B. TECH. PROJECT REPORT

On

Machine Learning Under Constrained Environments

BY

ADITYA PANDEY

ASHISH KUMAR GOUR

RAJA SHREE PRABHU

DISCIPLINE OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

NOVEMBER 2019

Machine Learning Under Constrained Environments

A PROJECT REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE AWARD OF THE DEGREES

OF

BACHELOR OF TECHNOLOGY

IN

COMPUTER AND SCIENCE ENGINEERING

SUBMITTED BY:

ADITYA PANDEY (160003004)

ASHISH KUMAR GOUR (160002008)

RAJA SHREE PRABHU (160003042)

GUIDED BY:

Dr. I.A. Palani
Associate Professor
Mechanical Engineering

Dr. Abhishek Srivastava
Associate Professor
Computer Science and Engineering

INDIAN INSTITUTE OF TECHNOLOGY INDORE

NOVEMBER 2019

CANDIDATE'S DECLARATION

We hereby declare that the project entitled "Machine Learning under Constrained Environments" submitted in partial fulfillment for the award of the degree of Bachelor of Technology in Computer Science and Engineering completed under the supervision of Dr. I.A. Palani, Associate Professor, Mechanical Engineering and Dr. Abhishek Srivastava, Associate Professor, Computer Science and Engineering, IIT Indore is an authentic work.

Further, we declare that we have not submitted this work for the award of any other degree elsewhere.

Aditya Pandey (160003004)
Ashish Kumar Gour (160002008)
Raja Shree Prabhu (160003042)

CERTIFICATE by BTP Guide(s)

It is certified that the above statement made by the students is correct to the best of my/our knowledge.

Dr. I.A. Palani
Associate Professor
Mechanical Engineering
IIT Indore

Dr. Abhishek Srivastava
Associate Professor
Computer Science and Engineering
IIT Indore

PREFACE

This report on "Machine Learning Under Constrained Environment" is prepared under the guidance of Dr. I.A. Palani and Dr. Abhishek Srivastava. Through this report, we propose a novel approach that addresses the problem of real-time and accurate prediction on resource-scarce devices along with development of a self-energized part. In algorithm side, we did two stage compression, using a probability difference approach and K-Means Clustering followed by shifting of clusters and the machine learning model is implemented using Python. Our approach provides several orders lower storage and prediction complexity. For the self-energized part we developed a vertical axis wind turbine. One of the practical applications of the presented work is forest fire prediction, Client-Server data transfer using Arduino IDE is also established.

We have tried to the best of our abilities and knowledge to explain the content in a lucid manner. We have also added and figures and working code(s) to make it more illustrative.

Aditya Pandey (160003004)
Ashish Kumar Gour (160002008)
Raja Shree Prabhu (160003042)

Acknowledgements

We would like to thank my B-Tech project supervisors **Dr. I. A. Palani** and **Dr. Abhishek Srivastava** for their guidance and constant support in structuring the project and their valuable feedback throughout the course of this project. Their overseeing the project meant there was a lot that we learnt while working on it. We thank them for their time and efforts.

We are grateful to **Mr. Arun Kumar** without whom this project would have been impossible. He provided valuable guidance with the Mathematics involved in the project.

We am really grateful to the Institute for the opportunity to be exposed to systemic research especially **Dr. I. A. Palani** and **Dr. Abhishek Srivastava's Lab** for providing the necessary hardware utilities to complete the project. Lastly, we offer my sincere thanks to everyone who helped me complete this project, whose name we might have forgotten to mention.

Aditya Pandey (160003004)
Ashish Kumar Gour (160002008)
Raja Shree Prabhu (160003042)

Abstract

Several real-world applications require real-time prediction on resource-scarce devices such as an Internet of Things (IOT) sensor. Such applications demand prediction models with small storage and computational complexity that do not compromise significantly on accuracy. Our approach is inspired by k-Nearest Neighbor (k-NN) but has several orders lower storage and prediction complexity. A probability difference based approach is presented, to eliminate non-representative points from the dataset. Using k-Means Clustering, a new set of prototypes are formed for the nearest neighbor classifier. These prototype locations are optimized through an iterative shifting process. With these steps, we get a modified training set with the minimum and optimal number of prototypes. Experiments show the efficiency of this approach on various datasets. For the self-energized part, we developed and tested several models. Models were prepared using additive manufacturing.

CONTENTS

LIST OF FIGURES

List of Abbreviations

SVM	**S**upport **V**ector **M**achine
KNN	**K-N**earest **N**eighbor
PCA	**P**rinciple **C**omponent **A**nalysis
TCP	**T**ransmission **C**ontrol **P**rotocol
UDP	**U**ser **D**atagram **P**rotocol
DNN	**D**eep **N**eural **N**etwork
LMNN	**L**arge **M**argin **N**earest **N**eighbors
SNC	**S**tochastic **N**eighborhood **C**ompression
PDM	**P**robability **D**ifference **M**ethod
BNC	**B**inary **N**eighborhood **C**ompression
DSNC	**D**eep **S**tochastic **N**eighborhood **C**ompression
HAWT	**H**orizontal **A**xis **W**ind **T**urbine
VAWT	**V**ertical **A**xis **W**ind **T**urbine
PLA	**P**olylactide

CHAPTER 1
INTRODUCTION

1.1 Internet-of-things

I.o.T. is a technology which has the potential to provide fast real time, sensor-based responses for a variety of applications like wildlife, image-sensing, unmanned missions and so on. Using Artificial Intelligence on data collected from I.o.T. sensors opens up an ocean full of possibilities. For example, pollution control uses smart sensors which measure ppm, smoke, humidity and various other parameters of their machines. Anomaly detection models in airplanes collect vibration rate, rpm, heat etc. from sensors implanted which assist pilots in the flying.

Forest fire prediction is one of the major applications of using machine learning algorithms in constraint environment. By predicting forest fire, we can control the fire before it spreads in a larger area, by which we can reduce the loss of life and environmental resources. To detect the forest fire, we capture four main components of the environment using IoT sensors. We use Humidity, Temperature, Smoke, IR sensitivity to predict the forest fire. We have several Arduino UNO devices (with 2kB RAM) installed at several close locations in the forest to predict the forest fire. We can perform the real-time prediction on one of these devices. For this, we need to establish communication among these devices, which can be done using Wi-Fi modules.

1.2 Existing models and k-NN

Existing machine learning in IoT rely on cloud-based predictions where large deep learning models are deployed to provide accurate predictions. Models like SVM and Deep Neural Networks requires storage of thousands of parameters even at the time of prediction. This is because embedded devices like Arduino have limited a processor and storage abilities and are only meant to transmitting data to the cloud.

This solution doesn't concerns about issues like bandwidth, latency, privacy, battery and other issues. For example, consider a device being implanted in a human brain. This device continuously sends how good/bad a tumor is progressing. If we have poor latency over sending to the cloud in the network, the responses may be delayed and wrong decisions will be taken. This can be very serious in such a scenario.

We use k-NN type or distance-metric algorithm for prediction. K-NN works by assigning an unlabeled input to the majority label of its k nearest training inputs. K-NN is very easy to implement on tiny devices, and has a very small number of parameters, therefore its avoid overfitting and provides a good generalization over the results.

1.3 Rescuing k-NN

A critical issue with k-NN is its slow test-time performance. It has to compute the distances between the test input point and all elements in the training set, it's time complexity becomes **O(nd)** with respect to the data dimensionality d and the training set size n. Similarly, storage complexity is also **O(nd)**, as the entire training set needs to be stored. What if we could simply choose some representative data points, termed as "prototypes", out of the training data, train only on this data, and make predictions locally on our device, with transferring data to cloud. The question is, how we choose these points.

We can use a probability difference based approach to eliminate non-representative points, ie. the points which are not near the boundaries between the classes. If we think of SVM classifier, we are eliminating nearly all points which are not hyper vectors. After this, we choose some points thorough K-Means clustering as our prototypes. These prototype locations are optimized through an iterative shifting process. With these steps, we get a modified training set with the minimum and optimal number of prototypes.

CHAPTER 2
LITERATURE REVIEW

There are various research papers already published on compression algorithm which are used to compress the data points but each of them suffer from their individual drawbacks and limitations.

