and score evidence, and merge and rank hypotheses.

In recent years, the Watson capabilities have been extended and the way in which Watson works has been changed to take advantage of new deployment models (Watson on IBM Cloud) and evolved machine learning capabilities and optimised hardware available to developers and researchers. It is no longer purely a question answering (QA) computing system designed from Q&A pairs but can now 'see', 'hear', 'read', 'talk', 'taste', 'interpret', 'learn' and 'recommend'.

- **Software**

  Watson uses IBM's software and the Apache UIMA (Unstructured Information Management Architecture) framework implementation. The system was written in

various languages, including Java, C++, and Prolog, and runs on the SUSE Linux Enterprise Server 11.

- **Hardware**

  The system is workload-optimized, integrating massively parallel POWER7 processors and built on IBM's DeepQA technology, which it uses to generate hypotheses, gather massive evidence, and analyze data. Watson employs a cluster of ninety IBM Power 750 servers, each of which uses a 3.5GHz POWER7 eight-core processor, with four threads per core. In total, the system has 2,880 POWER7 processor threads and 16 terab ytes of RAM.

- **Data**

  The sources of information for Watson include encyclopedias, dictionaries, thesauri, newswire articles and literally works. Watson also used databases, taxonomies and ontologies including DBPedia, WordNet and Yago. The IBM team provided Watson with millions of documents, including dictionaries, encyclopedias and other reference material that it could use to build its knowledge.

3.2.4. IBM Watson Studio

IBM Watson Studio helps data scientists and analysts prepare data and build models at scale across any cloud. With its open, flexible multicloud architecture, Watson Studio provides capabilities that empower businesses to simplify enterprise data science and AI:

- Automate AI lifecycle management with AutoAI.
- Visually prepare and build models with IBM SPSS Modeler.
- Build models using images with IBM Watson Visual Recognition and texts with IBM Watson Natural Language Classifier.
- Deploy and run models through one-click integration with IBM Watson Machine Learning.
- Manage and monitor models through integration with IBM Watson Open Scale.

Hardware:

- A laptop with at least 4GB RAM
- A 2GB GPU

Software:

- **IDE**- Spyder, Jupyter
- **Scientific Computation Library** – Pandas
- **Visualization Libraries** – Matplotlib , Seaborn
- **Algorithmic Libraries** – Scikit-Learn , Stats models
- **Dependencies** – Data from internet

4. Experimental Investigations

Increasing age is a risk factor for many diseases; therefore developing pharmacological interventions that slow down ageing and consequently postpone the onset of many age-related diseases is highly desirable. In this work we analyse data from the Drug Age database, which contains chemical compounds and their effect on the lifespan of model organisms. Predictive models were built using the machine learning method random forests to predict whether or not a chemical compound will increase Caenorhabditis elegans' lifespan, using as features Gene Ontology (GO) terms annotated for proteins targeted by the compounds and chemical descriptors calculated from each compound's chemical structure. The model with the best predictive accuracy used both biological and chemical features, achieving a prediction accuracy of 80%. The top 20 most important GO terms include those related to mitochondrial processes, to enzymatic and immunological processes, and terms related to metabolic and transport processes. We applied our best model to predict compounds which are more likely to increase C. elegans' lifespan in the DGIdb database, where the effect of the compounds on an organism's lifespan is unknown. The top hit compounds can be broadly divided into four groups: compounds affecting mitochondria, compounds for cancer treatment, anti-inflammatories, and compounds for gonadotropinreleasing hormone therapies.

4.1. Prediction using AutoAI/ Without Python

In this project we do prediction by using AutoAI it is the app in IBM Cloud service which is used to define prediction by many algorithm within a step without writing a code in it.

Two types of algorithm applied in it :-

- Extra Trees Regression
- Decision Tree Regression

AutoAI have the ability to choose best algorithm for data by visualization.

It can chose, Linear Regression and Logistic Regression etc.

In this we used Node-Red application also, it is used for the make Flowgraph and calculate the prediction easily if your Flowgraph is correct.

- **AutoAI**

  Automated Artificial Intelligence (AutoAI) is a variation of the automated machine learning, or AutoML, technology, which extends the automation of model building towards automation of the full life cycle of a machine learning model. It applies intelligent automation to the task of building predictive machine learning models by preparing data for training, identifying the best type of model for the given data, then choosing the features, or columns of data, that best support the problem the model is solving. Finally, automation tests a variety of tuning options to reach the best result as it generates, then ranks, model-candidate pipelines. The best performing pipelines can be put into production to process new data, and deliver predictions based on the model training. Automated artificial intelligence can also be applied to making sure the model does not have inherent bias and automating the tasks for continuous improvement of the model.

The Automated Machine Learning and Data Science Team (AMLDS),[2] a small team within IBM Research, which was formed to "apply techniques from Artificial Intelligence (AI), Machine Learning (ML), and data management to accelerate and optimize the creation of machine learning and data science workflows," is credited with advancing the development of AutoAI.

Usecase of AutoAI:-

A typical use case for AutoAI would be training a model to predict how customers might respond to a sales incentive. The model is first trained with actual data on how customers responded to the promotion. Presented with new data, the model can provide a prediction of how a new customer might respond, with a confidence score for the prediction. Prior to AutoML, data scientists had to build these predictive models by hand, testing various combinations of algorithms, then testing to see how predictions compared to actual results. Where AutoML automated some of the process of preparing the data for training, applying algorithms to process the data and then further optimizing the results, AutoAI provides greater intelligent automation that allows for testing significantly more combinations of factors to generate model candidate pipelines that more accurately reflect and address the problem being solved. Once built, a model can be tested for bias and updated to improve performance.

The AutoAI have do their prediction in 5 steps:-

1. Provide data in .csv file.
2. Prepare data.
- Featured type detection.
- Missing values imputation.
- Feature encoding and scaling.
3. Select model type.
   Selection of best algorithm for the data.
4. Generate and Rank model pipelines.
- Hyper-Parameter optimization(HPO).
- Optimized feature engineering.
5. Save and Deploy a model.

- **Node-Red**
  Node-Red is used to make Flowgraph.

