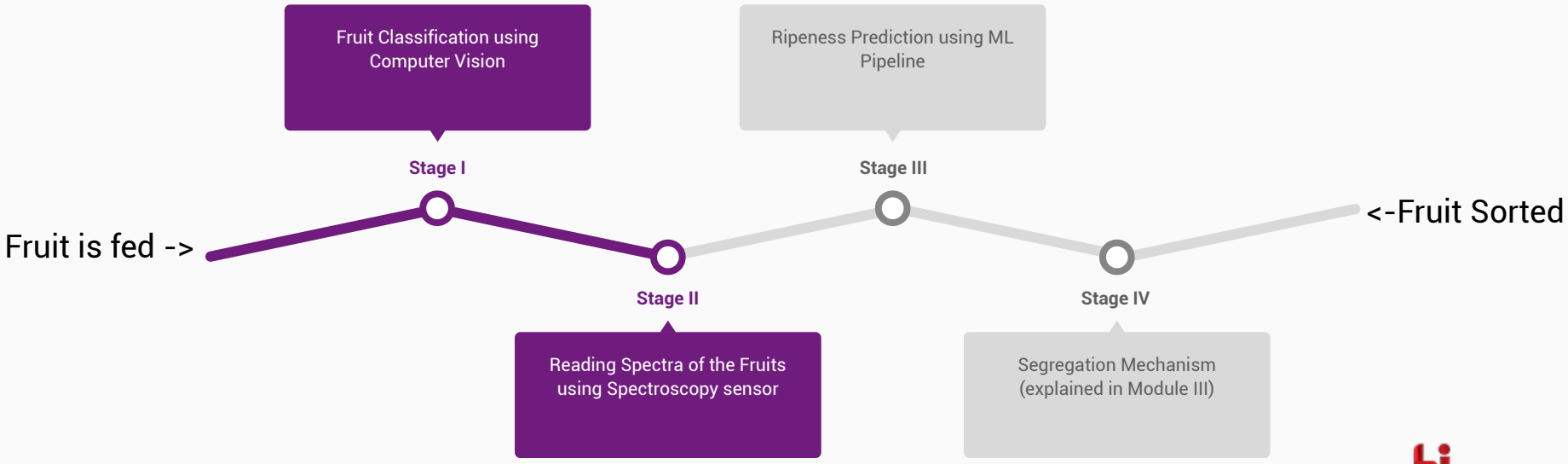
DATA SCIENCE PART

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

The Main Pipeline

A pipeline is a combined set of methodologies, adhering to those can help us succeed the task at hand, i.e. to develop a software/algorithm capable of solving the problem statement.

Problem Statement - To Detect which fruit is fed in the system, and to segregate it according to its ripeness level.



THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Stage I - Fruit Classification

Aim - Development of a system, capable enough to determine the type of the fruit fed into the machine.

Existing Solutions to the problem-

- Fruit tag based solutions, and detection using NFC Sensor (Lazaro, Antonio, et al)
- Fruit recognition using color sensors (Gongal, A., et al)
- Fruit Detection System using DeepNeural Networks (Sa, Inkyu, et al)
- Fruit detection using Thermal Sensor and other feature based system (Patel, Hetal N., R. K. Jain, and Manjunath V. Joshi.)

Problems in these existing solutions

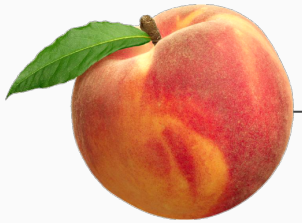
- Fruits have to be tagged beforehand, not flexible.
- Same colored fruits can have errors.
- High computation power is required, hardware insufficient
- Thermal cameras are expensive.

Stage I - Fruit Classification

Aim - Development of a system, capable enough to determine the type of the fruit fed into the machine.

Proposed Solution - Detection of fruits using Computer Vision and Image Classification. The image data will be extracted from the Raspberry Pi Camera and then passed to the designed software for predicting the type of the fruit. Advantages of this method -

- Computer Vision (image classification) allows mathematical models to be built uniquely for all the fruits, regardless of their color.
- The fruits need not to be prepared beforehand.
- Computation power is surely needed, but using some clever API's, they can be surpassed.
- Huge datasets already exist on the Internet under open license.
- Once models are trained, it's easy to detect the types of fruits in real time, from all angles.