2.1 Background

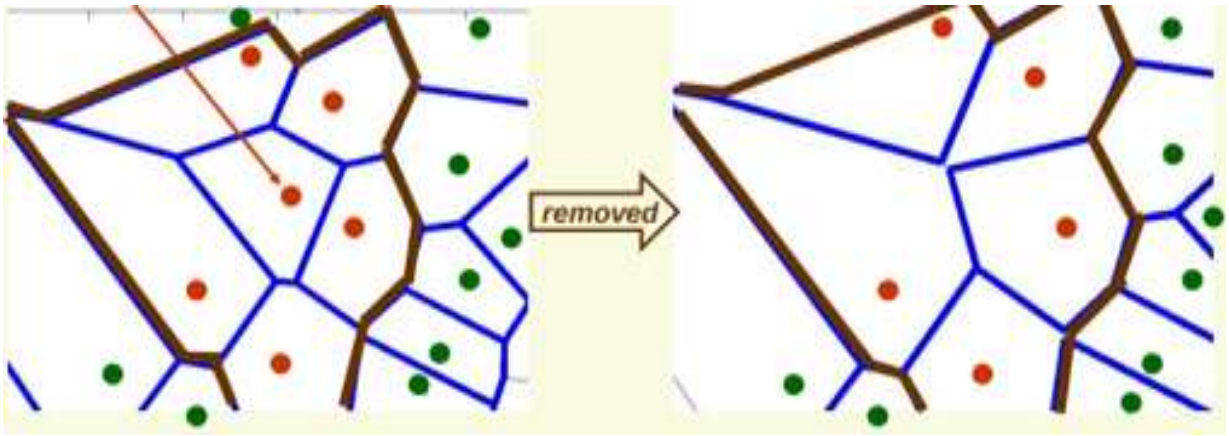


Fig 2.1 Effect of removing inner points

The Voronoi diagram of a set of objects decomposes space into Voronoi cells, where each object's cell consists of all points that are closer to the object than to any other object. The line segments of any Voronoi cell are perpendicular bisectors to the line joining two neighboring points. As seen from the figure above, if we try to remove a point which has all immediate neighbors of the same class, the outer Voronoi boundary *(shown in brown)* doesn't change. Removal of such points forms the initial basis for our compression.

Previous works in this field include using tree-like approaches as described above. But these approaches are very computationally expensive, especially in larger datasets. Though they find the correct points to eliminate, they don't guarantee the minimal set of points found.

2.2 Compression Algorithms

There are three approaches for compression:

2.2.1 Tree data structures

Firstly, we will use tree data structures like cover/ball trees which decrease the count of computation to measure distance to some logarithmic function in n.

In Binary space partitioning, we will recursively divide a space into two until the partitioning fulfills the one or more requirements. It will act as a generalization of other spatial tree structures like **k-d trees and quad trees**, in which the hyperplanes that partition the space maybe having any orientation, rather than merely being aligned with the coordinate axes as they are in *k*-d trees or quad trees.

In a *k*-d tree, every non-leaf node can be thought of as a hyperplane that divides the space into two half-spaces. Every node in the tree is associated with one of the *k* dimensions. We choose the normal vector of our hyperplane in the direction perpendicular to that dimension's axis. Points to the left of this hyperplane are represented by the left subtree of that node and points to the right of the hyperplane are represented by the right subtree. *Every* leaf node is a *k*-dimensional point is a binary tree.

Even though this process is faster, it still has to store the entire training data. This would cause the performance to deteriorate with increase in the data dimension.

2.2.2 Large Margin nearest Neighbors (LMNN)

Another class of methods improve accuracy of k-NN by learning a better metric to compare, given a pair of points. A Large Margin Nearest Neighbor (LMNN) classifier which transforms the input space such that in the transformed space points from same class are closer compared to points from disparate classes. LMNN's transformation matrix can map data into lower dimensions and reduce overall model size compared to k-NN, but it is still too large for most resource-scarce devices.

2.2.3 SNC and its modifications

SNC reduces the training set by reducing the number of data inputs n. This involves data set condensing (or thinning). SNC learns new set of prototypes in a way that their likelihood of a particular class probability model is maximized. Thus, SNC works only for multi-class problems but not, for multilabel/multiranking problems. SNC uses LMNN for low-d projection of the data using the LMNN based projection matrix and then learns prototypes in the projected space to decrease the model size.

DSNC uses a feedforward network to learn the non-linear transformation. Therefore, its model size can be significantly larger than SNC. It basically learns a non-linear low-d transformation jointly with the prototypes.

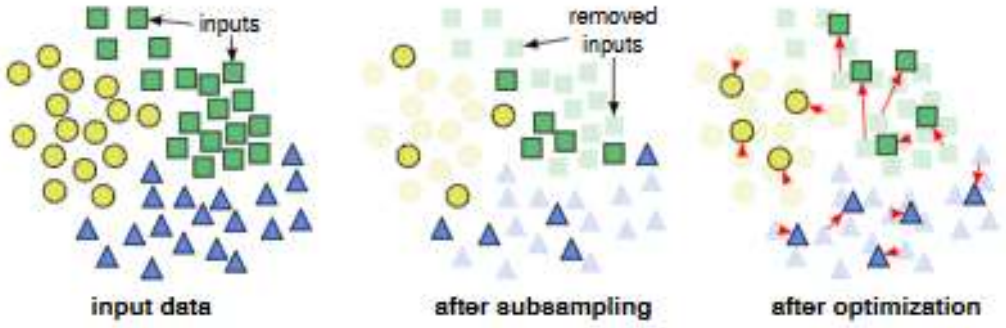


Fig 2.2 Our work shown on 3 classes

CHAPTER 3
OUR APPROACH

3.1 The probability of belongingness

Let's study our approach on two points. The **Softmax probabilities** for a particular point X belonging to a class 'j' is given by:

$$P(y = j|X) = \frac{e^{X^T W_j}}{\sum_{k=1}^{K} e^{X^T W_k}}$$

Let's say probabilities of belongingness of these 2 points to the 2 classes, come out to be:

- P1 = [0.5, 0.5] (Relevant) P2 = [0.8, 0.2] (Redundant)

The "Relevant" points

The inner points have a higher probability of belonging to the class it belongs compared to a point which is much closer to the margin.

So relevance order of points will be like: [0.5, 0.5] > [0.6, 0.4] > [0.8, 0.2] and so on. But this ordering isn't always true for >=2 classes. Why?

Consider for example: P1 = [0.4, 0.3, 0.2, 0.1] and P2 = [0.45, 0.45, 0.05, 0.05]. Obviously, P2 > P1 for priority ordering because being more on boundary than P1.

3.2 Probability Difference Approach

Then, we use the difference = [highest probability value - second highest probability], and append those points with this "**minimum difference value**". This finally forms our basis of ordering. At P2, [0.45 - 0.45 = 0] and at P1, [0.4 - 0.3 = 0.1], hence P2 will be chosen over P1.

At every iteration, we append a batch of some points in an empty set based on this ordering. We train our model using the same classifier on the points we appended till now, we find the current prediction score on all given training points, not only on the points till now. Let the predicted score at this iteration be 'P'. We keep on doing the iterations till: $(T - P \leq \varepsilon)$, where ε is a small number equal to 0.005.

3.3 Which classifier to choose for initial compression?

We use SVM as our classifier for probability difference method.

Optimization Problem that SVM solves:

$$\text{Minimize in } (\mathbf{w}, b)$$

$$\|\mathbf{w}\|$$

$$\text{subject to } y_i(\mathbf{w} \cdot \mathbf{x_i} + b) \geq 1$$

$$\text{(for any } i = 1, \ldots, n)$$

Here ||w|| is the norm of the vector perpendicular to the two hyperplanes.