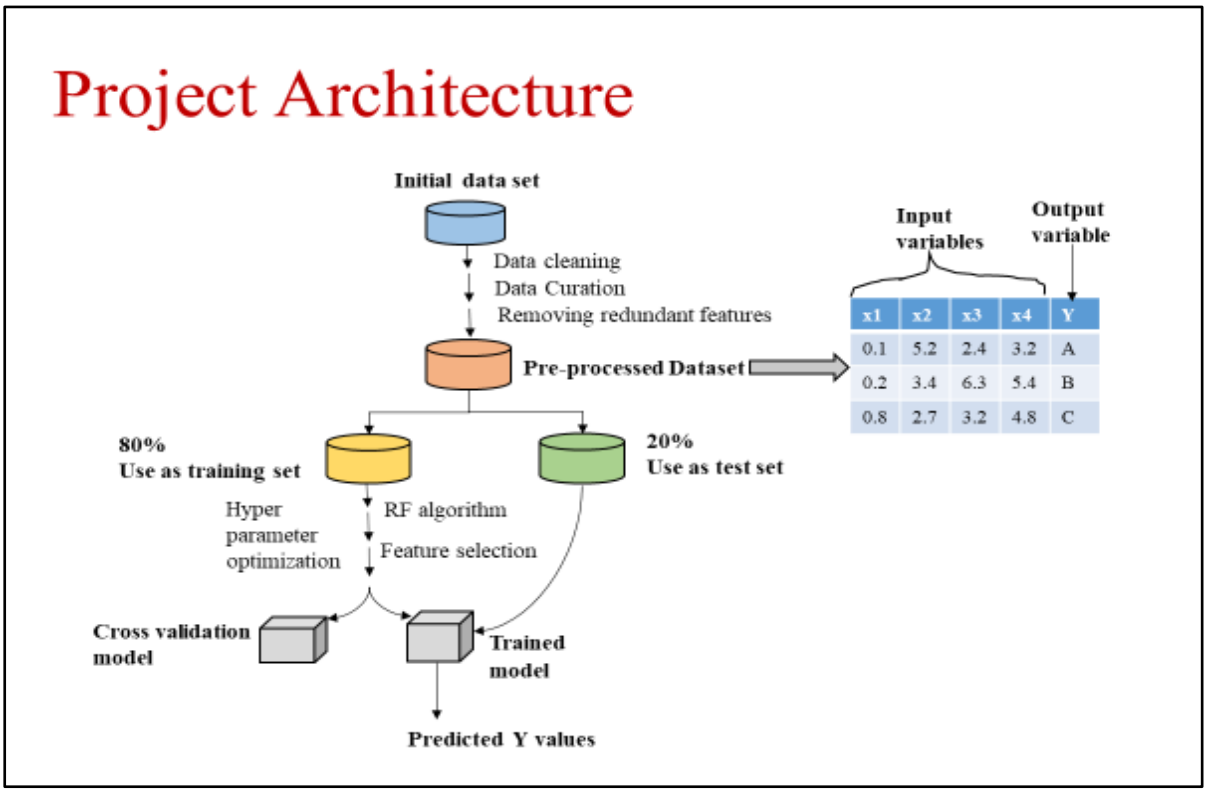
  Node-RED is a flow-based development tool for visual programming developed originally by IBM for wiring together hardware devices, APIs and online services as part of the Internet of Things.

  Node-RED provides a web browser-based flow editor, which can be used to create JavaScript functions. Elements of applications can be saved or shared for re-use.

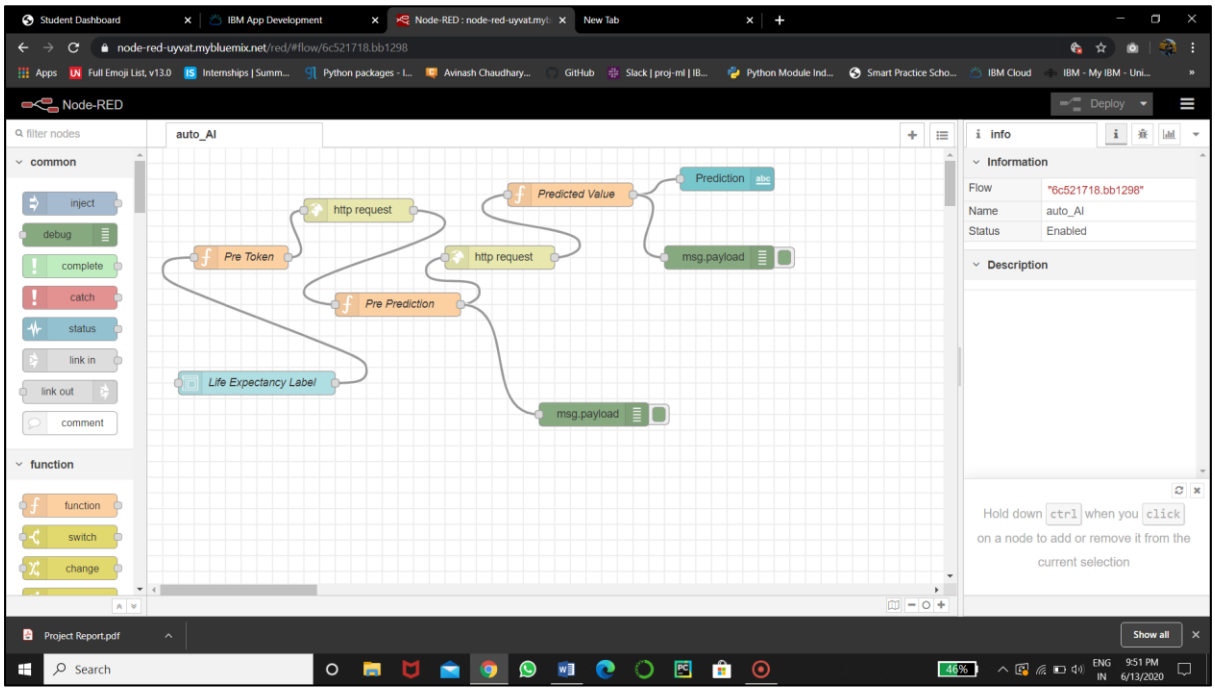
The runtime is built on Node.js. The flows created in Node-RED are stored using JSON. MQTT nodes can make properly configures TLS connections.

Node-Red is a Visual tool for wiring the internet of Things.

5. Flowchart

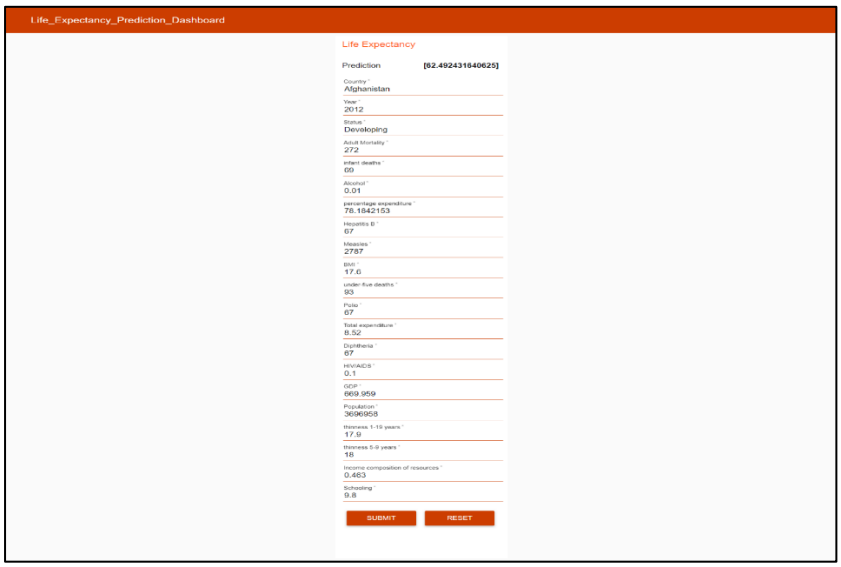


➢ Node-Red-Flow



6. Result

The user should enter the credentials required for prediction .The avg life expectancy prediction will be displayed on the screen.