Computer Vision Model

Predictions

90% Apple
1% Apricot
9% Peach

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Cloud Computing using Custom Vision service by Microsoft Azure

Microsoft Azure is a cloud computing service created by Microsoft for building, testing, deploying, and managing applications and services on Microsoft's Cloud Architecture.

This slide aims at our procedure of achieving Fruit Classification, by following the instructions rooted in the book by (Mund, Sumit).

01

Data Preparation and Visualization

The dataset was procured from (https://www.kaggle.com/moltean/fruits). The data was pre-sorted, and thus images of all major fruits (Apple, Orange, Peach, Tangerine etc.) were selected for the training of the model.

02

Model Training

The model was uploaded to CustomVision.ai website, which is the service provided by Microsoft Azure Cloud to train custom image classifiers quickly. The model was trained and it's evaluation metrics were presented.

03

Model Testing

A small code was written in python, which uploads an image from the computer. The model trained on the cloud classifies the image and the result is displayed on the cloud server.

04

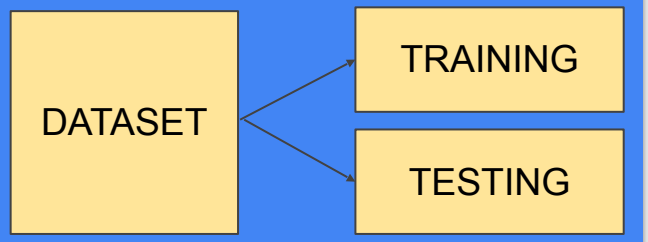
Model deployment

The model was finally deployed on the Raspberry Pi. The code, on the click of a button on the GUI, would click a picture from the Raspberry Pi Camera, and the model would, display the predicted fruit.

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Data Preparation and Visualization

The data was prepared and analyzed using python.



CODE TO CHECK DATASET SIZE

```
# First, we are going to load the file names and their respective target labels into
numpy array!
from sklearn.datasets import load_files
import numpy as np

train_dir = '../input/fruits-360_dataset/fruits-360/Training'
test_dir = '../input/fruits-360_dataset/fruits-360/Test'

def load_dataset(path):
    data = load_files(path)
    files = np.array(data['filenames'])
    targets = np.array(data['target'])
    target_labels = np.array(data['target_names'])
    return files,targets,target_labels

x_train, y_train,target_labels = load_dataset(train_dir)
x_test, y_test,_ = load_dataset(test_dir)
print('Loading complete!')

print('Training set size : ' , x_train.shape[0])
print('Testing set size : ', x_test.shape[0])
```

OUTPUT

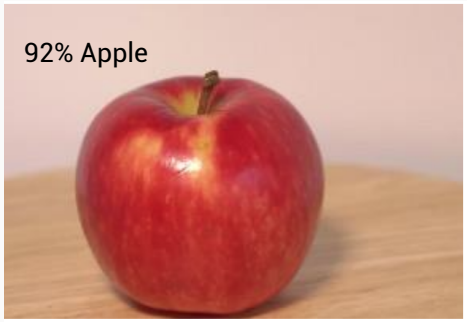
```
Loading complete!
Training set size :  41322
Testing set size :  13877
```

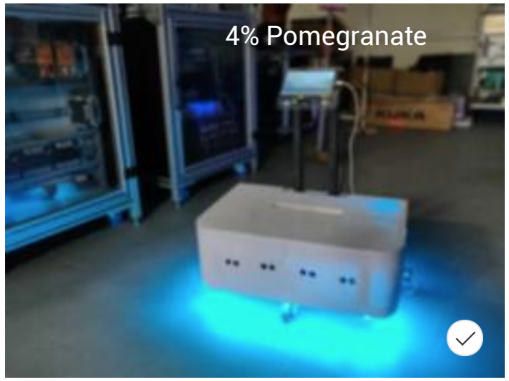
THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