The margin of SVM, is given by:

$$m = \frac{2}{\|\mathbf{w}\|}$$

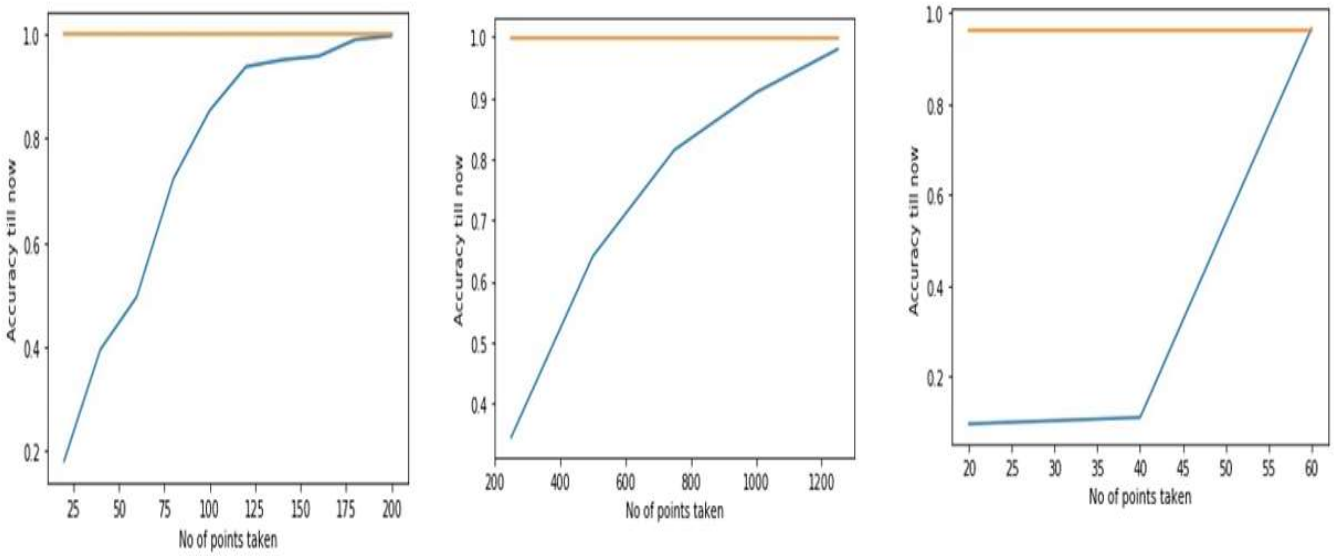
Therefore maximizing the margin is same as minimizing the value of ||w||. A hyperplane providing **maximum margin** between two classes would help our probability values to even out well. This

motivates us to use SVM as our classifier in hand. Also, using SVM, the probability model is created using cross validation, we don't have to use train_test_split explicitly.

3.4 Initial Results

Dataset	Category	No. of training examples	No. of features	No. of classes	Relevant points	Compression Ratio	Accuracy degradation
Breast Cancer	Small-Medium	569	30	2	40	569/60 = 14.22	0.72%
MNIST	Medium	1797	64	10	200	1797/200 = 9	0.8%
ISOLET	Large	6238	10	26	1000	6238/1000 = 6.23	3.9%

Plots of three datasets:



MNIST **ISOLET** **BREAST CANCER**

Fig 3.1 Plots showing convergence of accuracy as we keep on adding points

CHAPTER 4
FURTHER COMPRESSION AND SHIFTING

On an average, we currently have about 6-15 times average compression on our dataset.

4.1 Using k-Means Clustering

We use k-Means clustering on our prototypes of each class individually in our selected ones as well as the remaining excluded. The loss function at each iteration step is given by:

$$J(V) = \sum_{i=1}^{c} \sum_{j=1}^{c_i} \left(\left\| x_i - v_j \right\| \right)^2$$

Where,

'$||x_i - v_j||$' is the Euclidean distance between x_i and v_j.
'c_i' is the number of data points in i^{th} cluster.
'c' is the number of cluster centers.

After the loss function is decreased at each iteration, the new cluster center is given by:

$$v_i = (1 / c_i) \sum_{j=1}^{c_i} x_i$$

Where, 'c_i' represents the number of data points in i^{th} cluster.

This further reduces our dataset. The final reduction in our dataset becomes 30-50 times the original dataset.

We now find out the accuracy by fitting 1-kNN on these clusters and predict the labels for complete dataset. Our k-NN will use the Minkowski distance metric to fit the clusters.

Minkowski distance metric with p=2 (Euclidean):

$$\left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}$$

We see that our accuracy got reduced. We can restore our accuracy by shifting our prototypes.

4.2 Why is there a degradation in Accuracy?

$$\tilde{P}(y|X) = \frac{\sum_{X' \in N_K(X)} w(\Delta(X, X'))\delta(y, y')}{\sum_{X' \in N_K(X)} w(\Delta(X, X'))}$$

Till now, we applied k-Means and got our final cluster centers as prototypes. Note that we applied k-Means individually on each class. But now take a closer look at what's happening at the boundary now. Suppose we have a slightly curvy boundary. Let the class left to boundary be class A and that to the right of it be class B. Now, let's look at what's happening for a point which is just to the left of this curvy (convex-facing to the left).

If we take k nearest neighbors of our point in hand, we see that we get for a cluster centers left, we get δ(y, y') = 1, since it's of the same class. By the same argument, for a cluster centers in the right, we get δ(y, y') = 0, since it's of the opposite class. Therefore, the probability value is high for same class point, but low for opposite class.

But for this point (which is on a convex boundary), it should have been the other way around. Hence, there's a misclassification, or in other sense, a degradation in accuracy.

Typically, we have used 2 to 5 clusters in k-Means clustering for every class. This number depends on the dataset size and has to be tuned in accordance to the accuracy results.

Restoring our accuracy back

If we try to shift every prototype (k-Means clusters) in the direction which yields highest accuracy, then we are done. But how to do that optimally and without interfering the shifting of other prototypes?

If we look at every feature value of a particular prototype, we notice that it's independent of any other feature values of that prototype. So shifting features individually won't interfere the shifts of each other, hence shifting features one-by-one shifts the whole prototype.

How far should we shift each feature value?

- We take the standard deviation for all the features across all training points. Then, we divide that value by a number, say 50. This number typically may be even smaller, and it depends on how large/small is the standard deviation. If it's larger, our number should be larger and vice versa.
- Then, we take some points, say 10, on either side of our feature value point at distances being multiples of the value we obtained. We try to see, at which of these 20 points, do we get highest accuracy after shifting our original feature to this point.

We take only a few **neighboring points** on each side because we want to shift a point only when it's possible but in a close proximity. This way, we avoid overlaps and have a better representation of our data.

4.3 Why use standard deviation?

Standard Deviation gives an idea about how much spreadness our dataset has.

1. We have used standard deviation as our unit distance for how much to shift our feature value.
2. Suppose we have a dataset which has min_val = 0 and max_val = 100000, then this (max_val - min_val) will be a very large number. Seeing which point to choose will take a lot of time.
3. Instead, if we have an idea of the spread in that feature direction, we can get that number quantitatively as 'X' and choose our shifts as X,2X,3X… and so on till 10X in either direction.
4. It works for any kind of data spread, large or small std_dev.

Avoiding overlap of final prototypes

We achieved this by 2 ways:

1. Firstly, we try to shift a particular feature value only to certain number of steps to the right and the left (scaled by their std_dev). This avoids overlapping of multiple prototypes.

2. Secondly, we apply shifting individually to each feature value. Features are independent of each other and their shifting will cause the whole prototype to shift constructively.

4.4 Shifting Pseudo-Code:

Overall, the algorithm is summarized as a pseudo-code below:

For every prototype:

For every feature of that prototype:

1. **For** every 10 neighboring points (scaled to the standard deviation) on either sides of our feature point:

Find the point on feature line with the highest score/accuracy based on k-NN.

2. Shift it to that point.

Time complexity of the shifting process is very low because the number of prototypes have been reduced significantly for the first for loop.

4.5 Final outline of our Approach:

1. On training data, fit SVM classifier, find true accuracy and probabilities.
2. Apply probability difference method (PDM).
3. Fit SVM again, on points obtained from above method, predict accuracies on complete training data, <=1% less than true accuracy.
4. We keep those points we obtain from the PDM in the set “New” and the remaining points in points in the set “Rem”.
5. Now fit k-Means clustering individually on each class and individually on both New and Rem.
6. Obtain the cluster centers as prototypes.
7. Fit k-NN on these clusters and find accuracy of classification on or complete dataset.
8. In the above point, by closely inspecting the probability formula, the accuracy decreases, in other words, these prototypes are not able to represent our dataset properly, they must be shifted.
9. Apply shifting approach.
10. These new prototypes are now our final data points. Whenever we get a new data point, just look for its closest ‘k’ points (k = 1/2/3), and assign its class to the majority vote.

