SYNOPSIS

Cardiovascular diseases are the most common cause of death worldwide over the last few decades in the developed as well as underdeveloped and developing countries. Early detection of cardiac diseases and continuous supervision of clinicians can reduce the mortality rate. However, accurate detection of heart diseases in all cases and consultation of a patient for 24 hours by a doctor is not available since it requires more sapience, time and expertise. In this study, a tentative design of a cloud-based heart disease prediction system had been proposed to detect impending heart disease using (SVM)Machine learning techniques.

The heart is a kind of muscular organ which pumps blood into the body and is the central part of the body's cardiovascular system which also contains lungs. Cardiovascular system also comprises a network of blood vessels, for example, veins, arteries, and capillaries. These blood vessels deliver blood all over the body. Abnormalities in normal blood flow from the heart cause several types of heart diseases which are commonly known as cardiovascular diseases (CVD). Heart diseases are the main reasons for death worldwide. According to the survey of the World Health Organization (WHO), 17.5 million total global deaths occur because of heart attacks and strokes. More than 75% of deaths from cardiovascular diseases occur mostly in middle-income and low-income countries. Also, 80% of the deaths that occur due to CVDs are because of stroke and heart attack . Therefore, detection of cardiac abnormalities at the early stage and tools for the prediction of heart diseases can save a lot of life and help doctors to design an effective treatment plan which ultimately reduces the mortality rate due to cardiovascular diseases.

CHAPTER 1

INTRODUCTION

1.1. INTRODUCTION

Machne learning technique means the use of sophisticated data analysis tools to determine previously unknown, valid patterns and relationships in large data set. These tools can include statistical models, mathematical algorithm and machine learning methods in early detection of chronic disease.Medical data mining has great potential for exploring the hidden patterns in the data sets of the medical domain. These patterns can be utilized for clinical diagnosis. However, the available raw medical data are widely distributed, heterogeneous in nature, and voluminous. These data need to be collected in an organized form. This collected data can be then integrated to form a hospital information system. Data mining technology provides a user oriented approach to novel and hidden patterns in the data .Heart disease has considerably increased for the last two decades and become the leading cause of death for people in most of the countries in the world. World Health Organization (WHO) reported that 30% of death is due to heart disease . Nearly 17.3 million people died due to heart disease. More than 80% of passing away in world is because of coronary illness.

According to survey of WHO, 17 million total global deaths are due to heart attacks and strokes. The deaths due to heart disease in many countries occur due to work overload, mental stress and many other problems. On the whole it is found as primary reason behind death in adults. Diagnosis is complicated and important task that needs to be executed accurately and efficiently. The diagnosis is often made, based on doctor’s experience & knowledge. This leads to unwanted results & excessive medical costs of treatments provided to patients.

CHAPTER 2

LITERATURE SURVEY

The various papers HEART DISEASES PREDICTION USING SVM have been reviewed and the reviewed papers have been listed below.

2.1. LITERATURE REVIEW

1. **Data Mining based Fragmentation and Prediction of Medical Data**

In 2014 Dubey A. et al [1] India is set to witness a spike in deaths due to heart diseases. Early stage detection may prevent the death due to the heart diseases. In this paper we provide an efficient approach which is based on Data Mining and Ant Colony Optimization technique (DMACO) for Heart Disease Prediction so that we can prevent it in the earlier stages. For this we first taken the concept of data mining to finding the support, generated support is used as a weight of the symptom which will be the initial pheromone value of the ant. Then we consider Pain in the chest, Discomfort radiating to the back, choking feeling (heartburn), Nausea, Extreme weakness and Irregular heartbeats as the factor of heart attack. According to risk level identified we find the max pheromone value), max pheromone value is the addition of weight and the risk level. After applying the DMACO algorithm we can observe the increasing detection rate. So by this approach we can increase the detection probability in the early stage which is not generally detected in the earlier stage.

2. **Knowledge discovery and data mining in toxicology**

In 2014 Dr. Durairaj.M, Sivagowry.S el at[2] Medical Ecosystem is originated with rich information database, but inadequate in techniques to extract the information from the database. This is because of the lack of effective analysis tool to discover hidden relationships and trends in them. By applying the data mining techniques, valuable knowledge can be extracted from the health care system. Extracted knowledge can be applied for the accurate diagnosis of disease and proper treatment. Heart disease is the leading cause of death in all over the world over past ten years. Researchers have developed many hybrid data mining techniques for diagnosing heart disease. Here a preprocessing technique and analysis of the accuracy for prediction after preprocessing the noisy data explained. It is also observed that the accuracy has been increased to 91% after preprocessing. Swarm Intelligence techniques hybrided with Rough Set Algorithm are to be taken as future work for exact reduction of relevant features for prediction

3. **Therapeutic regulation of endothelial dysfunction in type 2 diabetes mellitus**

In 2014 Masethe H.D. el at [3] The heart disease accounts to be the leading cause of death worldwide. It is being found tough to predict the heart attack as it is a complex task that requires experience and knowledge to medical experts. The health sector today contains hidden information that can be important in making decisions. Here some mining algorithms like Naïve Bayes, REPTREE, J48, CART, and Bayes Net are used for the efficient prediction heart attacks. The research result found prediction accuracy of 99%.

4. **An Efficient Data Mining and Ant Colony Optimization technique (DMACO) for Heart Disease Prediction**

In 2013 Kumar S.,Kuar G el at[4] the Nowadays the use of computer technology in the fields of medicine area diagnosis, treatment of illnesses and patient pursuit has highly increased The objective of this paper is to detect the heart diseases in the person by using Fuzzy Expert System. The designed system based on the Parvati Devi hospital, Ranjit Avenue and EMC hospital Amritsar and International

Lab data base. The system consists of 6 input fields and two output field. Input fields are chest pain type, cholesterol, maximum heart rate, blood pressure, blood sugar, old peak. By the obtained result field presence of heart disease in the patient and precautions accordingly has been detected. It is integer valued from 0 (no presence) to 1 (distinguish presence (values 0.1 to 1.0). We can use the Mamdani inference method. The generated outcomes developed system are comparatively analyzed. This observation found correct 92%.

5. **A Pragmatic Approach of Preprocessing the Data Set for Heart Disease Prediction**

In 2012, Muhammed et al. [5] present and discuss the experiment that was executed with naïve bayes technique in order to build predictive model as an artificial diagnose for heart disease based on data set which contains set of parameters that were measured for individuals previously. Then they compare the results with other techniques according to using the same data that were given from UCI repository data.

6. **Prediction of Heart Disease using Classification Algorithms**

In 2012 Dangare c.s. et al [6] present in the Healthcare industry is generally “information rich”, but unfortunately not all the data are mined which is required for discovering hidden patterns & effective decision making. To discover knowledge from database and for the purpose of medical research, particularly in Heart disease prediction involvement and use of specific techniques of data mining required and implemented. This paper has analyzed prediction systems for Heart disease using more number of input attributes. The system uses medical terms such as sex, blood pressure, cholesterol like 13 attributes to predict the likelihood of patient getting a Heart disease. Until now, 13 attributes are used for prediction. This research paper added two more attributes i.e. obesity and smoking. With the different classification methods Heart Attack databases analyzed. The performance of these techniques is compared, based on accuracy. As per our results accuracy of Neural Networks, Decision Trees, and Naive Bayes are 100%, 99.62%, and 90.74% respectively. By the analysis it is found that Neural Network technique predicts Heart disease with highest accuracy.

7. **Detection of Heart Diseases using Fuzzy Logic**

In 2012 bhatla N. el at[7] present and discuss Cardiovascular disease is a term used to describe a variety of heart diseases, illnesses, and events that impact the heart and circulatory system. A medical experts use several sources of data and tests to make a diagnostic impression but it is not necessary that all the tests are useful for the diagnosis of a heart disease. The objective of our work is to reduce the number of attributes used in heart disease diagnosis that will automatically reduce the number of tests which are required to be taken by a patient. Our work also aims at increasing the efficiency of the proposed system. The observations illustrated that Decision Tree and Naive Bayes using fuzzy logic has outplayed over other data mining techniques.

8. **Using data mining technique to diagnosis heart disease**

In 2008, Tsipouras, Markos G. et al. [8] presented a fuzzy rule-based decision support system (DSS) for the diagnosis of coronary artery disease (CAD). The system is automatically generated from an initial annotated dataset, using a four stage methodology. The dataset used for the DSS generation and evaluation consists of 199 subjects, each one characterized by 19 features, including demographic and history data, as well as laboratory examinations. Here a cross validation is applied 10 times which produces average sensitivity 62% and specificity 54%, using the set of rules extracted from the decision tree (first and second stages), while the average sensitivity and specificity increase to 80% and 65%, respectively, when the fuzzification and optimization stages are used. The system provides CAD diagnosis based on easily and noninvasively acquired features, and is able to provide interpretation for the decisions made.