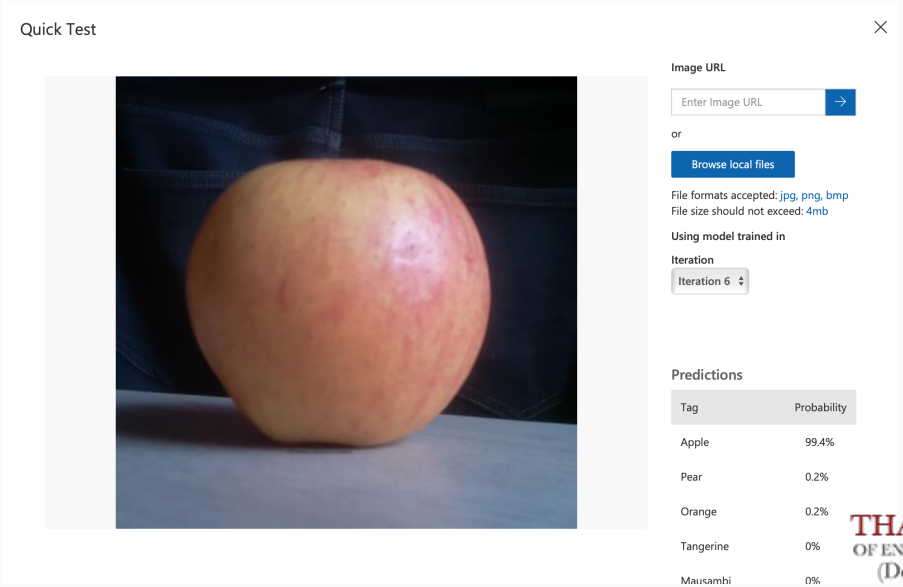
Model Testing

A more manual approach was chosen towards model testing, as we wanted to test the model on real-life images than take images from the test dataset.

The overall accuracy of the model on random images was around 83%. This was satisfactory enough for us to proceed.







THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

99%

Maximum Accuracy showed on detecting apples.

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Stage II - Reading Spectra of the Fruits using Spectroscopy sensor

Aim - Development of a hardware/software interface capable of data acquisition and solving

The most commonly used methods required for hardware/software interfacing of data acquisition systems are shown as below-

- ROS (Robot Operating System)- This is a middleware capable of solving complex hardware-software systems. (Quigley, Morgan, et al.)
- Python-Arduino Serial Interface. (Goradiya, B. H. A. R. G. A. V., and H. N. Pandya.)
- X-BEE and other wireless solutions such as bluetooth module.

Problems in commonly used methods required for hardware/software interfacing of data acquisition systems -

- This is very complex and is used for robotics and more complex systems
- Parallel transmission is not possible.
- X-BEE and bluetooth modules are slow and have a range constraint.

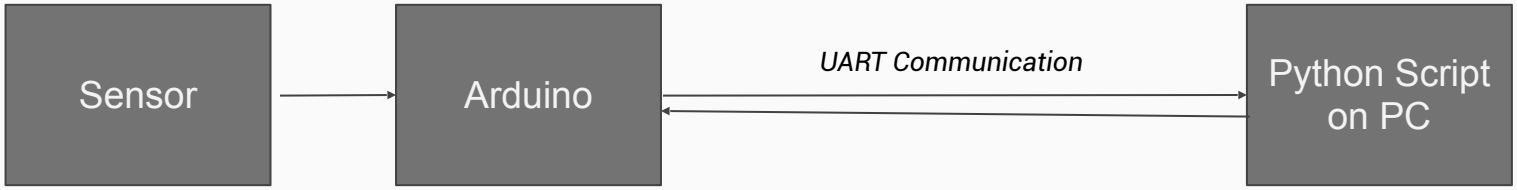
THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Stage II - Reading Spectra of the Fruits using Spectroscopy sensor

Aim - Development of a hardware/software interface capable of data acquisition and solving

Proposed Solution - Using Arduino and controlling it with Python on the Raspberry Pi, using Serial Communication. Advantages of this method -

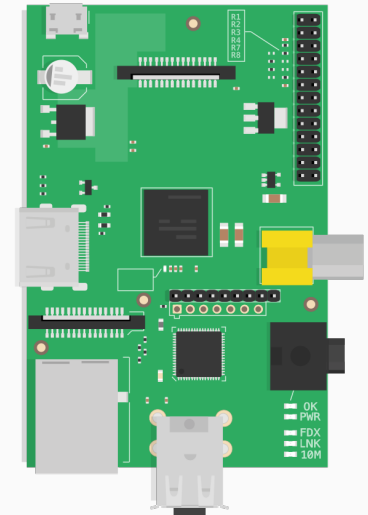
- Easy Communication method.
- Goes along with the UART Standard communication method (Universal Asynchronous Receiver/Transmitter).
- Easy reception and storage of data. No need to worry about buffer and data leakages.