4.6 Working Code

```
# Number of Classes
noc = 2

# Loading the Dataset-----------------------------------------------------------

from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
labels = data['target']
features = data['data']

# Standarsization/ Preproceessing-----------------------------------------------

from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
sc_y = StandardScaler()
features = sc_X.fit_transform(features)

# Initial Training--------------------------------------------------------------

clf = SVC(gamma='auto',probability=True)
clf.fit(features,labels)

# Probability Difference based Ordering-----------------------------------------------------------------------------

arr = clf.predict_proba(features)
true_score = clf.score(features, labels)
print("true_score: ", true_score,"\n")

arr_sorted = []
mini = 1
maxi = 0

for i in range(len(arr)):
    arr[i] = np.sort(arr[i])
    arr[i][:] = arr[i][::-1]
    t = abs(arr[i][0]-arr[i][1])
    mini = min(mini,t)
    maxi = max(maxi,t)
    arr_sorted.append((t,i))
arr_sorted.sort(key = lambda x: x[0], reverse=True)

# Appending---------------------------------------------------------------------

new_dfx = []
new_dfy = []
rem_dfx = []
rem_dfy = []
plot_x = []
plot_y = []
plot_y_true = []
```

```
siz = len(arr_sorted)
z = 0
score = 0

for i in range(0,siz,20):
    for j in range(i,i+20):
        new_dfx.append(features[arr_sorted[siz-1-j][1]])
        new_dfy.append(labels[arr_sorted[siz-1-j][1]])

    clf = SVC(gamma='auto',probability=True)
    clf.fit(features,labels)
    z = z+20;

    # Putting the leftover points in another set before breaking----------------------------------------------
    if(abs(true_score-score) < 0.01):
        i += 20
        while(i < siz):
            rem_dfx.append(features[arr_sorted[siz-1-i][1]])
            rem_dfy.append(labels[arr_sorted[siz-1-i][1]])
            i += 1
        break

# Compression after Appending-----------------------------------------------------------------

print(score,true_score,abs(true_score-score))
print(z,features.shape[0])
print("Compression ratio: ", features.shape[0]/z)
print("Accuracy change: ", abs(true_score-score)*100, "%\n")

# Clustering----------------------------------------------------------------

clus = 4
classes = []
classes_rem = []

for i in range(noc):
    classes.append([])
    classes_rem.append([])

for i in range(len(new_dfy)):
    classes[new_dfy[i]].append(new_dfx[i])

for i in range(len(rem_dfy)):
    classes_rem[rem_dfy[i]].append(rem_dfx[i])

Uf = []
claar = []
temps = []
ctr = 0
```

```
for i in range(noc):
    vara = min(int(clus),len(classes[i]))
    if(vara):
        Kmean = KMeans(n_clusters = vara)
        Kmean.fit(classes[i])
        U1 = list(Kmean.cluster_centers_)
        Uf.extend(U1)
        for j in range(vara):
            temps.append(i)
        ctr += vara

    # For remaining clusters-----------------------------------------------------------------------
    vara = min(int(clus),len(classes_rem[i]))

    if(vara):
        Kmean = KMeans(n_clusters = vara)
        Kmean.fit(classes_rem[i])
        U1 = list(Kmean.cluster_centers_)
        Uf.extend(U1)
        for j in range(vara):
            temps.append(i)
        ctr += vara

# Fitting kNN before shifing-------------------------------------------------------------------------
knn = KNeighborsClassifier(n_neighbors = 1)
knn.fit(Uf,temps)
bf_sh = knn.score(features,labels)

# Shifting logic-------------------------------------------------------------------------------------
nof_points = 50
std_dev = np.array(pd.DataFrame(Uf).std())

for i in range(len(Uf)):
    for j in range(len(Uf[i])):
        max_acc = 0
        the_k = 0
        for k in range(9):
            darX = Uf
            darX[i][j] += ((k-4)*std_dev[j]/nof_points)
            knn.fit(darX,temps)
            if(max_acc < knn.score(features,labels)):
                max_acc = knn.score(features,labels)
                the_k = k
        Uf[i][j] += ((the_k-4)*std_dev[j]/nof_points)

knn.fit(Uf,temps)
af_sh = knn.score(features,labels)

print("Accuracy before shifting: ",bf_sh)
print("Accuracy after shifting: ",af_sh)
print("Nof samples used: ",ctr)
print("Compression ratio after Clustering: ", features.shape[0]/ctr)
```

CHAPTER 5

RESULTS

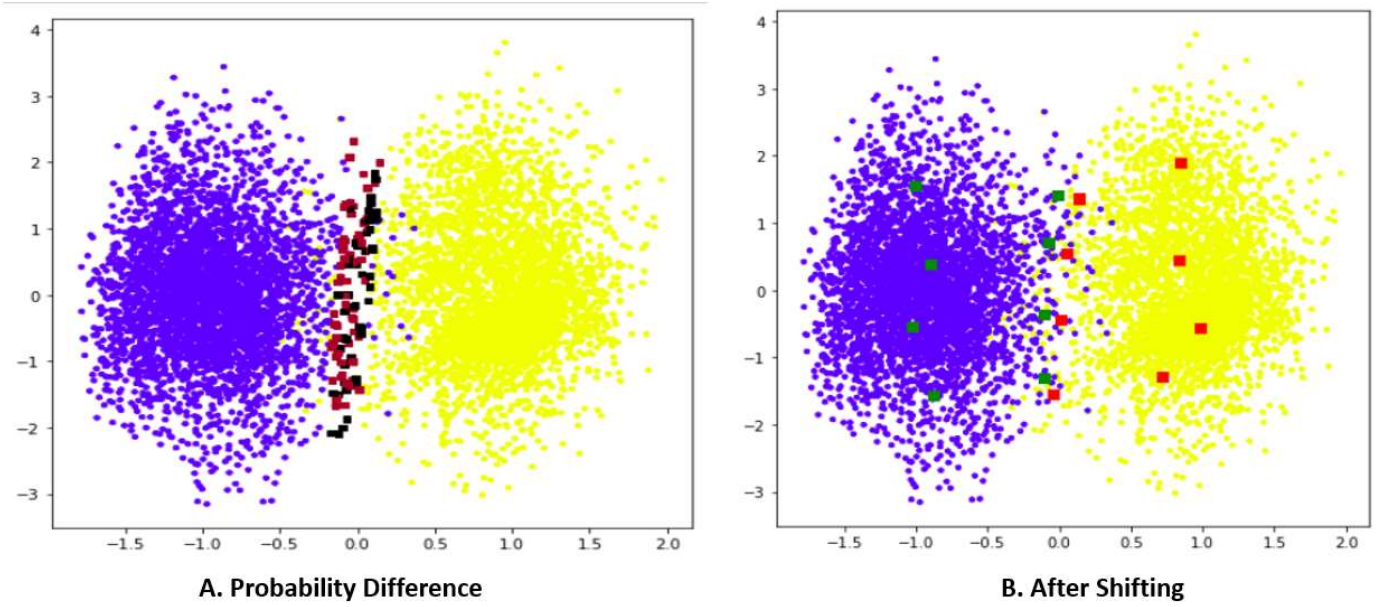


Fig 5.1 Plots showing results after probability difference and after shifting

Dataset	No. of training examples	Final number of Samples used	Compression ratio before clustering	Compression ratio after clustering	Accuracy before shifting	Accuracy after shifting
Breast Cancer	569	16	14.225	35.56	94.1%	95.8%
Wine	178	11	8.5	16.52	97.01%	98.87%
Iris	150	6	4.8	25	90%	96.67%