7. Advantages & Disadvantages

Advantages of Machine learning

1. Easily identifies trends and patterns

Machine Learning can review large volumes of data and discover specific trends and patterns that would not be apparent to humans. For instance, for an e-commerce website like Amazon, it serves to understand the browsing behaviours and purchase histories of its users to help cater to the right products, deals, and reminders relevant to them. It uses the results to reveal relevant advertisements to them.

2. No human intervention needed (automation)

With ML, you don't need to babysit your project every step of the way. Since it means giving machines the ability to learn, it lets them make predictions and also improve the algorithms on their own. A common example of this is anti-virus softwires; they learn to filter new threats as they are recognized. ML is also good at recognizing spam.

3. Continuous Improvement

As ml models gain experience, they keep improving in accuracy and efficiency. This lets them make better decisions. Say you need to make a weather forecast model. As the amount of data you have keeps growing, your algorithms learn to make more accurate predictions faster.

4. Handling multi-dimensional and multi-variety data

Machine Learning algorithms are good at handling data that are multi-dimensional and multi-variety, and they can do this in dynamic or uncertain environments.

5. Wide Applications

You could be an e-tailer or a healthcare provider and make ML work for you. Where it does apply, it holds the capability to help deliver a much more personal experience to customers while also targeting the right customers.

Disadvantages of Machine Learning

With all those advantages to its powerfulness and popularity, Machine Learning isn't perfect. The following factors serve to limit it:

1. Data Acquisition

Machine Learning requires massive data sets to train on, and these should be inclusive/unbiased, and of good quality. There can also be times where they must wait for new data to be generated.

2. Time and Resources

ML needs enough time to let the algorithms learn and develop enough to fulfill their purpose with a considerable amount of accuracy and relevancy. It also needs massive resources to function. This can mean additional requirements of computer power for you.

3. Interpretation of Results

Another major challenge is the ability to accurately interpret results generated by the algorithms. You must also carefully choose the algorithms for your purpose.

4. High error-susceptibility

ML is autonomous but highly susceptible to errors. Suppose you train an algorithm with data sets small enough to not be inclusive. You end up with biased predictions coming from a biased training set. This leads to irrelevant advertisements being displayed to customers. In the case of ML, such blunders can set off a chain of errors that can go undetected for long periods of time. And when they do get noticed, it takes quite some time to recognize the source of the issue, and even longer to correct it.

8. Applications

Existing health mobile app can integrate PLE feature as additional services (b) as both functions require the same physiological data to transfer to a monitoring center for PLE calculation. Applications Along with existing heath applications such as fitness tracking, chronic disease monitoring and real-time patient monitoring, the PLE application can be useful for users to improve their lifestyle and exercise by planning goals on a short and long-term basis. For example, the current PLE outcome of 85 years will be adjusted when the user changes

their attributes such as smoking cessation, reducing alcohol consumption, commencing regular exercise, or modifying dietary plans.

9. Conclusion

Prognostication of life expectancy is difficult for humans. Our research shows that machine learning and natural language processing techniques offer a feasible and promising approach to predicting life expectancy. The research has potential for real-life applications, such as supporting timely recognition of the right moment to start Advance Care Planning. After, the code we got the life ecpectancy prediction nearly accuracy with some error.

10. Future Scope

Few works have been done to provide an individually customized life expectancy prediction. We have reviewed existing works and techniques in the prediction of human LE, and reached a conclusion that it is feasible to predict a PLE for individuals using evolving technologies and devices such as big data, AI, machine learning techniques, and PHDs, wearables and mobile health monitoring devices. We also identified that the collection of data will be a huge challenge due to the privacy and government policy considerations, which will require collaboration of various bodies in the health industry. The interworking of a heterogeneous health network is also a challenge for data collection. Despite these challenges, we showed a possibility of a PLE prediction by proposing an approach of data collection and application by smartphone, with which users can enter their information to access the cloud server to obtain their own PLE. No attempt has been made to create this novel idea of using smartphone integrating cloud servers for real-time data entry. We investigated obstacles and barriers that can be resolved by future works described below. Previous works have described a five year LE prediction, however it is not oriented as a personalized prediction but rather utilizes a median model-predicted probability of 5-year survival of patients who are either sick or healthy. It is proposed that this can be extended to a lifetime prediction by using big data to generate a generic data, which can be used to create a PLE based on training data as a future solution. Building a generic database will take a considerable amount of time for data collection and analysis, taken from birth to death for various demographic groups to be useful and accurate in representing each attribute classifications. Whilst current applications attempt to show PLE for smartphone users, they are complicated and difficult to collect technical data requested by the questionnaire, as users are unlikely to be able to provide these data themselves. This can be resolved by connecting the app to the central cloud server with the Health networks which provide other health related applications. A centralized cloud server plays a key role in collecting, processing, and creating meaningful value using big data, which forms the input of the solution as well as creates generic data against each user's PLE requirements. Service providers shall envisage challenges and hurdles to obtain consent of personal health 'of heterogeneous health networks across developed countries. This will lead to classification of data based on processing big data and each group's traits, which can be used as personalized threshold ranges; When this has been completed in a cloud, it can be connected to a smart device app that can provide questionnaires developed by health specialists and collect answers

to customize the user's PLE; Optimization of the generic groups' data is done by developing an algorithm using machine learning for continuously building and optimizing the user's generic data. As the proposed solution requires processing and transmitting health information of users, information security is a key aspect to consider such as privacy as well as ethical requirements recommenced by regulation bodies, such as the Australian national health and medical research . The scope of security and ethical requirements need to be clearly defined and specified for future work as challenges are expected to build a centralized database with incorporation of health networks. For example, North America, Asia, and Europe may have their own unique requirements to satisfy in dealing with health data with different health research guidelines.