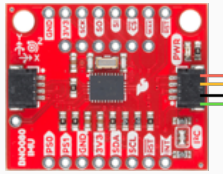
THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Electronic Setup

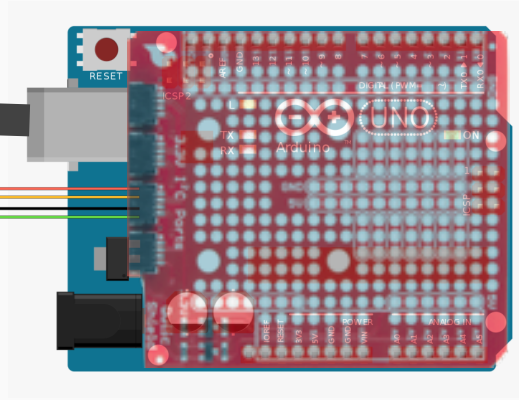
In order to create the dataset, and to use the sensor within the machine, this circuit here is used.



The Raspberry Pi 3.0



Triad SpectroScopy Sensor



Arduino Uno with QWUIIC Hat

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

1300+

Apples* analyzed for their spectroscopy data**

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

* Were not able to analyze more fruits, due to COVID-19. All apples of Golden-Delicious type.
** Each fruit was analysed in a dark environment (lighting exists within the sensor), at a distance of 2-3 cm from the fruit.

Stage III - Ripeness Prediction using Machine Learning Pipeline

Aim - Development of a software / algorithm capable to identify the level of ripeness, using scanned spectra values.

Existing Solutions to the problem-

- Color segmentation and Fuzzy Logic (Dadwal, Meenu, and Vijay Kumar Banga).
- Fruit ripeness measurement using Computer Vision (Al Ohali, Yousef).
- Electronic Nose based fruit ripeness estimation systems (Brezmes, J., et al.).
- Fruit Ripeness Estimation using Thermal Imaging (Sumriddetchkajorn, Sarun, and Yuttana Intaravanne).

Disadvantages-

- Less Explainable and not as accurate as other
- Tough to implement as increased computation is required
- Increased hardware complexity and customizations required for each fruit.
- Thermal imaging is a good solution, but only if coupled with other sensors.

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Stage III - Ripeness Prediction using Machine Learning Pipeline

Aim - Development of a software / algorithm capable to identify the level of ripeness, using scanned spectra values.

Proposed Solution - Ripeness level estimation with the help of a spectroscopy sensor, with the help of a specifically developed ML Algorithm. Advantages of this method -

- Every fruit has a unique spectra signature for it's ripe/unripe and over-ripe levels.
- Data driven approach, one general model cannot do justification to all.
- Highly scalable, can work on IoT as well as industry level spectroscopy sensors alike.
- If cost is an issue, the data acquisition can be done on expensive (industry level) spectroscopy sensor, but recognition can be done using the cheap (IoT) sensor (used in the machine).
- Future work can include using a thermal camera for a more accurate prediction metric.