Chapter 6

Practical Works - Forest Fire Prediction

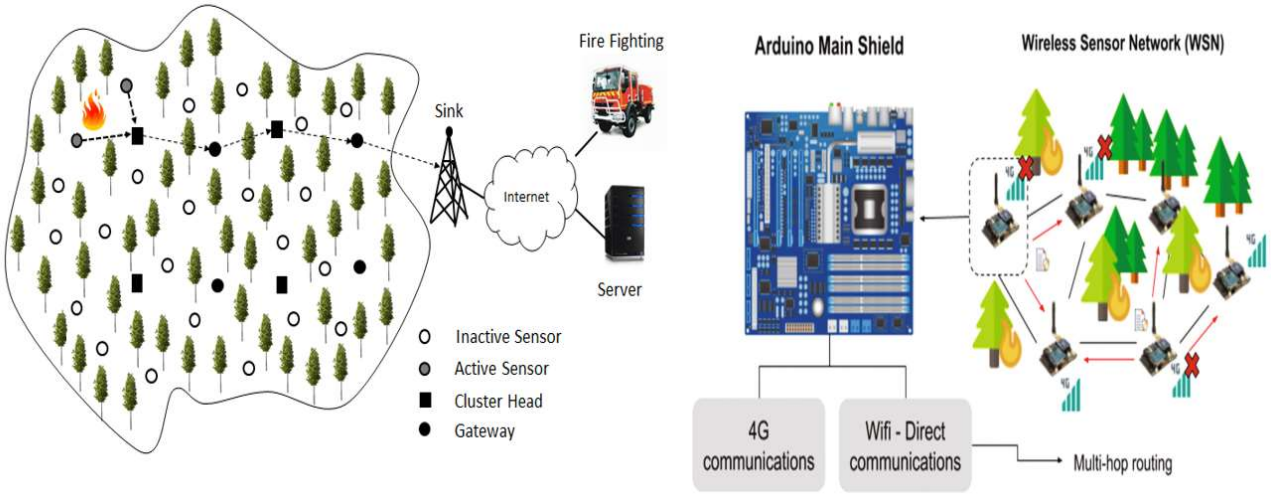


Fig 6.1.1 Outline of the forest fire prediction work

6.1 Objective and Proposed Methodology

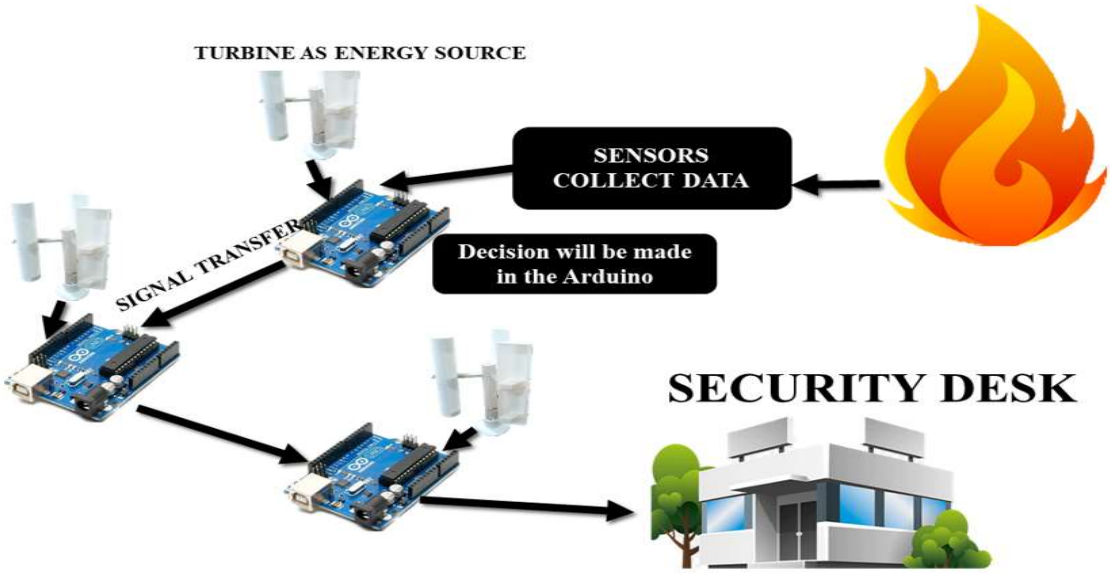


Fig 6.1.2 Objective and proposed methodology

Dataset Collection

We are capturing 4 components of the climate to predict the forest fire i.e. **temperature, humidity, smoke, IR sensitivity.**

1) **Temperature and Humidity sensor**- A capacitive humidity sensor and a thermistor to measure the surrounding temperature.

2) **IR Flame sensor**- This sensor is specifically designed to respond to 4.3μm light emitted by hydrocarbon flames.

3) **Smoke sensor (MQ7)** - This gas sensor detects the concentrations of CO in the air and output its reading as an analog voltage.

6.2 Communication between Arduino boards

We have written the scripts in Arduino IDE to setup a basic Server-Client (2-way) TCP/UDP connection. This connection is established where sensors are placed at a distance of few meters, maximum under the hotspot's range. The prediction on these data points could be done locally on any of these devices. We also wrote scripts to establish 3-way (Client, Client-Server and Server) connection. Here, the middle Arduino device acts as Server during the first part, and then as a Client in the second part.

Fig 6.2 Establishing communication between Arduino sensors

CHAPTER 7

SELF-ENERGIZED UNIT

7.1 SOURCE OF ENERGY IN THE FOREST:

Our fire detection units will be deployed in the forest over a large area. To power these units we use batteries because wired connections can be complex and can be damaged by the animals. Even though the unit itself consumes lower power and can last for a long time, it eventually would run empty. When that happens we would have to change the battery of each and every unit in the forest. This can cause network breakage between the devices and we could even miss out on fire detection. Hence we plan on making an electric generator which utilizes some source of renewable energy from the nature. Solar energy is the first choice of renewable source of energy but in our case it is not ideal because sunlight would be hindered in a dense forest. So we are looking to use wind energy which could be harnessed using a wind turbine.

7.2 WIND TURBINE:

Forest has a lot of different fossil fuels and among which wind is the most available one. As it is renewable resource, it can be used in uncountable amount without affecting the nature. To access this resource wind turbine is used. Wind turbine is basically axis based turbine and basically two types of turbines are available so far, which are vertical and horizontal axis wind turbine. Both of them works well on different situations. A lot of research work has to be done in the wind turbine, one of which is to increase their efficiency.

Parameters like angle of attack, tip-speed ratio, and airfoil shape play major role in increasing efficiency. As in the forest, we can expect in general turbulent wind for fir angle of attack can be properly defined so our main focus will be on to change the airfoil shape.

7.2.1 VERTICAL AXIS WIND TURBINE:

The VAWT is divided again into two groups: Darrius VAWT and Savonius VAWT. The significant difference between them is that Savonius VAWT works on the principle of drag by pushing it to rotate in the direction of the wind flow and the Darrieus VAWT works on the principle of lift which is created due to the varying pressure on either side of the airfoil. The Savonius VAWT rotates at lower speed but has a higher torque and the Darrieus VAWT has lower torque but rotates at a higher speed than the speed of the wind due to lift. The lower torque is beneficial since it doesn't induce as much as stress so, it can have a longer life. Savonius VAWT can self-start easily but this cannot be said for the darrieus VAWT which requires a starting torque to rotate. All of this indicate that the Darrieus VAWT is superior to the Savonius VAWT except the self-starting problem for which we will try to find a solution for later.