Bibilography

1. https://www.kaggle.com/kumarajarshi/life-expectancy-who
2. https://docs.anaconda.com/anaconda/user-guide/tasks/integration/spyder/
3. https://developer.ibm.com/tutorials/how-to-create-a-node-red-starter-application/
4. https://developer.ibm.com/technologies/machine-learning/series/learning-path-machine-learning-for-developers/
5. https://bookdown.org/caoying4work/watsonstudio-workshop/jn.html
6. https://developer.ibm.com/tutorials/watson-studio-auto-ai/
7. https://bookdown.org/caoying4work/watsonstudio-workshop/auto.html#add-asset-as-auto-ai

APPENDIX:-

A. Sample Code

1. Set Up

```
try:
   import autoai_libs
except Exception as e:
   import subprocess
   out = subprocess.check_output('pip install autoai-libs'.split(' '))
   for line in out.splitlines():
      print(line)
   import autoai_libs
import sklearn
try:
   import xgboost
except:
   print('xgboost, if needed, will be installed and imported later')
try:
   import lightgbm
except:
   print('lightgbm, if needed, will be installed and imported later')
from sklearn.cluster import FeatureAgglomeration
import numpy
from numpy import inf, nan, dtype, mean
from autoai_libs.sklearn.custom_scorers import CustomScorers
import sklearn.ensemble
from autoai_libs.cognito.transforms.transform_utils import TExtras, FC
from autoai_libs.transformers.exportable import *
from autoai_libs.utils.exportable_utils import *
from sklearn.pipeline import Pipeline
known_values_list=[]
```

compose a decorator to assist pipeline instantiation via import of modules and installation of packages

```
def decorator_retries(func):
   def install_import_retry(*args, **kwargs):
      retries = 0
      successful = False
      failed_retries = 0
      while retries < 100 and failed_retries < 10 and not successful:
         retries += 1
         failed_retries += 1
         try:
```

```
            result = func(*args, **kwargs)
            successful = True
        except Exception as e:
            estr = str(e)
            if estr.startswith('name ') and estr.endswith(' is not defined'):
                try:
                    import importlib
                    module_name = estr.split("'")[1]
                    module = importlib.import_module(module_name)
                    globals().update({module_name: module})
                    print('import successful for ' + module_name)
                    failed_retries -= 1
                except Exception as import_failure:
                    print('import of ' + module_name + ' failed with: ' + str(import_failure))
                    import subprocess
                    if module_name == 'lightgbm':
                        try:
                            print('attempting pip install of ' + module_name)
                            process = subprocess.Popen('pip install ' + module_name, shell=True)
                            process.wait()
                        except Exception as E:
                            print(E)
                            try:
                                import sys
                                print('attempting conda install of ' + module_name)
                                process = subprocess.Popen('conda install --yes --prefix {sys.prefix} -c
powerai ' + module_name, shell = True)
                                process.wait()
                            except Exception as lightgbm_installation_error:
                                print('lightgbm installation failed!' + lightgbm_installation_error)
                    else:
                        print('attempting pip install of ' + module_name)
                        process = subprocess.Popen('pip install ' + module_name, shell=True)
                        process.wait()
                    try:
                        print('re-attempting import of ' + module_name)
                        module = importlib.import_module(module_name)
                        globals().update({module_name: module})
                        print('import successful for ' + module_name)
                        failed_retries -= 1
                    except Exception as import_or_installation_failure:
                        print('failure installing and/or importing ' + module_name + ' error was: ' +
str(
                            import_or_installation_failure))
                        raise (ModuleNotFoundError('Missing package in environment for ' +
module_name +
```

```
                                    '? Try import and/or pip install manually?'))
            elif type(e) is AttributeError:
                if 'module ' in estr and ' has no attribute ' in estr:
                    pieces = estr.split("'")
                    if len(pieces) == 5:
                        try:
                            import importlib
                            print('re-attempting import of ' + pieces[3] + ' from ' + pieces[1])
                            module = importlib.import_module('.' + pieces[3], pieces[1])
                            failed_retries -= 1
                        except:
                            print('failed attempt to import ' + pieces[3])
                            raise (e)
                    else:
                        raise (e)
                else:
                    raise (e)
        if successful:
            print('Pipeline successfully instantiated')
        else:
            raise (ModuleNotFoundError(
                'Remaining missing imports/packages in environment? Retry cell and/or try pip install manually?'))
        return result
    return install_import_retry
```

2. Compose Pipeline

```
# metadata necessary to replicate AutoAI scores with the pipeline
_input_metadata = {'target_label_name': 'Life expectancy ', 'learning_type': 'regression', 'run_uid': '62500913-681c-4966-a566-5ceb0dd602bf', 'pn': 'P3', 'cv_num_folds': 3, 'holdout_fraction': 0.15, 'optimization_metric': 'neg_root_mean_squared_error', 'pos_label': None, 'random_state': 33, 'data_source': ''}

# define a function to compose the pipeline, and invoke it
@decorator_retries
def compose_pipeline():
    import numpy
    from numpy import nan, dtype, mean
    #
    # composing steps for toplevel Pipeline
    #
    _input_metadata = {'target_label_name': 'Life expectancy ', 'learning_type': 'regression', 'run_uid': '62500913-681c-4966-a566-5ceb0dd602bf', 'pn': 'P3', 'cv_num_folds': 3, 'holdout_fraction': 0.15, 'optimization_metric': 'neg_root_mean_squared_error', 'pos_label': None, 'random_state': 33, 'data_source': ''}
```

```
    steps = []
    #
    # composing steps for preprocessor Pipeline
    #
    preprocessor__input_metadata = None
    preprocessor_steps = []
    #
    # composing steps for preprocessor_features FeatureUnion
    #
    preprocessor_features_transformer_list = []
    #
    # composing steps for preprocessor_features_categorical Pipeline
    #
    preprocessor_features_categorical__input_metadata = None
    preprocessor_features_categorical_steps = []
    preprocessor_features_categorical_steps.append(('cat_column_selector',
autoai_libs.transformers.exportable.NumpyColumnSelector(columns=[0, 1, 2, 7, 11, 13])))
    preprocessor_features_categorical_steps.append(('cat_compress_strings',
autoai_libs.transformers.exportable.CompressStrings(activate_flag=True,
compress_type='hash', dtypes_list=['char_str', 'int_num', 'char_str', 'float_int_num',
'float_int_num', 'float_int_num'], missing_values_reference_list=['', '-', '?', nan],
misslist_list=[[], [], [], [nan], [nan], [nan]])))
    preprocessor_features_categorical_steps.append(('cat_missing_replacer',
autoai_libs.transformers.exportable.NumpyReplaceMissingValues(filling_values=nan,
missing_values=[nan])))
    preprocessor_features_categorical_steps.append(('cat_unknown_replacer',
autoai_libs.transformers.exportable.NumpyReplaceUnknownValues(filling_values=nan,
filling_values_list=[nan, nan, nan, 100001, 100001, 100001],
known_values_list=[[147001224856526974832278065163338061039,
260699343329343420250263481462496204476,
116118337134451577549341133852425958601,
467511389534709377721758916694869753O3,
329516381543508488967110039675080977027,
707333284213987214973227661435778335O7,
212739582897968815372252009904290618533,
907348335631687700647559265365332851O1,
202797882335700325897457917503166655784,
150820500098510481813415427383903426184,
145877534885301049267215912567317643431,
146042710492699183527689945925558913871,
329038285037642035448290014465150269573,
665691591312895184526793193825614006O4,
134599758546830170946961007508765805273,
143673223417367409597806890422171484583,
435148240873518881718280135000974728O9,
210599618041562972070936847789988036902,
```