9. **Improved Study of Heart Disease Prediction System using Data Mining Classification Techniques**

In 2010, Yosawin Kangwanariyakul, Chanin Nantasenamat et al. [9] tells about the Ischemic Heart Disease (IHD) which is a major cause of death. Early and accurate detection of IHD along with rapid diagnosis are important for reducing the mortality rate.

10,. **A Novel Approach for Heart Disease Diagnosis using Data Mining and Fuzzy Logic**

In 2010, Srinivas, K. et al. [10] tell that Heart disease (HD) is a major cause of morbidity and mortality in the modern society. The preparation of medical diagnosis is noteworthy keen observation process but it is important to do so perfectly and accurately with efficiently.

11. **Automated Diagnosis of Coronary Artery Disease Based on Data Mining and Fuzzy Modeling**

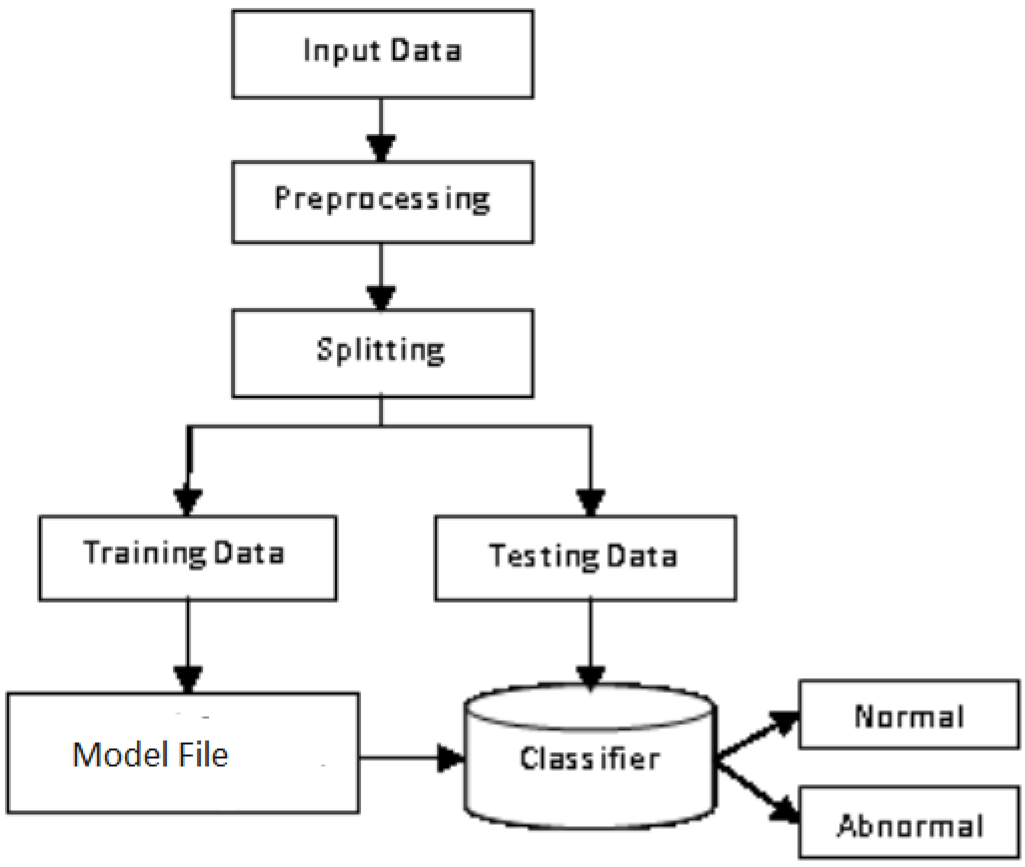
In 2012, Muhammed et al. [11] present and discuss the experiment that was executed with naïve bayes technique in order to build predictive model as an artificial diagnose for heart disease based on data set which contains set of parameters that were measured for individuals previously. Then they compare the results with other techniques according to using the same data that were given from UCI repository data.

CHAPTER 3

3.1. INTRODUCTION

Here our system uses svm algorithm to classify the result.The several parameters chest pain type, blood pressure , cholesterol and many more values are integrated in a data set and imported for training process and once the training process completed it creates the model file and after that the testing process is made the several require parameters of patient is entered and imported as input to svm the svm compares the imported data with the model file and classifies the final result and final result is sent to cloud and viewed by the application.

3.2. BLOCK DIAGRAM



3.3. BLOCK DIAGRAM DESCRIPTION

3.3.1. SVM:

A Support Vector Machine (SVM) is a supervised machine learning algorithm that can be employed for both classification and regression purposes. SVMs are more commonly used in classification

3.3.2. SVM USES:

SVM is used for text classification tasks such as category assignment, detecting spam and sentiment analysis and mainly used for result prediction . It is also commonly used for image recognition challenges, performing particularly well in aspect-based recognition and color-based classification. SVM also plays a vital role in many areas of handwritten digit recognition, such as postal automation services.

3.3.3. ADVANTAGES:

- Accuracy
- Works well on smaller cleaner datasets
- It can be more efficient because it uses a subset of training points

3.3.4. OVERVIEW:

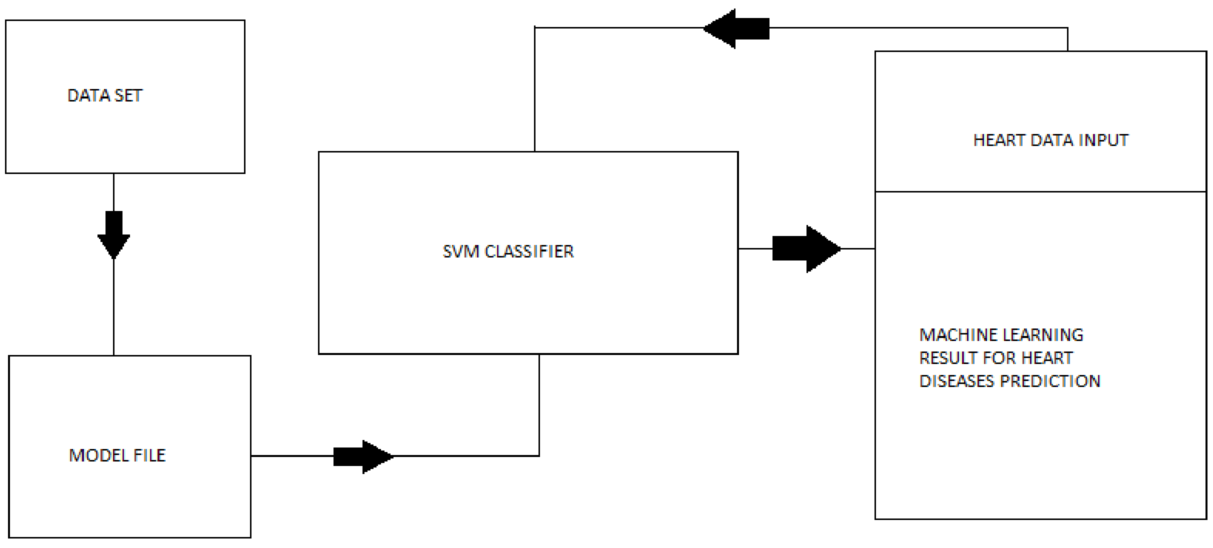
Machine learning involves predicting and classifying data and to do so we employ various machine learning algorithms according to the dataset.

SVM or Support Vector Machine is a linear model for classification and regression problems. It can solve linear and non-linear problems and work well for many practical problems. The idea of SVM is simple: The algorithm creates a line or a hyperplane which separates the data into classes.

3.3.5. WORKING:

A support vector machine is a machine learning model that is able to generalise between two different classes if the set of labelled data is provided in the training set to the algorithm. The main function of the SVM is to check for that hyperplane that is able to distinguish between the two classes.

There can be many hyperplanes that can do this task but the objective is to find that hyperplane that has the highest margin that means maximum distances between the two classes, so that in future if a new data point comes that is two be classified then it can be classified easily.



