SOFTWARE MODELS

The dataset after being pre-processed was fed to various models such as: Random Forest, DecisionTree, KNeighbors and Gaussian. To avoid overfitting, data was divided in train and test in preprocesssing with train datapoint being 75% and training the model at the remainig 25%.

- **Random Forest Classifier**

```
In [18]: forest_clf = RandomForestClassifier(n_estimators=50)
         forest_clf.fit(X_train, y_train)
         pred_forest = forest_clf.predict(X_test)

In [20]: accuracy_score(y_test,pred_forest)

Out[20]: 0.9988440821080216
```

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees. For regression tasks, the mean or average prediction of the individual trees is returned. Random decision forests correct for decision trees' habit of overfitting to their training set. Random forests generally outperform decision trees, but their accuracy is lower than gradient boosted trees. However, data characteristics can affect their performance.

As random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction and we have limited number of features hence a decison tree is suitable here.

- **Decision Tree Classifier**

```
In [22]: DT_clf = DecisionTreeClassifier()
         DT_clf.fit(X_train, y_train)
         pred_DT = DT_clf.predict(X_test)
         print("Accuracy:",accuracy_score(y_test, pred_DT))

         Accuracy: 0.9981336439074101
```

Decision tree learning or induction of decision trees is one of the predictive modelling approaches used in statistics, data mining and machine learning. It uses a decision tree (as a predictive model) to go from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). Tree models where the target variable can take a discrete set of values are called classification trees; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees. Decision trees are among the most popular machine learning algorithms given their intelligibility and simplicity.

Since we have already used random forest which is one of the best decison tree so we have implemented it with this.

- **KNN Classifier**

```
In [23]: KNN_clf = KNeighborsClassifier()
         KNN_clf.fit(X_train, y_train)
         pred_KNN = KNN_clf.predict(X_test)
         print("Accuracy:",accuracy_score(y_test, pred_KNN))

         Accuracy: 0.9970417819187655
```

The k-nearest neighbors (KNN) algorithm is a simple, supervised machine learning algorithm that can be used to solve both classification and regression problems. It's easy to implement and understand.

Since its purpose is to use a database in which the data points are separated into several classes to predict the classification of a new sample point and we have limited number of classes that are 5 this classifier was used.

- **Gaussian naive base Classifier**

```
In [25]: bayes_clf = GaussianNB()
         bayes_clf.fit(X_train, y_train)
         pred_bayes = bayes_clf.predict(X_test)
         print("Accuracy:",accuracy_score(y_test, pred_bayes))

         Accuracy: 0.7670490609986897
```

Naive Bayes are a group of supervised machine learning classification algorithms based on the Bayes theorem. It is a simple classification technique, but has high functionality. Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

Since other functions can be used to estimate the distribution of the data, but the Gaussian (or Normal distribution) is the easiest to work with because you only need to estimate the mean and the standard deviation from your training data.

Using K-fold Cross Validation -

Out of the classifiers used Decision tree, KNN and Random forest has shown promissing accuracies. With Random Forest leading with 99.8%. But the data being unbaised with many classes in the last 25%; hence it was required to shuffle the data randomly and split the data set in groups. Here, k-fold Cross Validation was done with k=10.

- **Decision Tree**

```
In [17]: DTree_clf = DecisionTreeClassifier()
         print(cross_val_score(DTree_clf, X, y, cv = 10, scoring = 'accuracy').mean())

         0.9668022470819995
```

Here, after doing cross validation with cv=10 we have accuracy of 96.6%.

- **KNN**

```
In [18]: K_NN_clf = KNeighborsClassifier()
         print(cross_val_score(K_NN_clf, X, y, cv = 10, scoring = 'accuracy').mean())

         0.9609702116708373
```