306190046251434926279001033014007339831,
257386254094586331070623087230906766309,
200250612019127811152210464978673413843,
144662901196442033596517283743993697168,
881625854716549439644439526212277392 61,
708969284789567717917084478846341671 42,
216684831300382153034372561484478 4907,
538405389624127231467251001177378497 69,
537835225801664894976920457896927455 62,
137433131769320511325160171555965548550,
901195658232428345540219593793457156 8,
100867498053965421926683001219366720175,
908714314979029656028785651557817883 27,
226182883875297231091414872587504708035,
298029156183361964131144349277717108303,
616690713383258750548474118972419460 95,
231725186024138418821343173350662523706,
317953072729355616170708786288199146493,
880032957927487990256591698635399168 64,
303846827209572410860542030980915489108,
117650741643591557067501287780038143451,
114382657829694160542291743577444141255,
688435158308810584788911486676096695 13,
311265945460282885639496306569138650308,
663363130508421849274852462009451050 96,
627855175476090413672379221051277581 6,
264133828541403902365817922285691706663,
237672356660114685772687796104882206514,
880721567167909398517429678152102123 61,
191655845835528033820508997679585417644,
135810150341533680846069152682895393814,
102836150724757230410725033869934678966,
301866387735507500246434251973708559904,
310276828594327150016809604736391170592,
217391883616977436792885859915168492233,
268223368701600857288201315803513880319,
158620802268026701334269447796294077650,
301204325272916423115124616267654497404,
113902216111633146684341394651838421184,
148167959092401686537001101821165870815,
403779725802210011384249441586366438 5,
336187489452407039394791603109816227725,
301285256860312455785994867529604192476,
312173900060238937270521904129323847731,
288027277195630534052428568134696149071,
307384067113960046097974087241019802729,

142816848122656347891616720396877128120,
926398978098771013701660213213179737 67,
197447874931603182353652347006218732046,
189153405801674659090813067540413446960,
603446537867084344241919552081482587 44,
275547366651759454322811788273436150404,
245291117150675811910476227597757516612,
324534391971762113271501167168766491614,
332939221964977781373840866644320659361,
243990464967881250218096721123374832934,
166412482558944353431194533546670637099,
934188325506260312770061788043005828 57,
100868194317926940154051177041401533500,
120809162883000100157782458277608403667,
915965732453594369293122591538697760 3,
120069515845204766982513747293765138127,
213085721284152195723899940840100181 74,
357083727578943720723862664043796285 53,
111185081072713903471978902125976427675,
147055629285626963240164699118015634288,
164962277741819786224663597464248928300,
197516484511383225722150740031787721089,
112016882230000284908020292505580427074,
693948927332904431213612325943283681 5,
125332411340882160702044993227052216001,
172336042347945209501542869303082370283,
213503579105731811632182799702211304277,
239759361309886661677188287055236569981,
310518635096297103991571280952035141839,
133154737170846526745897437786271419044,
154416530328912057507365665258369549472,
288469045950546730190193078847193098970,
848965961019870969114266672894906934 5,
241570600237737391575071415960427031848,
149654325505539066448015487336620561262,
838155520687440364816322364788531984 33,
130447722156717403042775871542217532557,
309775428472593936764999908740771513965,
194433639230741600743331401076929663747,
287105209433439742579762737349674202805,
105704239564513572043710144506496567 15,
188392261694543715233845217008066282321,
313379486574562434335207500262629208144,
249118345345366939574908646319607340890,
104442926034210402647486596299287548193,
102735254899278162311633387159256797168,

221751288586957193672400042500503422193,
445272982431317598285927213521415046 05,
140836680648177864032144374895547548555,
170055084650409142322787336947434585942,
221302440909767410133343183967811973171,
262018014442368137572517042065553566541,
797437811992203742657676060771281916 64,
254125374157180781180627978456850690728,
124301341133029451973017544367696533615,
284086970395620794029446821886845268390,
378400285825512928438414930137435177 02,
246455200204341180287241602094154852031,
143453842366453184196547081438629502439,
327575913974710642489356558011361228625,
152944767264824636164907421713899865093,
176501628211118575758228455305804789656,
159313682139237172887381305992317026546,
197432136167945168220584321832631475989,
311629718944900440174423894212659294477,
165862387931083018136327956233088224545,
200992098551941933530619672502718259277,
132112836046501443150745451782899562 39,
166016165608834327131607864783933314 88,
104166121630059556610737171074093387062,
209347361968514815068844481488500366786,
279050323757951516155403396511033587716,
264459266514432677140218732179794369611,
111378674825238759367412388488815025 31,
229483763589706052603319193282544725320,
244856816748827073573575820596953224849,
204069336107114946409550070958759494527,
637556697585320252140393497852503910 83,
280323114304267706123307379055557896439,
201141749708336609603825854380051921870,
924555942582725904899601970320043918 00,
181624327439045060956820224446253696114,
189246905584152577259182499180590580,
219924701173169887305713158245281315340,
194193646109240189407775880338662800887,
266654382484335115128215294640512574736,
138080610306521808113457784130292307 17,
192066837354053545233955008532086692251,
318124203124418465236696385800557006393,
154551519819035509041533706282186084707,
133278040923567948891376187569301318446,
329872229251628136616936033528845404207,