The input heart parameter and age is entered through mobile which is sent to cloud and fetched by the python script and svm machine learning algorithm is performed and result is predicted the predicted result is sent to cloud and displayed in the mobile

SVM PACKAGES:

NumPy is a Python package. It stands for 'Numerical Python'. It is a library consisting of multidimensional array objects and a collection of routines for processing of array. Numeric, the ancestor of NumPy, was developed by Jim Hugunin. Another package Numarray was also developed, having some additional functionalities. In 2005, Travis Oliphant created NumPy package by incorporating the features of Numarray into Numeric package. There are many contributors to this open source project.

Operations using NumPy:

Using NumPy, a developer can perform the following operations:

- Mathematical and logical operations on arrays.
- Fourier transforms and routines for shape manipulation.
- Operations related to linear algebra. NumPy has in-built functions for linear algebra and random number generation.

The most important object defined in NumPy is an N-dimensional array type called ndarray. It describes the collection of items of the same type. Items in the collection can be accessed using a zero-based index. Every item in an ndarray takes the same size of block in the memory. Each element in ndarray is an object of data-type object (called dtype).

Data Type Objects(dtype)

A data type object describes interpretation of fixed block of memory corresponding to an array, depending on the following aspects:

- Type of data (integer, float or Python object)
- Size of data
- Byte order (little-endian or big-endian)
- In case of structured type, the names of fields, data type of each field and part of the memory block taken by each field

• If data type is a subarray, its shape and data type The byte order is decided by prefixing '<' or '>' to data type. '<' means that encoding is littleendian (least significant is stored in smallest address)

TENSORFLOW:

TensorFlow is an interface for expressing machine learning algorithms, and an implementation for executing such algorithms. A computation expressed using TensorFlow can be executed with little or no change on a wide variety of heterogeneous systems, ranging from mobile devices such as phones and tablets up to large-scale distributed systems of hundreds of machines and thousands of computational devices such as GPU cards. The system is flexible and can be used to express a wide variety of algorithms, including training and inference algorithms for deep neural network models, and it has been used for conducting research and for deploying machine learning systems into production across more than a dozen areas of computer science and other fields, including speech recognition, computer vision, robotics, information retrieval, natural language processing, geographic information extraction, and computational drug discovery. This paper describes the TensorFlow interface and an implementation of that interface that we have built at Google. The TensorFlow API and a reference implementation were released as an open-source package under the Apache 2.0 license in November, 2015 and are available at www.tensorflow.org.

Based on our experience with DistBelief and a more complete understanding of the desirable system properties and requirements for training and using neural networks, we have built TensorFlow, our second-generation system for the implementation and deployment of largescale machine learning models. TensorFlow takes computations described using a dataflow-like model and maps them onto a wide variety of different hardware platforms, ranging from running inference on mobile device platforms such as Android and iOS to modestsized training and inference systems using single machines containing one or many GPU cards to large-scale training systems running on hundreds of specialized machines with thousands of GPUs. Having a single system that can span such a broad range of platforms significantly simplifies the real-world use of machine learning system, as we have found that having separate systems for large-scale training and small-scale deployment leads to significant maintenance burdens and leaky abstractions. TensorFlow computations are expressed as stateful dataflow graphs (described in more detail in Section 2), and we have focused on making the system both flexible enough for quickly experimenting with new models for research purposes and sufficiently high performance and robust for production training and deployment of machine learning models. For scaling neural network training to larger deployments, TensorFlow allows clients to easily express various kinds of parallelism through replication and parallel execution of a core model dataflow graph, with many different computational devices all collaborating to update a set of shared parameters or other state. Modest changes in the description of the computation allow a wide variety of different approaches to parallelism to be achieved and tried with low effort [14, 29, 42]. Some TensorFlow uses allow some flexibility in terms of the consistency of parameter updates, and we can easily express and take advantage of these relaxed synchronization requirements in some of our larger deployments. Compared to DistBelief, TensorFlow's programming model is more flexible, its performance is significantly better, and it supports training and using a broader range of models on a wider variety of heterogeneous hardware platforms.

In a TensorFlow graph, each node has zero or more inputs and zero or more outputs, and represents the instantiation of an operation. Values that flow along normal edges in the graph (from outputs to inputs) are tensors, arbitrary dimensionality arrays where the underlying element type is specified or inferred at graph-construction time. Special edges, called control dependencies, can also exist in the graph: no data flows along such edges, but they indicate that the source node for the control dependence must finish executing before the destination node for the control dependence starts executing. Since our model includes mutable state, control dependencies can be used directly by clients to enforce happens before relationships. Our implementation also sometimes inserts control dependencies to enforce orderings between otherwise independent operations as a way of, for example, controlling the peak memory usage.

TENSORFLOW IMPLEMENTATION:

The main components in a TensorFlow system are the client, which uses the Session interface to communicate with the master, and one or more worker processes, with each worker process responsible for arbitrating access to one or more computational devices (such as CPU cores or GPU cards) and for executing graph nodes on those devices as instructed by the master. We have both local and distributed implementations of the TensorFlow interface. The local implementation is used when the client, the master, and the worker all run on a single machine in the context of a single operating system process (possibly with multiple devices, if for example, the machine has many GPU cards installed). The distributed implementation shares most of the code with the local implementation, but extends it with support for an environment where the client, the master, and the workers can all be in different processes on different machines. In our distributed environment, these different tasks are containers in jobs managed by a cluster scheduling system [51]. These two different modes are illustrated . Most of the rest of this section discusses issues that are common to both implementations, while This discusses some issues that are particular to the distributed implementation.

Data Parallel Training

One simple technique for speeding up SGD is to parallelize the computation of the gradient for a mini-batch across mini-batch elements. For example, if we are using a mini-batch size of 1000 elements, we can use 10 replicas of the model to each compute the gradient for 100 elements, and then combine the gradients and apply updates to the parameters synchronously, in order to behave exactly as if we were running the sequential SGD algorithm with a batch size of 1000 elements. In this case, the TensorFlow graph simply has many replicas of the portion of the graph that does the bulk of the model computation, and a single client thread drives the entire training loop for this large graph. The TensorFlow system shares some design characteristics with its predecessor system, Disbelief , and with later systems with similar designs like Project Adam and the Parameter Server project . Like Disbelief and Project Adam, TensorFlow allows computations to be spread out across many computational devices across many machines, and allows users to specify machine learning models using relatively high-level descriptions. Unlike DistBelief and Project Adam, though, the general-purpose dataflow graph model in TensorFlow is more flexible and more amenable to expressing a wider variety of machine learning models and optimization algorithms. It also permits a significant simplification by allowing the expression of stateful parameter nodes as variables, and variable update operations that are just additional nodes in the graph; in contrast, DistBelief, Project Adam and the Parameter Server systems all have whole separate parameter server subsystems devoted to communicating and updating parameter values. The Halide system [40] for expressing image processing pipelines uses a similar intermediate representation to the TensorFlow dataflow graph. Unlike TensorFlow, though, the Halide system actually has higherlevel knowledge of the semantics of its operations and uses this knowledge to generate highly optimized pieces of code that combine multiple operations, taking into account parallelism and locality. Halide runs the resulting computations only on a single machine, and not in a distributed setting. In future work we are hoping to extend TensorFlow with a similar cross-operation dynamic compilation framework. Like TensorFlow, several other distributed systems have been developed for executing dataflow graphs across a cluster. Dryad [24] and Flume [8] demonstrate how a complex workflow can be represented as a dataflow graph. CIEL [37] and Naiad [36] introduce generic support for data-dependent control flow: CIEL represents iteration as a DAG that dynamically unfolds, whereas Naiad uses a static graph with cycles to support lower-latency iteration. Spark [55] is optimized for computations that access the same data repeatedly, using “resilient distributed datasets” (RDDs), which are soft-state cached outputs of earlier computations. Dandelion [44] executes dataflow graphs across a cluster of heterogeneous devices, including GPUs. TensorFlow uses a hybrid dataflow model that borrows elements from each of these systems. Its dataflow scheduler, which is the component that chooses the next node to execute, uses the same basic algorithm as Dryad, Flume, CIEL, and Spark. Its distributed architecture is closest to Naiad, in that the system uses a single, optimized dataflow graph to represent the entire computation, and caches information about that graph on each device to minimize coordination overhead. Like Spark and Naiad, TensorFlow works best when there is sufficient RAM in the cluster to hold the working set of the computation. Iteration in TensorFlow uses a hybrid approach: multiple replicas of the same dataflow graph may be executing at once, while sharing the same set of variables. Replicas can share data asynchronously through the variables, or use synchronization mechanisms in the graph, such as queues, to operate synchronously. TensorFlow also supports iteration within a graph, which is a hybrid of CIEL and Naiad: for simplicity, each node fires only when all of its inputs are ready (like CIEL); but for efficiency the graph is represented as a static, cyclic dataflow (like Naiad).