Predicted Fruit Class

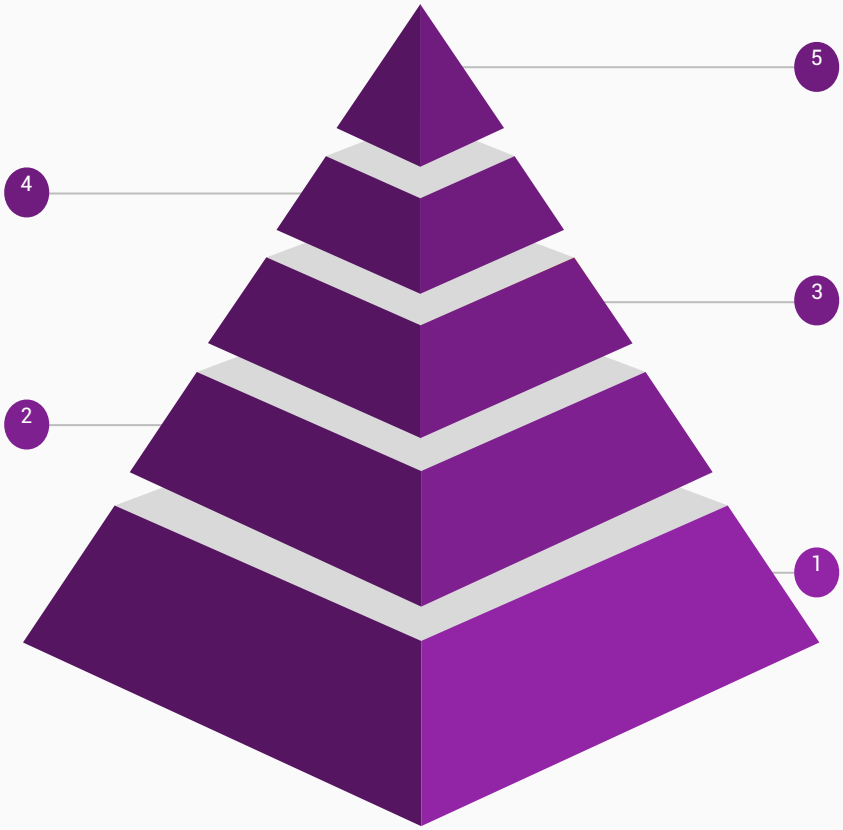
Ripeness ML Model

Ripeness Predictions

90% Ripe
1% Medium
9% Unripe

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Machine Learning Model Process Pipeline



5

Scoring Pipeline

A scoring pipeline will be designed, which would be tested on the test dataset.

4

Ensembling

The best models out of them will be stacked to create an ensemble for the final deployment.

3

Feature / Model Evolution (Engineering)

Using a lot of models, and a lot of engineered feature combinations, training is done to determine the best features.

2

Model Selection and Parameter Tuning

Identify which models to choose for our requirement. Parameters will be tuned to get the best model

1

Data Analysis

Data cleaning, data division, feature preprocessing and label/one-hot encoding. Also getting an insight using feature importance, using exploratory data analysis

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Step 1 - Data Analysis

In order to create a good machine learning application, the primary focus should be on the data engineering and analysis. The reason is because it is a data driven approach, and the model's parameters are totally judged by the data.

The data cleanup was manually done hence no point is visualizing that.

Data Ingestion

```
[ ]  url1 = "https://raw.githubusercontent.com/choprahetarth/FruitSpectroClassify/master/Raw
     url2 = "https://raw.githubusercontent.com/choprahetarth/FruitSpectroClassify/master/exp
     dataset1 = pd.read_csv(url1)
     dataset2 = pd.read_csv(url2)

     # Load the two datasets
```

```
[ ]  dataframe = pd.concat([dataset1,dataset2],axis=0).reset_index(drop=True)
     print(dataframe.shape)

     # conctenate the two datasets across axis=0 i.e. add rows not columns
```

```
     (1360, 19)
```

Data Division and Serialization

```
[ ]  X_train, X_test, y_train, y_test = train_test_split(dataframe, target, test_size=0.2)
     print (X_train.shape, y_train.shape)
     print (X_test.shape, y_test.shape)
     # split the datasets for testing, and print out their dimensions
```

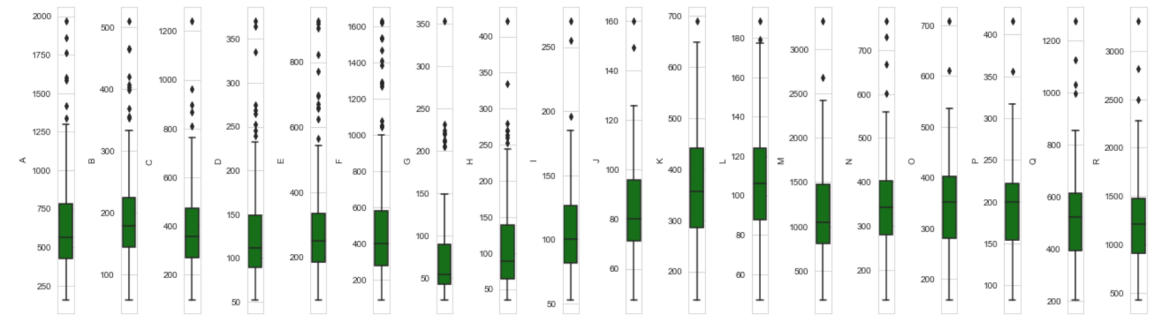
```
     (1088, 18) (1088,)
     (272, 18) (272,)
```

```
[ ]  # save all the files using pickle
     X_train.to_pickle('X_train.pickle')
     X_test.to_pickle('X_test.pickle')
     y_train.to_pickle('y_train.pickle')
     y_test.to_pickle('y_test.pickle')
     dataframeOriginal.to_pickle('dataframeOriginal.pickle')
     dataframe.to_pickle('dataframe.pickle')
     target.to_pickle('target.pickle')
```