267113292827397478038595255593918245163,
781778904501795085908548917030423176 68,
159821081429796844717628326887647430719,
241470080325170968009603514017433045249,
215342376973477470594007954334026128 58,
331485374723644292946494024341588888803,
134882069537064921943379986368979541 58,
404813441397650284453555408564810189 98,
294240741554643331939894211760785349296,
132374248062000760689647888900256269745,
285590841995412959284881794170248825349,
453436736597654903672615127919573595 38,
326790586085090417519839246359297909943,
111259412569678040964004624811056505841,
319179760373270718270012367276983387589,
190656778724682718055384399764338191549,
274586769213526317171775513233049188730,
650147024746022644039286167901351303 94,
413004585287127200092288260942223958 70,
155901190345235079326433256810539957264,
213527539495502148379722298681646646713,
337664892250511449878491415602564465320,
149187898203650094159705085163227929822,
192946003755599889927910624585418472243,
151926378701363846636626095694661523383,
234826980371610571253952539720242605948,
209109830453134779187508710962259670256], [2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015], [316899882848231090403546313428573395088,
269864576280947885342328752525343332 77], [1.0, 2.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 11.0, 14.0, 15.0, 17.0, 18.0, 21.0, 22.0, 23.0, 24.0, 25.0, 26.0, 27.0, 28.0, 29.0, 31.0, 32.0, 33.0, 35.0, 36.0, 37.0, 38.0, 39.0, 41.0, 42.0, 43.0, 44.0, 45.0, 46.0, 47.0, 48.0, 49.0, 51.0, 52.0, 53.0, 54.0, 55.0, 56.0, 57.0, 58.0, 59.0, 61.0, 62.0, 63.0, 64.0, 65.0, 66.0, 67.0, 68.0, 69.0, 71.0, 72.0, 73.0, 74.0, 75.0, 76.0, 77.0, 78.0, 79.0, 81.0, 82.0, 83.0, 84.0, 85.0, 86.0, 87.0, 88.0, 89.0, 91.0, 92.0, 93.0, 94.0, 95.0, 96.0, 97.0, 98.0, 99.0], [3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 23.0, 24.0, 26.0, 31.0, 32.0, 33.0, 35.0, 36.0, 37.0, 38.0, 39.0, 41.0, 42.0, 43.0, 44.0, 45.0, 46.0, 47.0, 48.0, 49.0, 51.0, 52.0, 53.0, 54.0, 55.0, 56.0, 57.0, 58.0, 59.0, 61.0, 62.0, 63.0, 64.0, 65.0, 66.0, 67.0, 68.0, 69.0, 71.0, 72.0, 73.0, 74.0, 75.0, 76.0, 77.0, 78.0, 79.0, 81.0, 82.0, 83.0, 84.0, 85.0, 86.0, 87.0, 88.0, 89.0, 91.0, 92.0, 93.0, 94.0, 95.0, 96.0, 97.0, 98.0, 99.0], [2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 19.0, 21.0, 23.0, 24.0, 25.0, 26.0, 27.0, 28.0, 29.0, 31.0, 32.0, 33.0, 34.0, 35.0, 36.0, 37.0, 38.0, 39.0, 41.0, 42.0, 43.0, 44.0, 45.0, 46.0, 47.0, 48.0, 49.0, 51.0, 52.0, 53.0, 54.0, 55.0, 56.0, 57.0, 58.0, 59.0, 61.0, 62.0, 63.0, 64.0, 65.0, 66.0, 67.0, 68.0, 69.0, 71.0, 72.0, 73.0, 74.0, 75.0, 76.0, 77.0, 78.0, 79.0, 81.0, 82.0, 83.0, 84.0, 85.0, 86.0, 87.0, 88.0, 89.0, 91.0, 92.0, 93.0, 94.0, 95.0, 96.0, 97.0, 98.0, 99.0]], missing_values_reference_list=['', '-', '?', nan])))
    preprocessor_features_categorical_steps.append(('boolean2float_transformer', autoai_libs.transformers.exportable.boolean2float(activate_flag=True)))

```
    preprocessor_features_categorical_steps.append(('cat_imputer',
autoai_libs.transformers.exportable.CatImputer(activate_flag=True, missing_values=nan,
sklearn_version_family='20', strategy='most_frequent')))
    preprocessor_features_categorical_steps.append(('cat_encoder',
autoai_libs.transformers.exportable.CatEncoder(activate_flag=True, categories='auto',
dtype=numpy.float64, encoding='ordinal', handle_unknown='error',
sklearn_version_family='20')))
    preprocessor_features_categorical_steps.append(('float32_transformer',
autoai_libs.transformers.exportable.float32_transform(activate_flag=True)))
    # assembling preprocessor_features_categorical_ Pipeline
    preprocessor_features_categorical_pipeline =
sklearn.pipeline.Pipeline(steps=preprocessor_features_categorical_steps)
    preprocessor_features_transformer_list.append(('categorical',
preprocessor_features_categorical_pipeline))
    #
    # composing steps for preprocessor_features_numeric Pipeline
    #
    preprocessor_features_numeric__input_metadata = None
    preprocessor_features_numeric_steps = []
    preprocessor_features_numeric_steps.append(('num_column_selector',
autoai_libs.transformers.exportable.NumpyColumnSelector(columns=[3, 4, 5, 6, 8, 9, 10, 12,
14, 15, 16, 17, 18, 19, 20])))
    preprocessor_features_numeric_steps.append(('num_floatstr2float_transformer',
autoai_libs.transformers.exportable.FloatStr2Float(activate_flag=True,
dtypes_list=['float_int_num', 'int_num', 'float_num', 'float_num', 'int_num', 'float_num',
'int_num', 'float_num', 'float_num', 'float_num', 'float_num', 'float_num', 'float_num',
'float_num', 'float_num'], missing_values_reference_list=[nan])))
    preprocessor_features_numeric_steps.append(('num_missing_replacer',
autoai_libs.transformers.exportable.NumpyReplaceMissingValues(filling_values=nan,
missing_values=[nan])))
    preprocessor_features_numeric_steps.append(('num_imputer',
autoai_libs.transformers.exportable.NumImputer(activate_flag=True, missing_values=nan,
strategy='median')))
    preprocessor_features_numeric_steps.append(('num_scaler',
autoai_libs.transformers.exportable.OptStandardScaler(num_scaler_copy=None,
num_scaler_with_mean=None, num_scaler_with_std=None, use_scaler_flag=False)))
    preprocessor_features_numeric_steps.append(('float32_transformer',
autoai_libs.transformers.exportable.float32_transform(activate_flag=True)))
    # assembling preprocessor_features_numeric_ Pipeline
    preprocessor_features_numeric_pipeline =
sklearn.pipeline.Pipeline(steps=preprocessor_features_numeric_steps)
    preprocessor_features_transformer_list.append(('numeric',
preprocessor_features_numeric_pipeline))
    # assembling preprocessor_features_ FeatureUnion
    preprocessor_features_pipeline =
sklearn.pipeline.FeatureUnion(transformer_list=preprocessor_features_transformer_list)
```

```
    preprocessor_steps.append(('features', preprocessor_features_pipeline))
    preprocessor_steps.append(('permuter',
autoai_libs.transformers.exportable.NumpyPermuteArray(axis=0, permutation_indices=[0, 1,
2, 7, 11, 13, 3, 4, 5, 6, 8, 9, 10, 12, 14, 15, 16, 17, 18, 19, 20])))
    # assembling preprocessor_ Pipeline
    preprocessor_pipeline = sklearn.pipeline.Pipeline(steps=preprocessor_steps)
    steps.append(('preprocessor', preprocessor_pipeline))
    #
    # composing steps for cognito Pipeline
    #
    cognito__input_metadata = None
    cognito_steps = []
    cognito_steps.append(('0',
autoai_libs.cognito.transforms.transform_utils.TAM(tans_class=sklearn.decomposition.pca.P
CA(copy=True, iterated_power='auto', n_components=None, random_state=None,
svd_solver='auto', tol=0.0, whiten=False), name='pca', tgraph=None, apply_all=True,
col_names=['Country', 'Year', 'Status', 'Adult Mortality', 'infant deaths', 'Alcohol', 'percentage
expenditure', 'Hepatitis B', 'Measles ', ' BMI ', 'under-five deaths ', 'Polio', 'Total expenditure',
'Diphtheria ', ' HIV/AIDS', 'GDP', 'Population', ' thinness  1-19 years', ' thinness 5-9 years',
'Income composition of resources', 'Schooling'], col_dtypes=[dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32')],
col_as_json_objects=None)))
    cognito_steps.append(('1',
autoai_libs.cognito.transforms.transform_utils.FS1(cols_ids_must_keep=range(0, 21),
additional_col_count_to_keep=20, ptype='regression')))
    cognito_steps.append(('2',
autoai_libs.cognito.transforms.transform_utils.TA1(fun=numpy.sqrt, name='sqrt',
datatypes=['numeric'], feat_constraints=[autoai_libs.utils.fc_methods.is_non_negative,
autoai_libs.utils.fc_methods.is_not_categorical], tgraph=None, apply_all=True,
col_names=['Country', 'Year', 'Status', 'Adult Mortality', 'infant deaths', 'Alcohol', 'percentage
expenditure', 'Hepatitis B', 'Measles ', ' BMI ', 'under-five deaths ', 'Polio', 'Total expenditure',
'Diphtheria ', ' HIV/AIDS', 'GDP', 'Population', ' thinness  1-19 years', ' thinness 5-9 years',
'Income composition of resources', 'Schooling', 'pca_0', 'pca_1', 'pca_2', 'pca_3', 'pca_4',
'pca_5', 'pca_6', 'pca_7', 'pca_8', 'pca_9', 'pca_10', 'pca_11', 'pca_12', 'pca_13', 'pca_14',
'pca_15', 'pca_16', 'pca_17', 'pca_19', 'pca_20'], col_dtypes=[dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'),
```

```
dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32')],
col_as_json_objects=None)))
    cognito_steps.append(('3',
autoai_libs.cognito.transforms.transform_utils.FS1(cols_ids_must_keep=range(0, 21),
additional_col_count_to_keep=20, ptype='regression')))
    # assembling cognito_ Pipeline
    cognito_pipeline = sklearn.pipeline.Pipeline(steps=cognito_steps)
    steps.append(('cognito', cognito_pipeline))
    steps.append(('estimator', sklearn.linear_model.base.LinearRegression(copy_X=True,
fit_intercept=True, n_jobs=1, normalize=True)))
    # assembling  Pipeline
    pipeline = sklearn.pipeline.Pipeline(steps=steps)
    return pipeline
pipeline = compose_pipeline()
```