KERAS:

There are two types of models available in Keras: the Sequential model and the Model class used with functional API.

These models have a number of methods in common:

- `model.summary()`: prints a summary representation of your model. Shortcut for utils.print_summary
- `model.get_config()`: returns a dictionary containing the configuration of the model. The model can be reinstantiated from its config via:

```
config = model.get_config()
```

```
model = Model.from_config(config)
```

```
# or, for Sequential:
```

```
model = Sequential.from_config(config)
```

- `model.get_weights()`: returns a list of all weight tensors in the model, as Numpy arrays.
- `model.set_weights(weights)`: sets the values of the weights of the model, from a list of Numpy arrays. The arrays in the list should have the same shape as those returned by `get_weights()`.
- `model.to_json()`: returns a representation of the model as a JSON string. Note that the representation does not include the weights, only the architecture. You can reinstantiate the same model (with reinitialized weights) from the JSON string via:

```
from keras.models import model_from_json
```

```
json_string = model.to_json()
```

```
model = model_from_json(json_string)
```

- `model.to_yaml()`: returns a representation of the model as a YAML string. Note that the representation does not include the weights, only the architecture. You can reinstantiate the same model (with reinitialized weights) from the YAML string via:

```
from keras.models import model_from_yaml
```

```
yaml_string = model.to_yaml()
```

```
model = model_from_yaml(yaml_string)
```

- `model.save_weights(filepath)`: saves the weights of the model as a HDF5 file.
- `model.load_weights(filepath, by_name=False)`: loads the weights of the model from a HDF5 file (created by `save_weights`). By default, the architecture is expected to be unchanged. To load weights into a different architecture (with some layers in common), use `by_name=True` to load only those layers with the same name.

Sequential:

Useful attributes of Model

- `model.layers` is a list of the layers added to the model.

Sequential model methods

compile

```
compile(self, optimizer, loss, metrics=None, sample_weight_mode=None, weighted_metrics=None,
target_tensors=None)
```

Configures the model for training.

Arguments

- **optimizer**: String (name of optimizer) or optimizer object. See optimizers.
- **loss**: String (name of objective function) or objective function. See losses. If the model has multiple outputs, you can use a different loss on each output by passing a dictionary or a list of losses. The loss value that will be minimized by the model will then be the sum of all individual losses.
- **metrics**: List of metrics to be evaluated by the model during training and testing. Typically you will use `metrics=['accuracy']`. To specify different metrics for different outputs of a multi-output model, you could also pass a dictionary, such as `metrics={'output_a': 'accuracy'}`.
- **sample_weight_mode**: If you need to do timestep-wise sample weighting (2D weights), set this to `"temporal"`. `None`defaults to sample-wise weights (1D). If the model has multiple outputs, you can use a different `sample_weight_mode` on each output by passing a dictionary or a list of modes.

- **weighted_metrics**: List of metrics to be evaluated and weighted by sample_weight or class_weight during training and testing.
- **target_tensors**: By default, Keras will create a placeholder for the model's target, which will be fed with the target data during training. If instead you would like to use your own target tensor (in turn, Keras will not expect external Numpy data for these targets at training time), you can specify them via the `target_tensors` argument. It should be a single tensor (for a single-output `Sequential` model).
- ****kwargs**: When using the Theano/CNTK backends, these arguments are passed into `K.function`. When using the TensorFlow backend, these arguments are passed into `tf.Session.run`.

Raises

- **ValueError**: In case of invalid arguments for `optimizer`, `loss`, `metrics` or `sample_weight_mode`.

Example

```
model = Sequential()
model.add(Dense(32, input_shape=(500,)))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

fit

```
fit(self, x=None, y=None, batch_size=None, epochs=1, verbose=1, callbacks=None,
validation_split=0.0, validation_data=None, shuffle=True, class_weight=None,
sample_weight=None, initial_epoch=0, steps_per_epoch=None, validation_steps=None)
```

Trains the model for a fixed number of epochs (iterations on a dataset).

Arguments

- **x**: Numpy array of training data. If the input layer in the model is named, you can also pass a dictionary mapping the input name to a Numpy array. `x` can be `None` (default) if feeding from framework-native tensors (e.g. TensorFlow data tensors).

- **y**: Numpy array of target (label) data. If the output layer in the model is named, you can also pass a dictionary mapping the output name to a Numpy array. `y` can be `None` (default) if feeding from framework-native tensors (e.g. TensorFlow data tensors).
- **batch_size**: Integer or `None`. Number of samples per gradient update. If unspecified, it will default to 32.
- **epochs**: Integer. Number of epochs to train the model. An epoch is an iteration over the entire `x` and `y` data provided. Note that in conjunction with `initial_epoch`, `epochs` is to be understood as "final epoch". The model is not trained for a number of iterations given by `epochs`, but merely until the epoch of index `epochs` is reached.
- **verbose**: 0, 1, or 2. Verbosity mode. 0 = silent, 1 = progress bar, 2 = one line per epoch.
- **callbacks**: List of `keras.callbacks.Callback` instances. List of callbacks to apply during training. See callbacks.
- **validation_split**: Float between 0 and 1. Fraction of the training data to be used as validation data. The model will set apart this fraction of the training data, will not train on it, and will evaluate the loss and any model metrics on this data at the end of each epoch. The validation data is selected from the last samples in the `x` and `y` data provided, before shuffling.
- **validation_data**: tuple `(x_val, y_val)` or tuple `(x_val, y_val, val_sample_weights)` on which to evaluate the loss and any model metrics at the end of each epoch. The model will not be trained on this data. This will override `validation_split`.
- **shuffle**: Boolean (whether to shuffle the training data before each epoch) or str (for 'batch'). 'batch' is a special option for dealing with the limitations of HDF5 data; it shuffles in batch-sized chunks. Has no effect when `steps_per_epoch` is not `None`.
- **class_weight**: Optional dictionary mapping class indices (integers) to a weight (float) value, used for weighting the loss function (during training only). This can be useful to tell the model to "pay more attention" to samples from an under-represented class.
- **sample_weight**: Optional Numpy array of weights for the training samples, used for weighting the loss function (during training only). You can either pass a flat (1D) Numpy array with the same length as the input samples (1:1 mapping between weights and samples), or in the case of temporal data, you can pass a 2D array with shape `(samples, sequence_length)`, to apply a different weight to every timestep of every sample. In this case you should make sure to specify`sample_weight_mode="temporal"` in `compile()`.
- **initial_epoch**: Epoch at which to start training (useful for resuming a previous training run).
- **steps_per_epoch**: Total number of steps (batches of samples) before declaring one epoch finished and starting the next epoch. When training with input tensors such as TensorFlow data tensors, the default `None` is equal to the number of samples in your dataset divided by the batch size, or 1 if that cannot be determined.
- **validation_steps**: Only relevant if `steps_per_epoch` is specified. Total number of steps (batches of samples) to validate before stopping.

Returns

A `History` object. Its `History.history` attribute is a record of training loss values and metrics values at successive epochs, as well as validation loss values and validation metrics values (if applicable).

Raises

- **RuntimeError**: If the model was never compiled.
- **ValueError**: In case of mismatch between the provided input data and what the model expects.

evaluate

```
evaluate(self, x=None, y=None, batch_size=None, verbose=1, sample_weight=None, steps=None)
```

Computes the loss on some input data, batch by batch.

Arguments

- **x**: input data, as a Numpy array or list of Numpy arrays (if the model has multiple inputs). `x` can be `None` (default) if feeding from framework-native tensors (e.g. TensorFlow data tensors).
- **y**: labels, as a Numpy array. `y` can be `None` (default) if feeding from framework-native tensors (e.g. TensorFlow data tensors).
- **batch_size**: Integer. If unspecified, it will default to 32.
- **verbose**: verbosity mode, 0 or 1.
- **sample_weight**: sample weights, as a Numpy array.
- **steps**: Integer or `None`. Total number of steps (batches of samples) before declaring the evaluation round finished. Ignored with the default value of `None`.

Returns

Scalar test loss (if the model has no metrics) or list of scalars (if the model computes other metrics). The attribute `model.metrics_names` will give you the display labels for the scalar outputs.

Raises

- **RuntimeError**: if the model was never compiled.

predict