Step 1 - Data Analysis

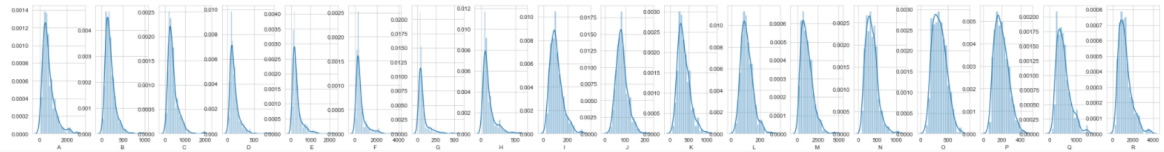
Exploratory Data Analysis. Checking the data for outliers is important, as this would help us decide to choose the correct model.

```
l = new_df.columns.values
number_of_columns=18
number_of_rows = len(l)-1/number_of_columns
plt.figure(figsize=(number_of_columns,5*number_of_rows))
for i in range(0,len(l)):
    plt.subplot(number_of_rows + 1,number_of_columns,i+1)
    sns.set_style('whitegrid')
    sns.boxplot(new_df[l[i]],color='green',orient='v')
    plt.tight_layout()
```



This diagram shows the distribution of each of the frequencies. All of them have indicated outliers.

```
plt.figure(figsize=(2*number_of_columns,5*number_of_rows))
for i in range(0,len(l)):
    plt.subplot(number_of_rows + 1,number_of_columns,i+1)
    sns.distplot(df[l[i]],kde=True)
```



This diagram shows that each frequency has a skewed relationship.

Keeping both characteristics in mind, it can be hypothesized that, a tree-based model will be better used on this dataset as it effectively handles outlier cases and original data modification will not be needed.

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Step 1 - Data Analysis

Feature Selection - RF Model Insights-

1. A model is used to train on the dataset using all features
2. The features are selected independently using permutation and combinations.
3. The model with the lowest error is selected and number of features are produced.

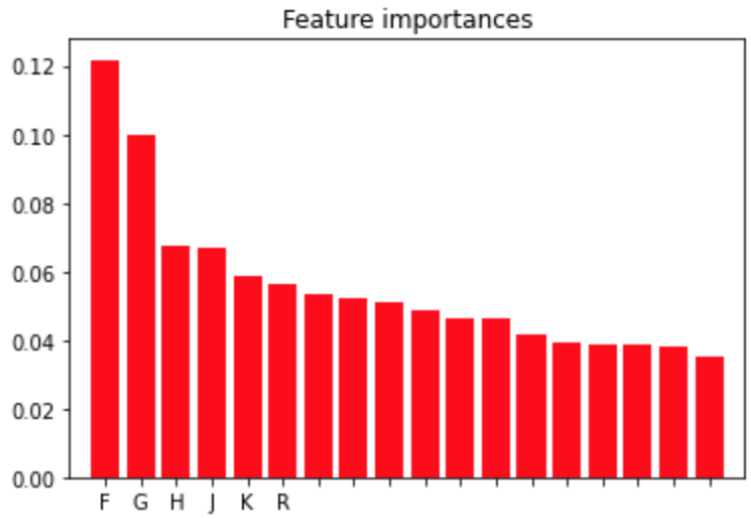
It somehow shows features common to the previous ones. But they don't include any feature from block A.