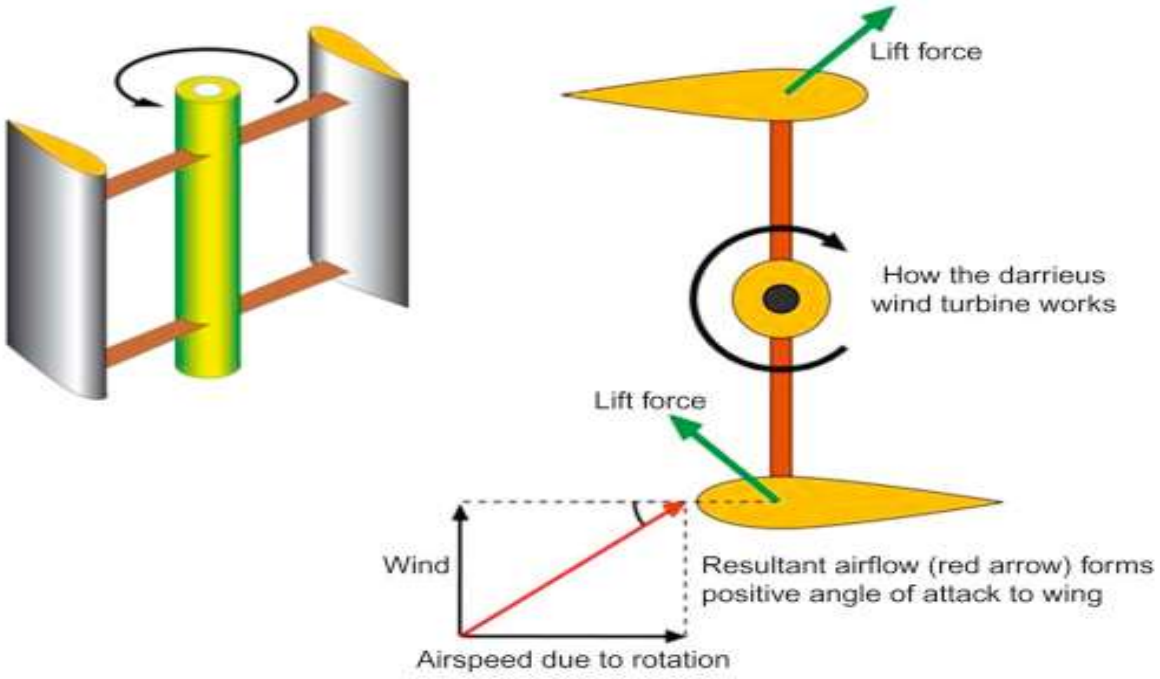


Fig 7.1 Working of VAWT

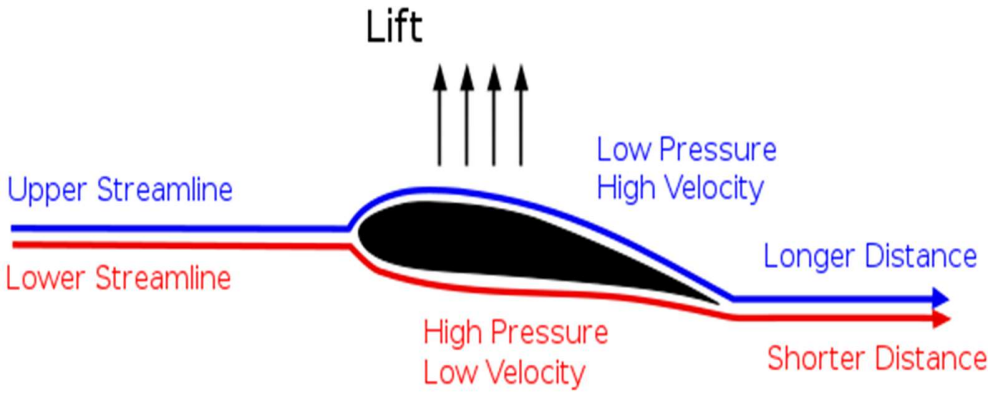


Fig 7.2 Lift generation

The blades of the darrieus VAWT have an airfoil shape in order to produce lift. Whenever airfoil and lift are mentioned, it is necessary to talk about the angle of attack. In darrieus VAWT the blade's velocity vector always changes direction but it is always tangential. The varying angle of attack is given below as:

$$\alpha = \tan^{-1}\left(\frac{\sin\theta}{\cos\theta + \lambda}\right)$$

Where θ is the azimuthal position and λ is the tip speed ratio.

$$\lambda = (\omega R)/U$$

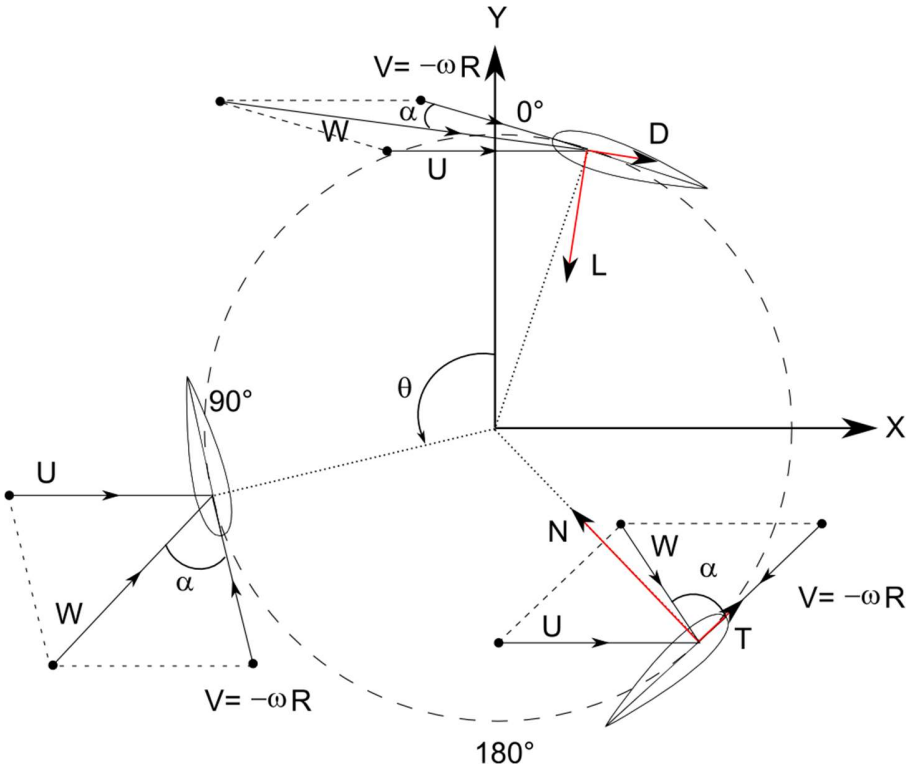


Fig 7.3 Angle of attack at each point

Where W is the resultant velocity.

$$W = U\sqrt{1 + 2\lambda\cos\theta + \lambda^2}$$

The amount of power, P that can be absorbed by a wind turbine:

$$P = \frac{1}{2} C_p \rho A v^3$$

Where Cp is the power coefficient, ρ is the density of air, A is the swept area of the turbine and v is the wind speed.

7.2.2 OUR APPROACH

We have decided to use darrieus VAWT but it doesn't self-start in order to bypass this problem we decided to use the hybrid of Darrieus and Savonius VAWT design which is the darrieus VAWT with J-type blades. This blade utilizes both lift and drag and doesn't suffer from self-starting problems.

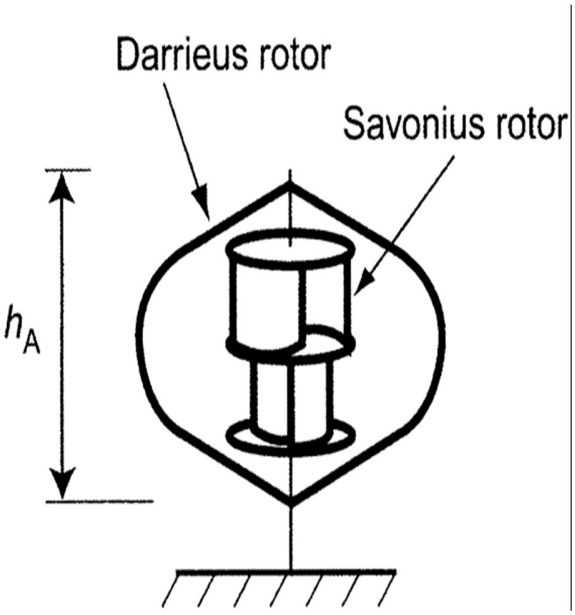


Fig 7.4.1 Traditional Hybrid Solution

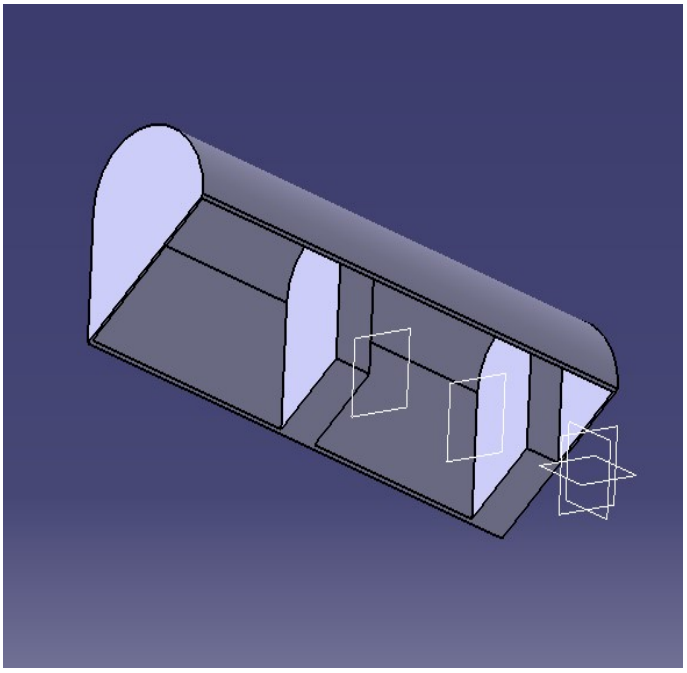


Fig 7.4.2 Our proposed design of turbine

7.3 Choice of Battery

- All our components require a total current of 80-100 mA and a maximum voltage of 5V. So, we will be needing a battery of minimum 5V and a current rating of 1000-2000 mAh.
- Our generator is not able to produce this amount of potential then we will be needing a boost converter.
- A boost converter works on the principle of a constant power by increasing the voltage with a decrease in the current.