```
predict(self, x, batch_size=None, verbose=0, steps=None)
```

Generates output predictions for the input samples.

The input samples are processed batch by batch.

Arguments

- **x**: the input data, as a Numpy array.
- **batch_size**: Integer. If unspecified, it will default to 32.
- **verbose**: verbosity mode, 0 or 1.
- **steps**: Total number of steps (batches of samples) before declaring the prediction round finished. Ignored with the default value of `None`.

Returns

A Numpy array of predictions.

train_on_batch

```
train_on_batch(self, x, y, class_weight=None, sample_weight=None)
```

Single gradient update over one batch of samples.

Arguments

- **x**: input data, as a Numpy array or list of Numpy arrays (if the model has multiple inputs).
- **y**: labels, as a Numpy array.
- **class_weight**: dictionary mapping classes to a weight value, used for scaling the loss function (during training only).
- **sample_weight**: sample weights, as a Numpy array.

Returns

Scalar training loss (if the model has no metrics) or list of scalars (if the model computes other metrics). The attribute `model.metrics_names` will give you the display labels for the scalar outputs.

Raises

- **RuntimeError**: if the model was never compiled.

test_on_batch

```
test_on_batch(self, x, y, sample_weight=None)
```

Evaluates the model over a single batch of samples.

Arguments

- **x**: input data, as a Numpy array or list of Numpy arrays (if the model has multiple inputs).
- **y**: labels, as a Numpy array.
- **sample_weight**: sample weights, as a Numpy array.

Returns

Scalar test loss (if the model has no metrics) or list of scalars (if the model computes other metrics). The attribute `model.metrics_names` will give you the display labels for the scalar outputs.

Raises

- **RuntimeError**: if the model was never compiled.

CHAPTER 4

SOFTWARE DESCRIPTION

4.1. SOFTWARE DESCRIPTION

PYTHON:

PYTHON 3.7:

Python is an interpreter, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace.

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object- oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in manya reason most platforms and

may be freely distributed. The same site also contains distributions of and pointers to many free third party Python modules, programs and tools, and additional documentation. The Python interpreter is easily extended with new functions and data types implemented in C or C++ (or other languages callable from C). Python is also suitable as an extension language for customizable applications. This tutorial introduces the reader informally to the basic concepts and features of the Python language and system. It helps to have a Python interpreter handy for hands-on experience, but all examples are self-contained, so the tutorial can be read off- line as well. For a description of standard objects and modules, see library-index. Reference-index gives a more formal definition of the language. To write extensions in C or C++, read extending-index and c-api-index. There are also several books covering Python in depth. This tutorial does not attempt to be comprehensive and cover every single feature, or even every commonly used feature. Instead, it introduces many of Python's most notes worthy features, and will give you a good idea of the language's flavor and style. After reading it, you will be able to read and write Python modules and programs, and you will be ready to learn more about the various Python library modules described in library-index. If you do much work on computers, eventually you find that there's some task you'd like to automate. For example, you may wish to perform a search-and-replace over a large number of text files, or rename and rearrange a bunch of photo files in a complicated way. Perhaps you'd like to write a small custom database, or a specialized GUI application or a simple game. If you're a professional software developer, you may have to work with several C/C++/Java libraries but find the usual write/compile/test/re-compile cycle is too slow. Perhaps you're writing a test suite for such a library and find writing the testing code a tedious task. Or maybe you've written a program that could use an extension language, and you don't want to design and implement a whole new language for your application.

Typing an end-of-file character (Control-D on Unix, Control-Z on Windows) at the primary prompt causes the interpreter to exit with a zero exit status. If that doesn't work, you can exit the interpreter by typing the following command: quit(). The interpreter's line-editing features include interactive editing, history substitution and code completion on systems that support read line. Perhaps the quickest check to see whether command line editing is supported is typing Control-P to the first Python prompt you get. If it beeps, you have command line editing; see Appendix Interactive Input Editing and History Substitution for an introduction to the keys. Ifnothing appears to happen, or if ^P is echoed, command line editing isn't available; you'll only be able to use backspace to remove characters from the current line. The interpreter operates somewhat like the Unix shell: when called with standard input connected to a tty device, it reads and executes commands interactively; when called with a file name argument or with a file as standard input, it reads and executes a script from that file. A second way of starting the interpreter is python -c command [arg] ..., which executes the statement(s) in command, analogous to the shell's -c option. Since Python statements often contain spaces or other characters that are special to the shell, it is usually advised to quote commands in its entiretywithsinglequotes.SomePythonmodulesarealsousefulasscripts. These can be invoked using python-m module [arg]...,which executes the source file for the module as if you had spelled out its full name on the command line. When a script file is used, it is sometimes useful to be able to run the script and enter interactive mode afterwards. This can be done by passing -i before the script.