3. Extract needed parameter values from AutoAI run metadata

```
# Metadata used in retrieving data and computing metrics.  Customize as necessary for your
environment.
#data_source='replace_with_path_and_csv_filename'
target_label_name = _input_metadata['target_label_name']
learning_type = _input_metadata['learning_type']
optimization_metric = _input_metadata['optimization_metric']
random_state = _input_metadata['random_state']
cv_num_folds = _input_metadata['cv_num_folds']
holdout_fraction = _input_metadata['holdout_fraction']
if 'data_provenance' in _input_metadata:
    data_provenance = _input_metadata['data_provenance']
else:
    data_provenance = None
if 'pos_label' in _input_metadata and learning_type == 'classification':
    pos_label = _input_metadata['pos_label']
else:
    pos_label = None
```

4. Create dataframe from dataset in Cloud Object Storage

```
# @hidden_cell
# The following code contains the credentials for a file in your IBM Cloud Object Storage.
# You might want to remove those credentials before you share your notebook.
credentials_0 = {
    'ENDPOINT': 'https://s3-api.us-geo.objectstorage.softlayer.net',
    'IBM_AUTH_ENDPOINT': 'https://iam.bluemix.net/oidc/token/',
    'APIKEY': 'yTCnifPkUuTz5GQZuKDe6uyCPcQKkW2r2GGckNV516g3',
    'BUCKET': 'predictinglifeexpectancywithoutpy-donotdelete-pr-8anb8fa6sncmht',
    'FILE': 'datasets_12603_17232_Life Expectancy Data.csv',
```

```
    'SERVICE_NAME': 's3',
    'ASSET_ID': '1',
    }
#  Read the data as a dataframe
import pandas as pd

csv_encodings=['UTF-8','Latin-1'] # supplement list of encodings as necessary for your data
df = None
readable = None  # if automatic detection fails, you can supply a filename here

# First, obtain a readable object
# Cloud Object Storage data access
# Assumes COS credentials are in a dictionary named 'credentials_0'

credentials = df = globals().get('credentials_0')
if readable is None and credentials is not None :
    try:
        import types
        import pandas as pd
        import io
        import os
    except Exception as import_exception:
        print('Error with importing packages - check if you installed them on your environment')
    try:
        if credentials['SERVICE_NAME'] == 's3':
            try:
                from botocore.client import Config
                import ibm_boto3
            except Exception as import_exception:
                print('Installing required packages!')
                !pip install ibm-cos-sdk
                print('accessing data via Cloud Object Storage')
            try:
                cos_client = ibm_boto3.resource(service_name=credentials['SERVICE_NAME'],
                                                ibm_api_key_id=credentials['APIKEY'],
                                                ibm_auth_endpoint=credentials['IBM_AUTH_ENDPOINT'],
                                                config=Config(signature_version='oauth'),
                                                endpoint_url=credentials['ENDPOINT'])
            except Exception as cos_exception:
                print('unable to create client for cloud object storage')
            try:
                cos_client.meta.client.download_file(Bucket=credentials['BUCKET'],
Filename=credentials['FILE'], Key=credentials['FILE'])
            except Exception as cos_access_exception:
                print('unable to access data object in cloud object storage with credentials supplied')
            try:
                for encoding in csv_encodings:
                    df = pd.read_csv(credentials['FILE'], encoding = encoding, sep = None, engine =
'python')
                    os.remove(credentials['FILE'])
```

```
                print('Data loaded from cloud object storage with encoding ' + encoding)
                break
            except Exception as cos_object_read_exception:
                print('unable to access data object from cos object with encoding ' + encoding)
        elif credentials['SERVICE_NAME'] == 'fs':
            print('accessing data via File System')
            try:
                df = pd.read_csv(credentials['FILE'], sep = None, engine = 'python')
            except Exception as FS_access_exception:
                print('unable to access data object in File System with path supplied')
    except Exception as data_access_exception:
        print('unable to access data object with credentials supplied')

# IBM Cloud Pak for Data data access
project_filename = globals().get('project_filename')
if readable is None and 'credentials_0' in globals() and 'ASSET_ID' in credentials_0:
    project_filename = credentials_0['ASSET_ID']
if project_filename != 'None' and project_filename != '1':
    print('attempting project_lib access to ' + str(project_filename))
    try:
        from project_lib import Project
        project = Project.access()
        storage_credentials = project.get_storage_metadata()
        readable = project.get_file(project_filename)
    except Exception as project_exception:
        print('unable to access data using the project_lib interface and filename supplied')


# Use data_provenance as filename if other access mechanisms are unsuccessful
if readable is None and type(data_provenance) is str:
    print('attempting to access local file using path and name ' + data_provenance)
    readable = data_provenance

# Second, use pd.read_csv to read object, iterating over list of csv_encodings until successful
if readable is not None:
    for encoding in csv_encodings:
        try:
            df = pd.read_csv(readable, encoding=encoding, sep = None, engine = 'python')
            print('successfully loaded dataframe using encoding = ' + str(encoding))
            break
        except Exception as exception_csv:
            print('unable to read csv using encoding ' + str(encoding))
            print('handled error was ' + str(exception_csv))
    if df is None:
        print('unable to read file/object as a dataframe using supplied csv_encodings ' +
str(csv_encodings))
        print(f'Please use \'insert to code\' on data panel to load dataframe.')
        raise(ValueError('unable to read file/object as a dataframe using supplied csv_encodings
' + str(csv_encodings)))
```

```
if isinstance(df,pd.DataFrame):
    print('Data loaded succesfully')
```