After doing cross validation in KNN with cv=10 we have an accuracy of 96%

- **Random Forest**

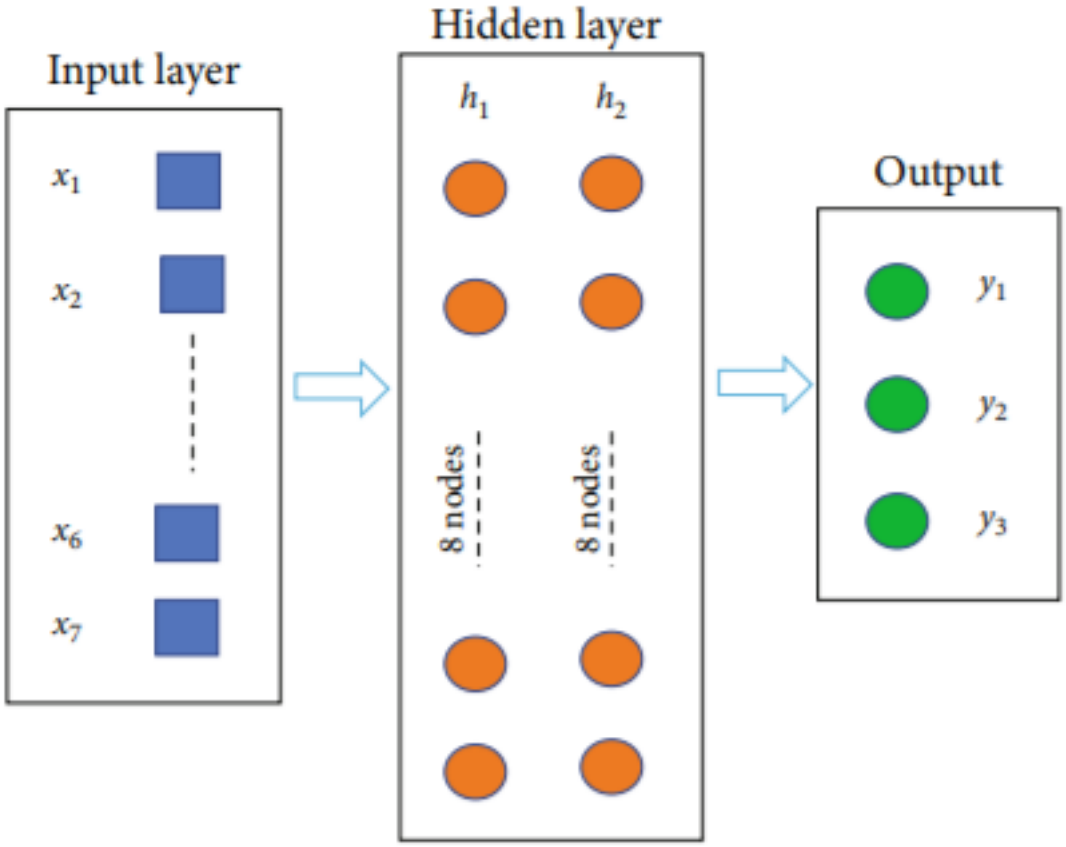
```
In [14]: Randforest_clf = RandomForestClassifier(n_estimators=50)
         print(cross_val_score(Randforest_clf, X, y, cv = 10, scoring = 'accuracy').mean())

         0.9732930058190219
```

After applying cross validation to this model we have an average accuracy of 97.4%.

Upon training the Decision tree, KNN and Random forest with cross validation it was observed that the accuracy percentage was reduced by 2-3%. This was due to the pressence of unbaised dataset hence the accuracy result after cross validation is the accurate one. The highest accuracy observed was of Random Forest: with an average of 97.45%.

NEURAL NETWORKS

The performance of a trained model for a PV system using ML techniques can greatly vary if new data is fetched from a different environmental condition, especially data from the winter season. The irradiation level in the winter is substantially lower than that in the summer, and studies have shown faults occurring in such lower irradiation levels have higher chances of remaining undetected.



DNN / MLP

```
Epoch 25/30
1099037/1099037 [==============================] - 176s 160us/step - loss: 0.0669 - accuracy: 0.9831 - val_loss: 0.0503 - val
_accuracy: 0.9872
Epoch 26/30
1099037/1099037 [==============================] - 1381s 1ms/step - loss: 0.0664 - accuracy: 0.9827 - val_loss: 0.0492 - val_
accuracy: 0.9868
Epoch 27/30
1099037/1099037 [==============================] - 488s 444us/step - loss: 0.0621 - accuracy: 0.9830 - val_loss: 0.0379 - val
_accuracy: 0.9909
Epoch 28/30
1099037/1099037 [==============================] - 490s 445us/step - loss: 0.0576 - accuracy: 0.9836 - val_loss: 0.0407 - val
_accuracy: 0.9881
Epoch 29/30
1099037/1099037 [==============================] - 485s 441us/step - loss: 0.0545 - accuracy: 0.9840 - val_loss: 0.0561 - val
_accuracy: 0.9824
Epoch 30/30
1099037/1099037 [==============================] - 486s 442us/step - loss: 0.0576 - accuracy: 0.9832 - val_loss: 0.0513 - val
_accuracy: 0.9849
dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])
```

Fig: Result of Multi-layer Perceptron of 30 Epochs.