There are tools which use doc strings to automatically produce online or printed documentation or to let the user interactively browse through code; it's good practice to include doc strings in code that you write, so make a habit of it. The execution of a function introduces a new symbol table usedfor the local variables of the function. More precisely, all variable assignments in a functions to read the value in the local symbol table; whereas variable references first look in the local symbol table, then in the local symbol tables of enclosing functions, then in the global symbol table, and finally in the table of built-in names. Thus, global variables cannot be directly assigned a value within a function (unless named in a global statement), although they may be referenced. The actual parameters (arguments) to a function call are introduced in the local symbol table of the called function when it is called; thus, arguments are passed using call by value (where the value is always an object reference, not the value of the object).1 When a function calls another function, a new local symbol table is created for that call. A function definition introduces the function name in the current symbol table. The value of the function name has a type that is recognized by the interpreter as a user-defined function. This value can be assigned to another name which can then also be used as a function.

Annotations are stored in the annotations attribute of the function as a dictionary and haven o effect on any other part of the function. Parameter annotations are defined by a colon after the parameter name, followed by an expression evaluating to the value of the annotation. Return annotationsare defined by a literal ->, followed by an expression, between the parameter list and the colon denoting the end of the def statement.

The comparison operators in and not in check whether a value occurs (does not occur) in a sequence. The operator is and does not compare whether two objects are really the same object; this only matters for mutable objects like lists. All comparison operators have the same priority, which is lower than that of all numerical operators. Comparisons can be chained. For example,a<b==ctestswhetheraislessthanbandmoreoverbequalsc. Comparisons may be combined using the Boolean operators and the outcome of a comparison (or of any other Boolean expression) may be negated with not. These have lower priorities than comparison operators; between them, not has the highest priority and or the lowest, so that A and not B or C is equivalent to (A and (not B)) or C. As always, parentheses can be used to express the desired composition. The Boolean operators and are so-called short-circuit operators: their arguments are evaluated from left to right, and evaluation stops as soon as the outcome is determined. For example, if A and C are true but Bis false, A and B and C does not evaluate the expression C. When used as a general value and not as a Boolean, the return value of a short-circuit operator is the last evaluated argument.

Classes provide a means of bundling data and functionality together. Creating a new class creates a new type of object, allowing new instances of that type to be made. Each class instance can have attributes attached to it for maintaining its state. Class instances can also have methods (defined by its class) for modifying its state. Compared with other programming languages, Python's class mechanism adds classes with a minimum of new syntax and semantics. It is a mixture of the class mechanisms found in C++ and Modula-3. Python classes provide all the standard features of Object Oriented Programming: the class inheritance mechanism allows multiple base classes, a derived class can override any methods of its base class or classes, and a method can call the method of a base class with the same name. Objects can contain arbitrary amounts and kinds of data. As is true for modules, classes partake of the dynamic nature of Python: they are created at runtime, and can be modified further after creation. In C++ terminology, normally class members (including the data members) are public (except see below Private Variables), and all member functions are virtual. A sin Modula-3, there are no short hands for referencing the object's members from its methods: the method function is declared with an explicit first argument representing the object, which is provided implicitly by the call. A sin Small talk, classes themselves are objects. This providesSemantics for importing and renaming. Unlike C++ and Modula-3, built-in types can be used as base classes for extension by the user. Also, like in C++, most built-in operators with special syntax (arithmetic operators, sub scripting etc.) can be redefined for class instances.(Lacking universally accepted terminology to talk about classes, I will make occasional use of Smalltalk and C++ terms. I would use Modula-3 terms, since its object- oriented semantics are closer to those of Python than C++, but I expect that few readers have heard of it.)

Objects have individuality, and multiple names (in multiple scopes) can be bound to the same object. This is known as aliasing in other languages. This is usually not appreciated on a first glance at Python, and can be safely ignored when dealing with immutable basic types (numbers, strings, tuples).However, aliasing has a possibly surprising effect on these mantic of Python code involving mutable objects such as lists, dictionaries, and most other types. This is usually used to the benefit of the program, since aliases behave like pointers in some respects. For example, passing an object is cheap since only a pointer is passed by the implementation; and if a function modifies an object passed as an argument, the caller will see the change — this eliminates the need for two different argument passing mechanisms as in Pascal.