5. Preprocess Data

```
# Drop rows whose target is not defined
target = target_label_name # your target name here
if learning_type == 'regression':
    df[target] = pd.to_numeric(df[target], errors='coerce')
df.dropna('rows', how='any', subset=[target], inplace=True)
# extract X and y
df_X = df.drop(columns=[target])
df_y = df[target]
# Detach preprocessing pipeline (which needs to see all training data)
preprocessor_index = -1
preprocessing_steps = []
for i, step in enumerate(pipeline.steps):
    preprocessing_steps.append(step)
    if step[0]=='preprocessor':
        preprocessor_index = i
        break
#if len(pipeline.steps) > preprocessor_index+1 and pipeline.steps[preprocessor_index + 1][0]
== 'cognito':
    #preprocessor_index += 1
    #preprocessing_steps.append(pipeline.steps[preprocessor_index])
if preprocessor_index >= 0:
    preprocessing_pipeline = Pipeline(memory=pipeline.memory, steps=preprocessing_steps)
    pipeline = Pipeline(steps=pipeline.steps[preprocessor_index+1:])
# Preprocess X
# preprocessor should see all data for cross_validate on the remaining steps to match autoai
scores
known_values_list.clear()  #  known_values_list is filled in by the preprocessing_pipeline if
needed
preprocessing_pipeline.fit(df_X.values, df_y.values)
X_prep = preprocessing_pipeline.transform(df_X.values)
```

6. Split data into Training and Holdout sets

```
# determine learning_type and perform holdout split (stratify conditionally)
if learning_type is None:
    # When the problem type is not available in the metadata, use the sklearn type_of_target to
determine whether to stratify the holdout split
    # Caution:  This can mis-classify regression targets that can be expressed as integers as
multiclass, in which case manually override the learning_type
    from sklearn.utils.multiclass import type_of_target
    if type_of_target(df_y.values) in ['multiclass', 'binary']:
        learning_type = 'classification'
    else:
        learning_type = 'regression'
    print('learning_type determined by type_of_target as:',learning_type)
else:
    print('learning_type specified as:',learning_type)

from sklearn.model_selection import train_test_split
```

```
if learning_type == 'classification':
    X, X_holdout, y, y_holdout = train_test_split(X_prep, df_y.values,
test_size=holdout_fraction, random_state=random_state, stratify=df_y.values)
else:
    X, X_holdout, y, y_holdout = train_test_split(X_prep, df_y.values,
test_size=holdout_fraction, random_state=random_state)
```

7. Generate features via Feature Engineering pipeline

```
#Detach Feature Engineering pipeline if next, fit it, and transform the training data
fe_pipeline = None
if pipeline.steps[0][0] == 'cognito':
    try:
        fe_pipeline = Pipeline(steps=[pipeline.steps[0]])
        X = fe_pipeline.fit_transform(X, y)
        X_holdout = fe_pipeline.transform(X_holdout)
        pipeline.steps = pipeline.steps[1:]
    except IndexError:
        try:
            print('Trying to compose pipeline with some of cognito steps')
            fe_pipeline = Pipeline(steps =
list([pipeline.steps[0][1].steps[0],pipeline.steps[0][1].steps[1]]))
            X = fe_pipeline.fit_transform(X, y)
            X_holdout = fe_pipeline.transform(X_holdout)
            pipeline.steps = pipeline.steps[1:]
        except IndexError:
            print('Composing pipeline without cognito steps!')
            pipeline.steps = pipeline.steps[1:]
```

8. Additional setup: Define a function that returns a scorer for the target's positive label

```
# create a function to produce a scorer for a given positive label
def make_pos_label_scorer(scorer, pos_label):
    kwargs = {'pos_label':pos_label}
    for prop in ['needs_proba', 'needs_threshold']:
        if prop+'=True' in scorer._factory_args():
            kwargs[prop] = True
    if scorer._sign == -1:
        kwargs['greater_is_better'] = False
    from sklearn.metrics import make_scorer
    scorer=make_scorer(scorer._score_func, **kwargs)
    return scorer
```

9. Fit pipeline, predict on Holdout set, calculate score, perform cross-validation

```
# fit the remainder of the pipeline on the training data
pipeline.fit(X,y)
# predict on the holdout data
y_pred = pipeline.predict(X_holdout)
# compute score for the optimization metric
# scorer may need pos_label, but not all scorers take pos_label parameter
```

```
from sklearn.metrics import get_scorer
scorer = get_scorer(optimization_metric)
score = None
#score = scorer(pipeline, X_holdout, y_holdout)  # this would suffice for simple cases
pos_label = None  # if you want to supply the pos_label, specify it here
if pos_label is None and 'pos_label' in _input_metadata:
   pos_label=_input_metadata['pos_label']
try:
   score = scorer(pipeline, X_holdout, y_holdout)
except Exception as e1:
   if pos_label is None or str(pos_label)=="":
      print('You may have to provide a value for pos_label in order for a score to be
calculated.')
      raise(e1)
   else:
      exception_string=str(e1)
      if 'pos_label' in exception_string:
         try:
            scorer = make_pos_label_scorer(scorer, pos_label=pos_label)
            score = scorer(pipeline, X_holdout, y_holdout)
            print('Retry was successful with pos_label supplied to scorer')
         except Exception as e2:
            print('Initial attempt to use scorer failed.  Exception was:')
            print(e1)
            print('')
            print('Retry with pos_label failed.  Exception was:')
            print(e2)
      else:
         raise(e1)
if score is not None:
   print(score)
# cross_validate pipeline using training data
from sklearn.model_selection import cross_validate
from sklearn.model_selection import StratifiedKFold, KFold
if learning_type == 'classification':
   fold_generator = StratifiedKFold(n_splits=cv_num_folds, random_state=random_state)
else:
   fold_generator = KFold(n_splits=cv_num_folds, random_state=random_state)
cv_results = cross_validate(pipeline, X, y, cv=fold_generator,
scoring={optimization_metric:scorer}, return_train_score=True)
import numpy as np
np.mean(cv_results['test_' + optimization_metric]) cv_results
```

OUTPUT:-

Node-Red UI URL:- https://node-red-uyvat.mybluemix.net/ui/#!/0?socketid=Vb-vecPVBdTNThDzAAAD

Result Image:-