Confusion Matrix

```
Out[21]: Text(0.5, 1.0, 'Confusion Matrix')
```

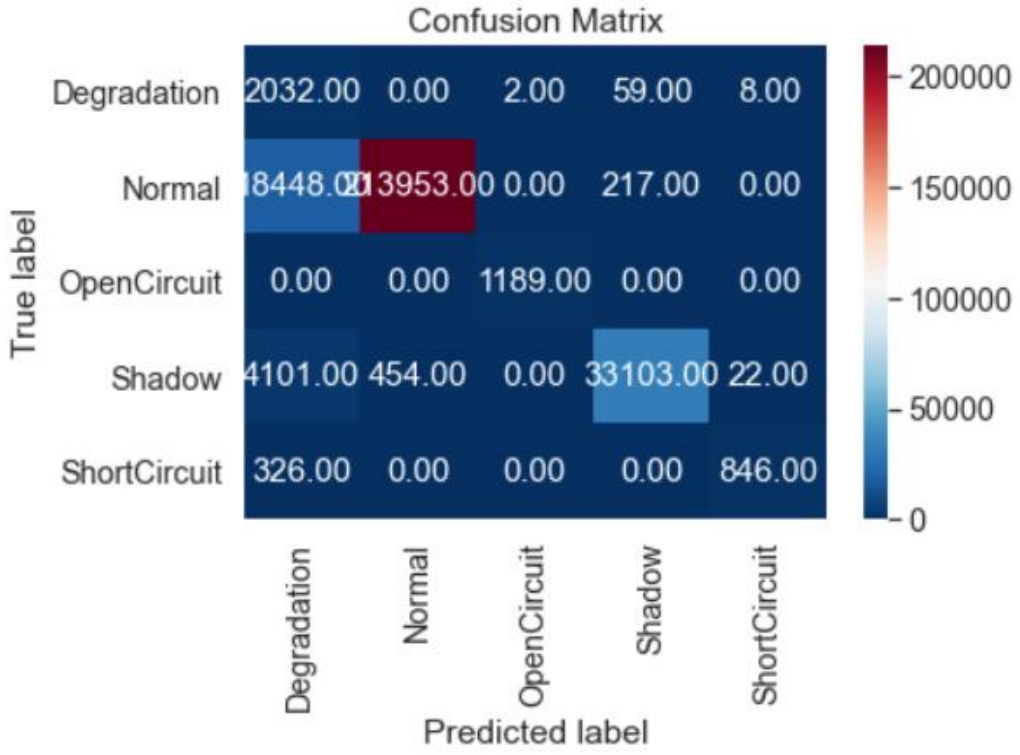


Fig: depiction of correlation between predicted and True labels of various states

To check the bias-variance tradeoff, a k-fold crossvalidation test is performed with the 5 validations split into the training data. Also, for improving the model and reducing overfitting, we implemented the dropout regularization technique. The dropout rates of 0.1 and 0.2 were selected for the first and second hidden layers, respectively. The result of the evaluation, improvement, and tuning of the model is provided:

```
In [17]: def build_classifier():
             model = Sequential()
             model.add(Dense(input_dim = 6, units = 8, kernel_initializer = 'uniform', activation = 'relu'))
             model.add(Dense(units = 8, kernel_initializer = 'uniform', activation = 'relu'))
             model.add(Dense(units = 5, kernel_initializer = 'uniform', activation = 'softmax'))
             model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
             return model

         model = KerasClassifier(build_fn = build_classifier, batch_size = 5, epochs = 20)
         accuracies = cross_val_score(estimator = model, X = X_train, y = y_train, cv = 5) #CV = K-fold cross val. splits
         mean = accuracies.mean()
         variance = accuracies.std()
         print("Average accuracy:", mean, "Variance:", variance)
         Epoch 12/20
         879230/879230 [==============================] - 117s 133us/step - loss: 0.0588 - accuracy: 0.9822
         Epoch 13/20
         879230/879230 [==============================] - 117s 133us/step - loss: 0.0540 - accuracy: 0.9828
         Epoch 14/20
         879230/879230 [==============================] - 117s 133us/step - loss: 0.0552 - accuracy: 0.9835
         Epoch 15/20
         879230/879230 [==============================] - 117s 133us/step - loss: 0.0493 - accuracy: 0.9840
         Epoch 16/20
         879230/879230 [==============================] - 117s 133us/step - loss: 0.0538 - accuracy: 0.9839
         Epoch 17/20
         879230/879230 [==============================] - 117s 133us/step - loss: 0.0502 - accuracy: 0.9842
         Epoch 18/20
         879230/879230 [==============================] - 117s 133us/step - loss: 0.0455 - accuracy: 0.9849
         Epoch 19/20
         879230/879230 [==============================] - 117s 133us/step - loss: 0.0444 - accuracy: 0.9852
         Epoch 20/20
         879230/879230 [==============================] - 117s 133us/step - loss: 0.0462 - accuracy: 0.9853
         219807/219807 [==============================] - 12s 56us/step
         Average accuracy: 0.9780726313591004 Variance: 0.005480874747660622
```

Fig: Result of MLP of 20 Epochs with cv =5