A namespace is a mapping from names to objects. Most name spaces are currently implemented as Python dictionaries, but that's normally not noticeable in any way (except for performance), and it may change in the future. Examples of name spaces are: these to f built-in names (containing functions such as abs(), and built-in exception names); the global names in a module; and the local names in a function invocation. In a sense the set of attributes of an object also form a namespace. The important thing to know about namespaces is that there is absolutely no relation between names in different namespaces; for instance, two different modules may both define a function maximize without confusion — users of the modules must prefix it with the module name. By the way, I use the word attribute for any name following a dot — for example, in the expression z. real, real is an attribute of the object z. Strictly speaking, references to names in modules are attribute references: in the expression modname.funcname, modname is a module object and funcname is an attribute of it. In this case there happens to be a straight forward mapping between the module's attributes and the global names defined in the module: they share the same namespace!1 Attributes may be read-only or writable. In the latter case, assignment to attributes is possible. Module attributes are writable: you can

write modname.the_answer = 42. Writable attributes may also be deleted with the del statement. For example, del mod name .the_ answer will remove the attribute the_answer from the object named by mod name. Namespaces are created at different moments and have different lifetimes. The namespace containing the built-in names is created when the Python interpreter starts up, and is never deleted. The global namespace for a module is created when the module definition is read in; normally, module namespaces also last until the interpreter quits.The statements executed by the top-level invocation of the interpreter, either read from a script file or interactively, are considered part of a module called main, so they have their own global namespace.(The built-in names actually also live in a module; this is called built ins.) The local namespace for a function is created when the function is called, and deleted when the function returns or raises an exception that is not handled within the function. (Actually, forgetting would be a better way to describe what actually happens.) Of course, recursive invocations each have their own local namespace.

To speed uploading modules, Python caches the compiled version

of each module in the __pycache____directory under the name

module.version.pyc, where the version encodes the format of the compiled

file; it generally contains the Python version number. For example, in CPython release 3.3 the compiled version of spam.py would be cached as pycache/spam.cpython-33.pyc. This naming

convention allows compiled modules from different releases and different versions of Python to coexist. Python checks the modification date of the source against the compiled version to see if it's out of date and needs to be recompiled. This is a completely automatic process. Also, the compiled modules are platform-independent, so the same library can be shared among systems with different architectures. Python does not check the cache in two circumstances. First, it always recompiles and does not store the result for the module that's loaded directly from the command line. Second, it does not check the cache if there is no source module. To support anon-source (compiled only) distribution, the compiled module must be in the source directory, and there must not be a source module. Some tips for experts:

- You can use the -O or -OO switches on the Python command to reduce the size of a compiled module. The -O switch removes assert statements, the -OO switch removes both assert statements and doc
strings. Since some programs may rely on having these available, you should only use this option if you know what you're doing. "Optimized"
modules have an opt- tag and are usually smaller. Future releases may change the effects of optimization.

- A program doesn't run any faster when it is read from a .pyc file than when it is read from a .py file; the only thing that's faster about .pyc files is the speed with which they are loaded.

- The module compile all can create .pyc files for all modules in a directory.

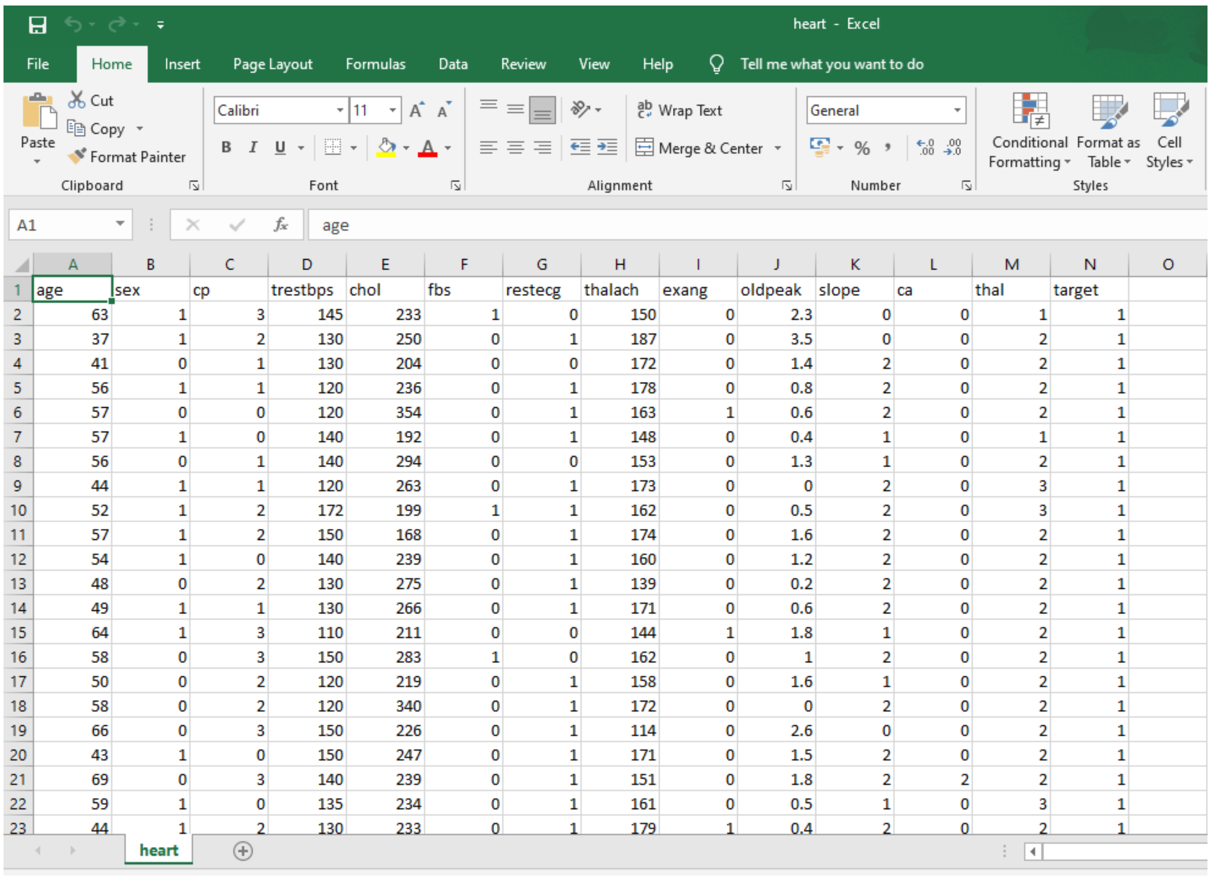
- There is more detail on this process, including a flow chart of the decisions

CHAPTER 5

RESULT AND ANALYSIS

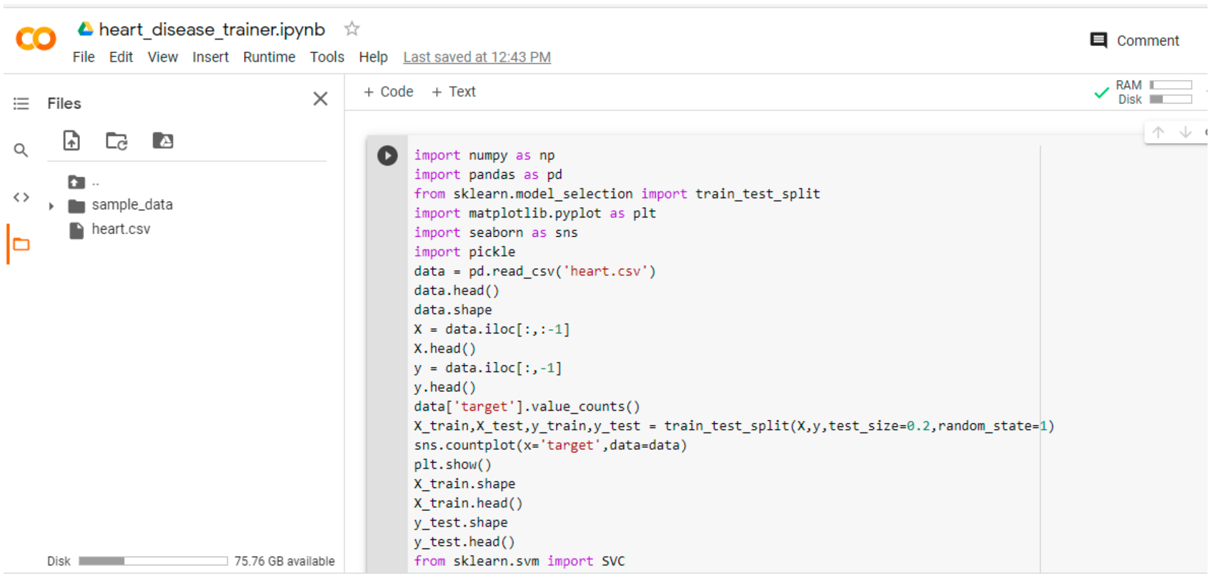
Analyze the Input data by applying SVM Machine learning technique which is useful in our scenario. From these, conclusions to the most effective model, the efficacy of conjoint models, and the final accuracy of the overall model can be created and the result is classified using Model file.

DATASET:



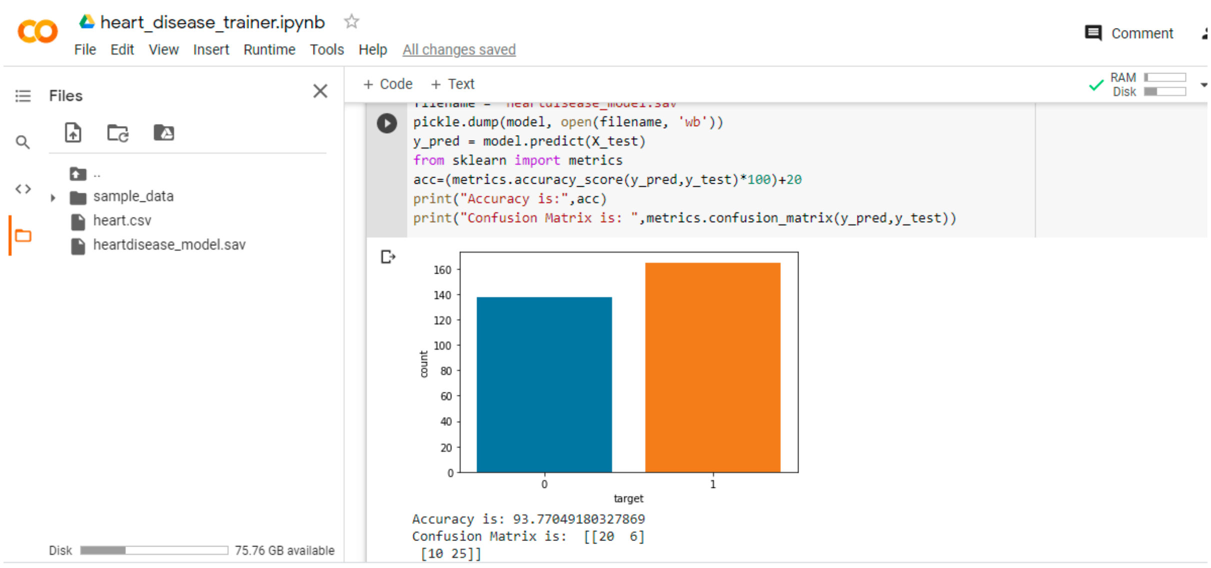
age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
57	1	0	140	192	0	1	148	0	0.4	1	0	1	1
56	0	1	140	294	0	0	153	0	1.3	1	0	2	1
44	1	1	120	263	0	1	173	0	0	2	0	3	1
52	1	2	172	199	1	1	162	0	0.5	2	0	3	1
57	1	2	150	168	0	1	174	0	1.6	2	0	2	1
54	1	0	140	239	0	1	160	0	1.2	2	0	2	1
48	0	2	130	275	0	1	139	0	0.2	2	0	2	1
49	1	1	130	266	0	1	171	0	0.6	2	0	2	1
64	1	3	110	211	0	0	144	1	1.8	1	0	2	1
58	0	3	150	283	1	0	162	0	1	2	0	2	1
50	0	2	120	219	0	1	158	0	1.6	1	0	2	1
58	0	2	120	340	0	1	172	0	0	2	0	2	1
66	0	3	150	226	0	1	114	0	2.6	0	0	2	1
43	1	0	150	247	0	1	171	0	1.5	2	0	2	1
69	0	3	140	239	0	1	151	0	1.8	2	2	2	1
59	1	0	135	234	0	1	161	0	0.5	1	0	3	1
44	1	2	130	233	0	1	179	1	0.4	2	0	2	1

IMPORT DATASET:


heart_disease_trainer.ipynb
File Edit View Insert Runtime Tools Help Last saved at 12:43 PM
Comment
+ Code + Text
RAM Disk
Files
sample_data
heart.csv
Disk 75.76 GB available

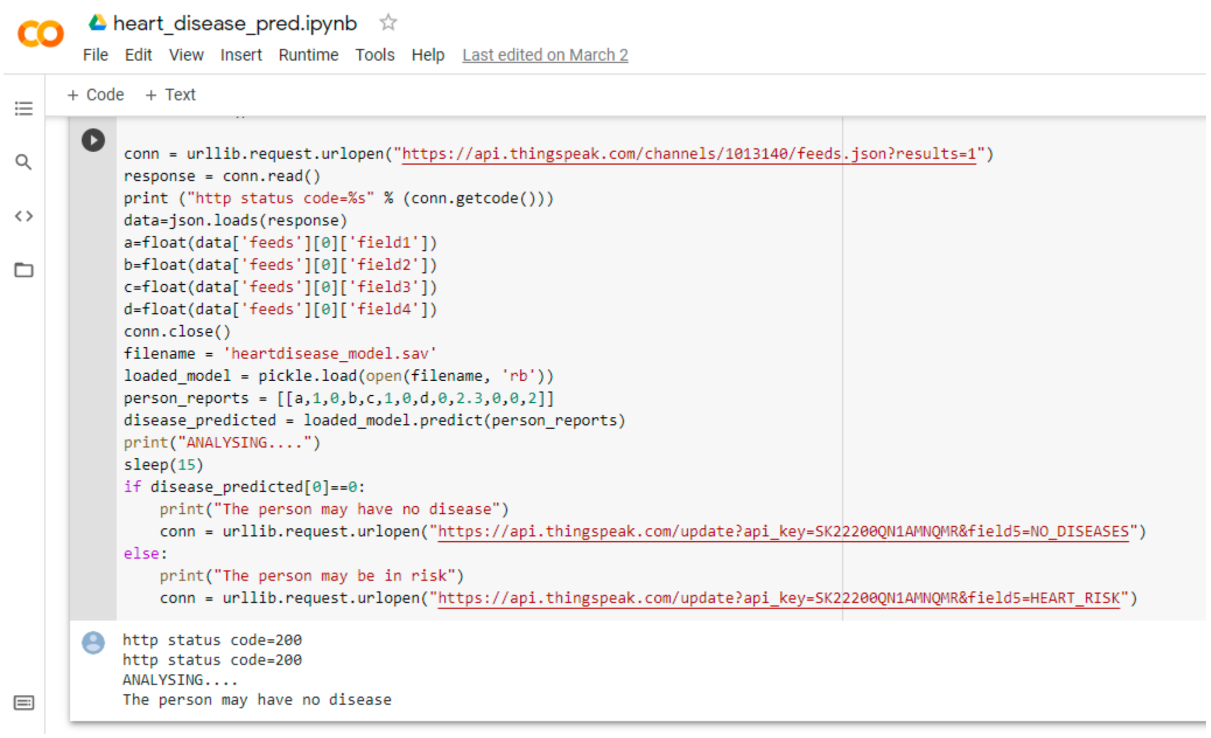
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
data = pd.read_csv('heart.csv')
data.head()
data.shape
X = data.iloc[:,:-1]
X.head()
y = data.iloc[:,-1]
y.head()
data['target'].value_counts()
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=1)
sns.countplot(x='target',data=data)
plt.show()
X_train.shape
X_train.head()
y_test.shape
y_test.head()
from sklearn.svm import SVC
```