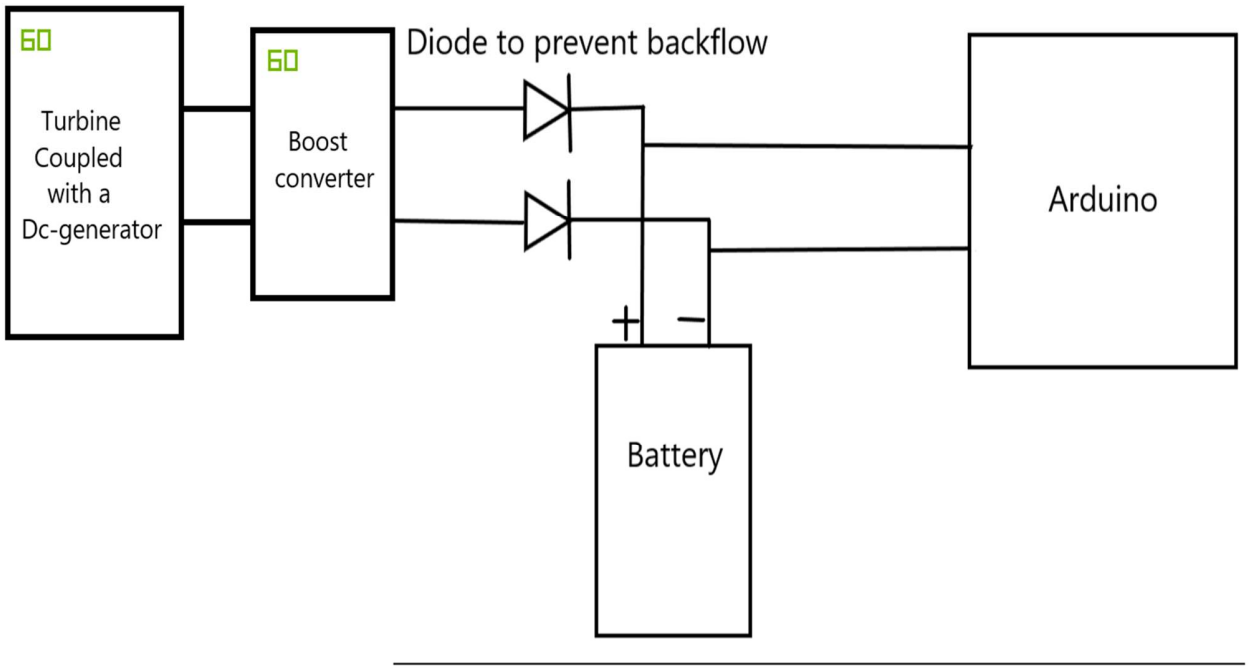


Fig 7.5 Current generation circuit diagram

CHAPTER 8
CALCULATIONS AND RESULTS

8.1 Power Calculations

The wind turbine works on the principle of converting kinetic energy of the wind to mechanical energy. The kinetic energy of any particle is equal to one half its mass times the square of its velocity.

K.E=½mv^2, m =ρAV

K E = ½ ρAV^3 watts.

Where, A is swept area of turbine.

ρ is density of air (1.225 kg/m^3)

V is wind velocity.

Assuming, P=1 watts.

Considering turbine efficiency as 25% and generator efficiency 85%,

Value of P at full efficiency, P = 1/ (0.25*0.85) = 4.7

P= ½ ρAV^3

For a wind velocity of 6 m/s and density of air is 1.225 kg/m^3

4.7= ½*1.125*A*(6)^3

A= 0.0356 m^2.

A = D*H, where D and H are the diameter and height of the turbine.

Taking diameter as 16 cm, height of turbine can be calculated as

H=A/D=22.3 cm, D=16 cm

- The design parameters are :
  1) Height of the blade is 22.3cm
  2) Connecting rod to shaft is 7.5cm
  3) 3 Blades
- We fabricated this using Polylactide (PLA) with the help of additive manufacturing.
- For all the parts an infill density of 60% was selected.

8.2 Design Specification

Wind Rotor	Rated Power	0.85W
	Rated Speed	6.1m/s
	Rotor Diameter	16 cm
	Swept Area	356.8 cm^2
	Gear Box Type	None
	Brake	Not Required
Turbine	Blade Type	J-Type
	Blade Number	3
	Blade Material	PLA
Blade Dimension	Length	22.3 cm
	Radius	20 cm

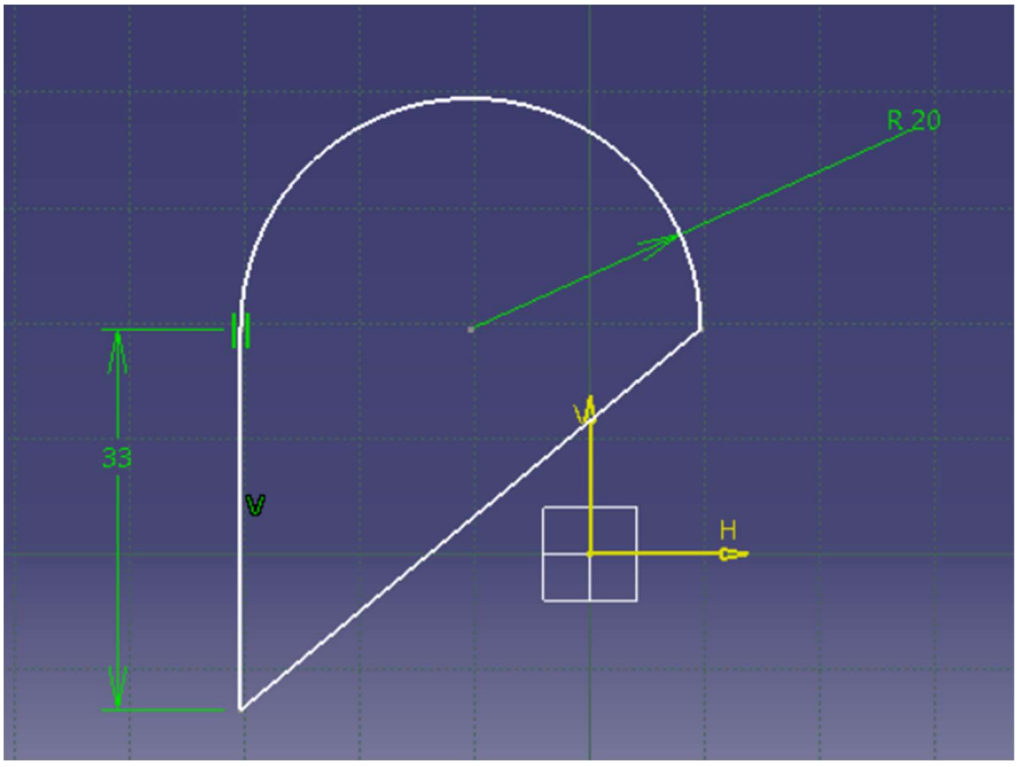


Fig 8.1 Airfoil shape

8.3 Results

Wind Speed (m/s)	Minimum Voltage (V)	Maximum Voltage (V)	Mean Voltage (V)	RPM
3.5	3.48	3.54	3.51	244
4.3	3.82	3.84	3.83	315
4.7	3.97	4.07	4.02	356
5	4.16	4.22	4.19	372
5.6	4.39	4.46	4.425	397
6.1	4.68	4.73	4.705	422