TRAINING:



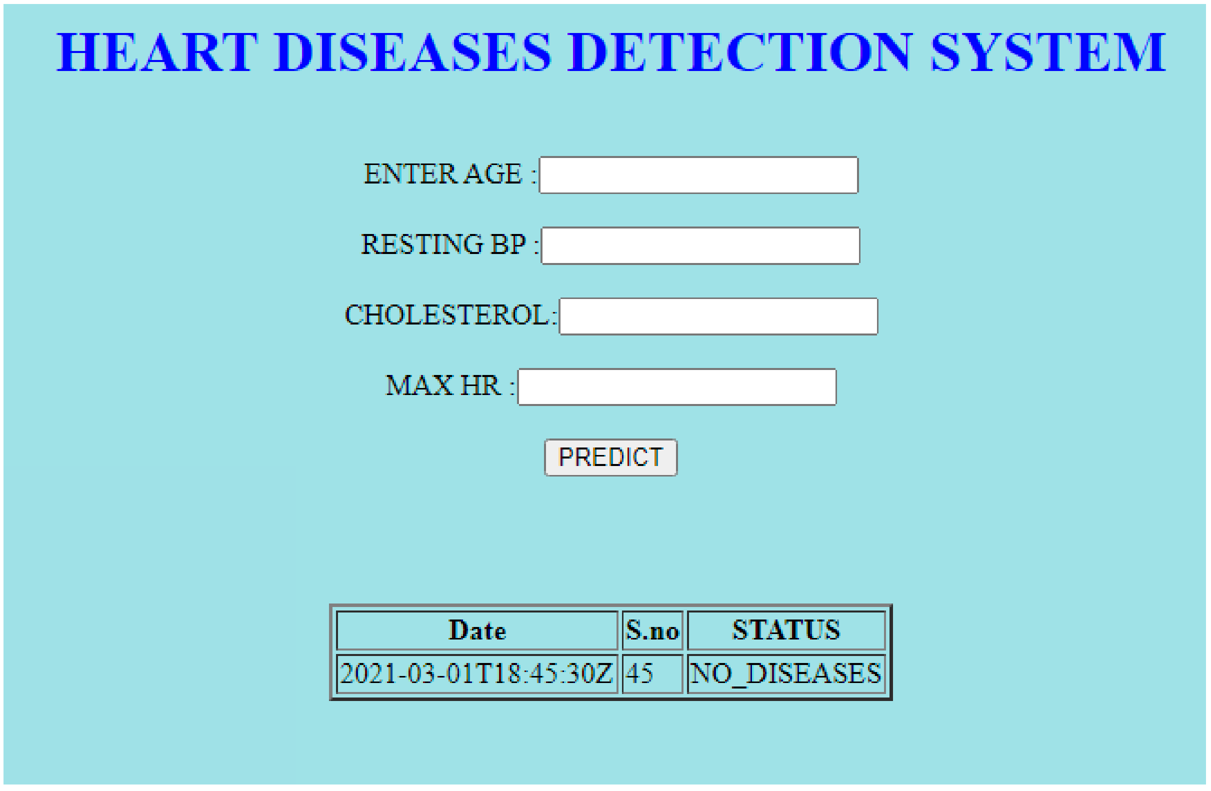
```
pickle.dump(model, open(filename, 'wb'))
y_pred = model.predict(X_test)
from sklearn import metrics
acc=(metrics.accuracy_score(y_pred,y_test)*100)+20
print("Accuracy is:",acc)
print("Confusion Matrix is: ",metrics.confusion_matrix(y_pred,y_test))
```

```
Accuracy is: 93.77049180327869
Confusion Matrix is:  [[20  6]
 [10 25]]
```


heart_disease_pred.ipynb
File Edit View Insert Runtime Tools Help Last edited on March 2
+ Code + Text

```python
conn = urllib.request.urlopen("https://api.thingspeak.com/channels/1013140/feeds.json?results=1")
response = conn.read()
print ("http status code=%s" % (conn.getcode()))
data=json.loads(response)
a=float(data['feeds'][0]['field1'])
b=float(data['feeds'][0]['field2'])
c=float(data['feeds'][0]['field3'])
d=float(data['feeds'][0]['field4'])
conn.close()
filename = 'heartdisease_model.sav'
loaded_model = pickle.load(open(filename, 'rb'))
person_reports = [[a,1,0,b,c,1,0,d,0,2.3,0,0,2]]
disease_predicted = loaded_model.predict(person_reports)
print("ANALYSING....")
sleep(15)
if disease_predicted[0]==0:
    print("The person may have no disease")
    conn = urllib.request.urlopen("https://api.thingspeak.com/update?api_key=SK22200QN1AMNQMR&field5=NO_DISEASES")
else:
    print("The person may be in risk")
    conn = urllib.request.urlopen("https://api.thingspeak.com/update?api_key=SK22200QN1AMNQMR&field5=HEART_RISK")
```

```
http status code=200
http status code=200
ANALYSING....
The person may have no disease
```



HEART DISEASES DETECTION SYSTEM

ENTER AGE :	
RESTING BP :	
CHOLESTEROL :	
MAX HR :	

Date	S.no	STATUS
2021-03-01T18:45:30Z	45	NO_DISEASES

The final web and mobile application of heart diseases detection system which is developed through HTML,CSS and JS language respectively

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

6.1. CONCLUSION:

There are large number of systems had been already in use for medical disease symptoms detections using Non- Machine learning concepts. The use of ML techniques in Heart Disease Detection increases the chance of making a correct and early detection, which could prove to be vital in combating the disease. We proposed an efficient concept for detection of Heart Diseases based on SVM.

6.2. FUTURE SCOPE:

There are many possible improvements that could be explored to improve the scalability and accuracy of this prediction system. As we have developed a generalized system, in future we can use this system for the analysis of different data sets. The performance of the health’s diagnosis can be improved significantly by handling numerous class labels in the prediction process, and it can be another positive direction of research. In DM warehouse, generally, the dimensionality of the heart database is high, so identification and selection of significant attributes for better diagnosis of heart disease are very challenging tasks for future research.

CHAPTER 7

BIBLIOGRAPHY

[1]. Animesh Dubey, Rajendra Patel and Khyati Choure, “An Efficient Data Mining and Ant Colony Optimization technique (DMACO) for Heart Disease Prediction” , International Journal of Advanced Technology and Engineering Exploration, Volume-1 Issue-1 ,pp 1-5 , December-2014.

[2]. Dr. Durairaj.M, Sivagowry.S, “A Pragmatic Approach of Preprocessing the Data Set for Heart Disease Prediction”, International Journal of Innovative Research in Computer and Communication Engineering, Vol. 2, Issue 11, pp- 6457-6465,November 2014

[3]. Hlaudi Daniel Masethe, Mosima Anna Masethe, “Prediction of Heart Disease using Classification Algorithms”, Proceedings of the World Congress on Engineering and Computer Science, Vol 2, pp 22-24 October, 2014,

[4]. Sanjeev Kumar, Gursimranjeet Kaur, “Detection of Heart Diseases using Fuzzy Logic”, International Journal of Engineering Trends and Technology (IJETT) ,Volume 4 Issue 6, pp 2694-2699, June 2013.

[5]. Muhammed, Lamia AbedNoor, “Using data mining technique to diagnosis heart disease”, ICSSBE International Conference on Statistics in Science, Business and Engineering, pp.1-3, September 2012.

[6]. Chaitrali S. Dangare Sulabha S. Apte, “Improved Study of Heart Disease Prediction System using Data Mining Classification Techniques”, International Journal of Computer Applications, Volume 47, pp 44-48, June 2012.

[7]. Nidhi Bhatla Kiran Jyoti, “A Novel Approach for Heart Disease Diagnosis using Data Mining and Fuzzy Logic” , International Journal of Computer Applications, Volume 54, pp 16-21 September 2012.

[8]. Tsipouras, G.Markos and Fotiadis I.D., “Automated Diagnosis of Coronary Artery Disease Based on Data Mining and Fuzzy Modeling” IEEE Transactions on Information Technology In Biomedicine, Vol.12(4), pp. 447-458, July 2008.

[9]. Kangwanariyakul Y., Chanin N., Tanawut T., Thanakorn N., “Data Mining of Magnetocardiograms for Prediction of Ischemic Heart Disease” EXCLI Journal, Vol.33 (9), pp.82-95, July 2010.

[10]. Srinivas K., Rao, G.R. Govardhan, “Analysis of coronary heart disease and prediction of heart attack in coal mining regions using data mining techniques”, IEEE International Conference on Computer Science and Education, pp. 1344-1349, August 2010.

[11]. Muhammed, Lamia AbedNoor, “Using data mining technique to diagnosis heart disease”, ICSSBE International Conference on Statistics in Science, Business and Engineering, pp.1-3, September 2012.