OBSERVATION TABLE

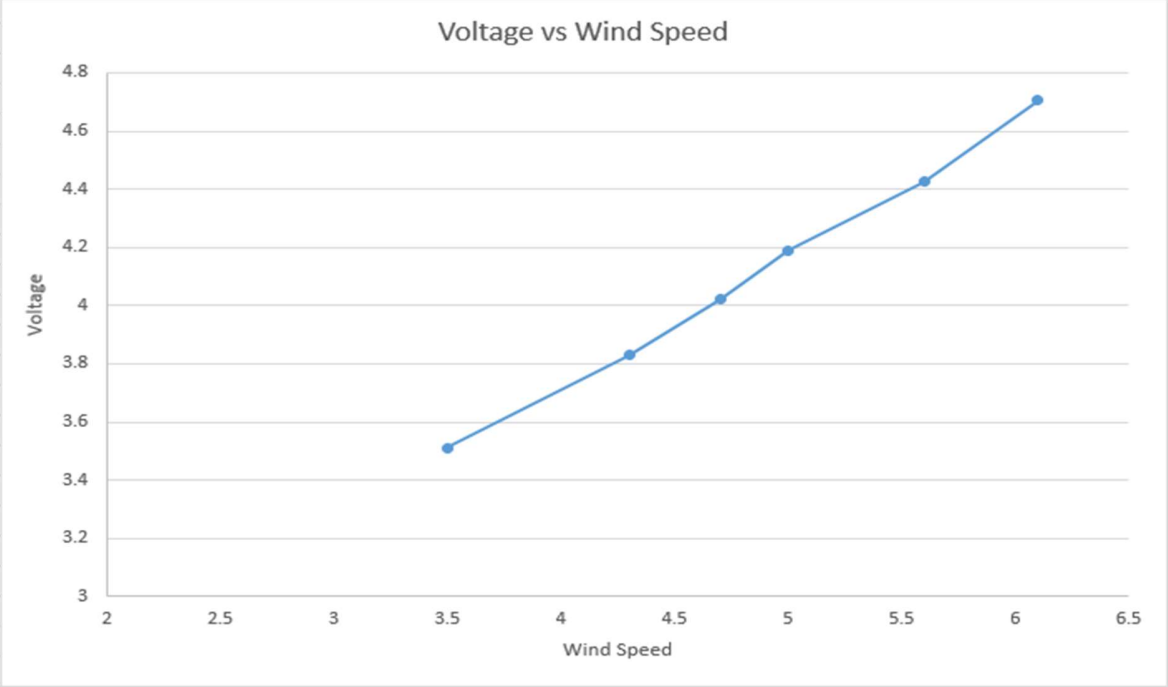


Fig 8.2.1 Plot showing Voltage vs Wind Speed

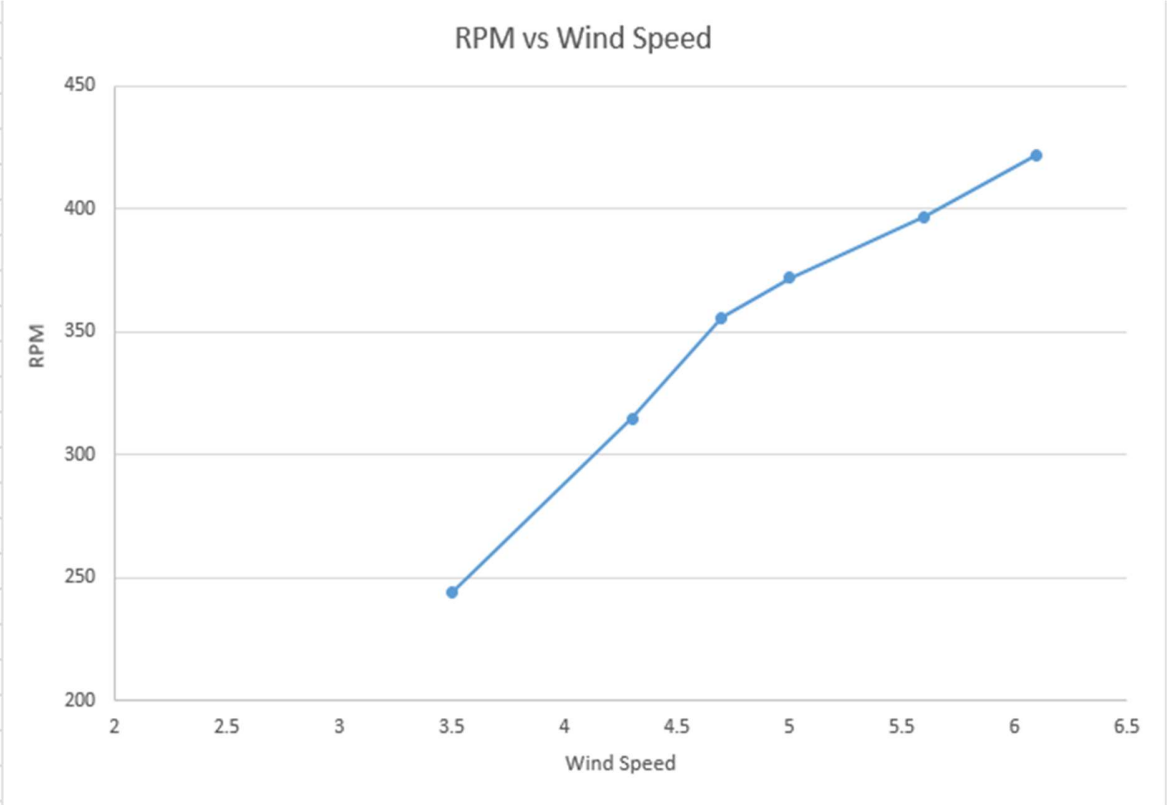


Fig 8.2.2 Plot showing RPM vs Wind Speed

CHAPTER 9

CONCLUSION AND FUTURE WORKS

A lot of work that has been done in the field of real-time prediction in IoT devices. Research is still going on to optimize the k-NN algorithm and make it faster. As discussed above, each of them suffered from their drawbacks. The inference that removing the prototypes with neighbors of same class won't change the Voronoi boundaries was made and these points were removed. Then we used k-Means and shifted our cluster centers optimally to get our required prototypes.

We can now reduce several thousands of data points to hundreds of prototypes while maintain comparable accuracy. Therefore, all the real time computations can be done using IOT devices. We will create a distributed network of IOT devices to solve the problem of forest fire prediction. Due to reduction in number of prototypes, shifting process executes in seconds. But if we have a dataset that has a lot of features, we can use PCA (Principal Component Analysis) to reduce the dimensionality of our dataset and our future work includes making this as less computationally feasible as possible.

Our J-Shaped VAWT is producing 0.85W and 4.72V which is sufficient to run an Arduino. Additionally, we can use boost convertor circuit or buck convertor circuit to meet the need of extra voltage and current requirement respectively. The issue with Darrius and Savonius turbine to provide starting torque and low lift respectively is been solved by making a hybrid turbine, which involve combination of Savonius and Darrius design. We can further enhance performance by improving the airfoil shape. In particular for forest situation, cage can be made to prevent outside debris.

References

[1] Decaestecker C (1997) Finding prototypes for nearest neighbor classification by means of gradient descent and deterministic annealing. Pattern Recogn 30(2):281–288.

[2] Hinton, Geoffrey & Roweis, Sam. (2003). Stochastic Neighbor Embedding.

[3] Kusner, M.J., Tyree, S., Weinberger, K.Q., & Agrawal, K. (2014). Stochastic Neighbor Compression. ICML.

[4] Van de Merckt, Thierry. (1992). NFDT: A System that Learns Flexible Concepts Based on Decision Trees for Numerical Attributes.. 322-331.

[5] Manik Varma and Ashish Kumar. Resource-efficient Machine Learning in 2 KB RAM for the Internet of Things.

[6] Manik Varma and Ashish Kumar. ProtoNN: Compressed and Accurate kNN for Resource-scarce Devices.

[7] Wenlin Wang, Changyou Chen, Wenlin Chen. Deep Distance Metric Learning with Data Summarization

[8] Koontz, Narendra and Fukunaga. A Graph-Theoretic Approach to Nonparametric Cluster Analysis

[9] https://www.datascience.com/blog/k-means-clustering.

[10] https://www.pcboard.ca/flame-sensor-module.

[11] Jacob Goldberger, Sam Roweis, Geoff Hinton, Ruslan Salakhutdinov, Neighbourhood Components Analysis

[12] Jon Louis Bentley, Multidimensional Binary Search Trees Used for Associative Searching

[13] https://en.wikipedia.org/wiki/K-d_tree

[14]https://www.researchgate.net/publication/330657388_255_Experimental_Investigations_of_Hybrid_Vertical_Axis_Wind_Turbine

[15]https://www.researchgate.net/publication/319678764_Review_Paper_Overview_of_the_Vertical_Axis_Wind_Turbines

[16]https://en.wikipedia.org/wiki/Binary_space_partitioning

Datasets Used

1. https :// archive . ics . uci . edu / ml / datasets / isolet
2. https :// archive . ics . uci . edu / ml / datasets / glass + identification
3. Sklearn Breast_Cancer Dataset
4. Sklearn Load_digits Dataset
5. Sklearn Wine Dataset.
6. Sklearn Iris Dataset.