Thus an aggregate provided final features for selection as -

[F,G,H,J,K,R,D and M]

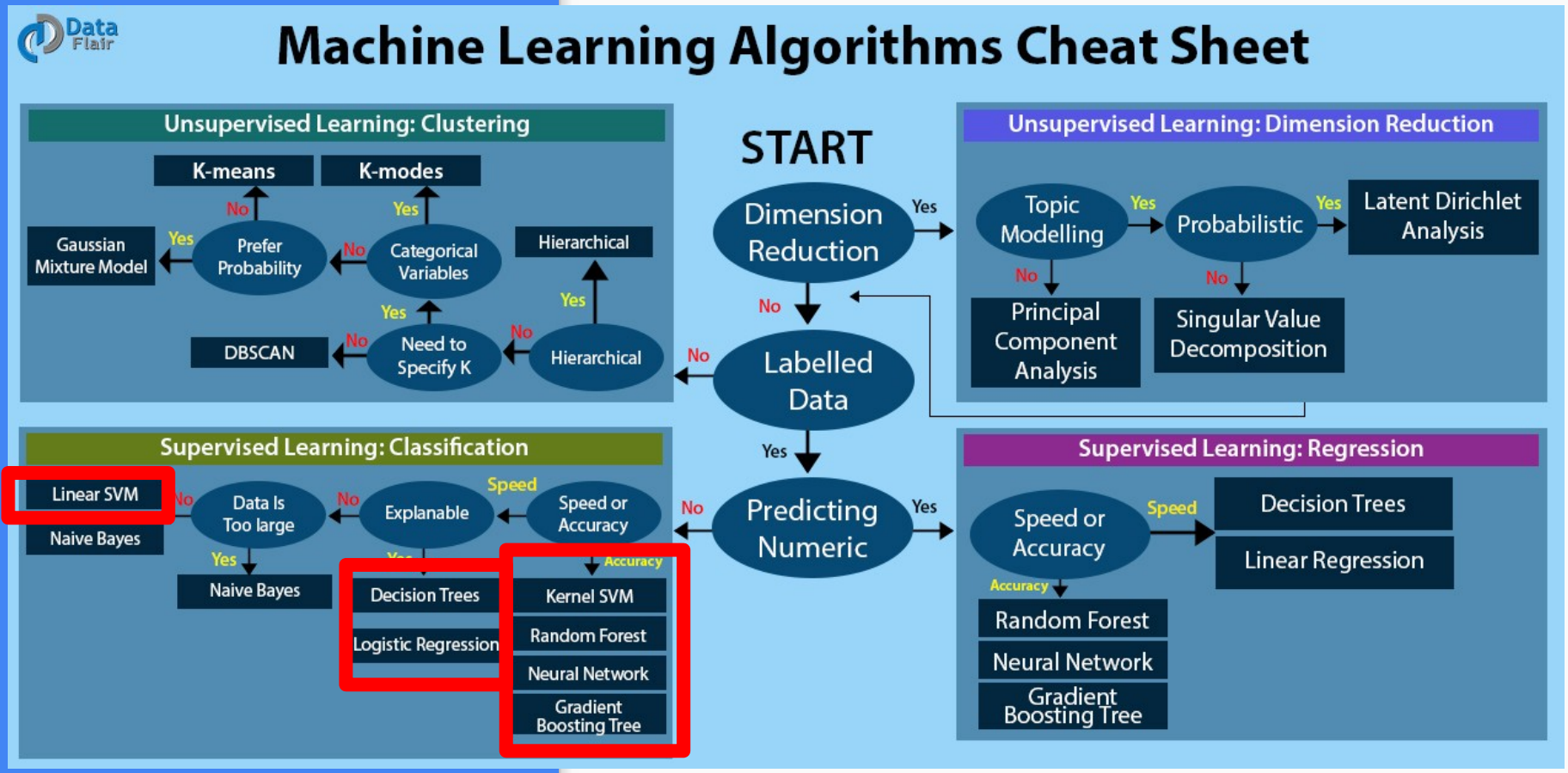
```
importances = sel.estimator_.feature_importances_
indices = np.argsort(importances)[::-1]
# X is the train data used to fit the model
plt.figure()
plt.title("Feature importances")
plt.bar(range(X_train.shape[1]), importances[indices],
       color="r", align="center")
plt.xticks(range(X_train.shape[1]), selected_feat)
plt.xlim([-1, X_train.shape[1]])
plt.show()

# PLOT
```



Step 2 - Model Selection and Parameter Tuning

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)



Step 2 - Model Selection and Parameter Tuning

Model Selection -
All the models selected from the previous slide are researched about thoroughly and their pros and cons are compared in order to select the best models out of them.

Random Forest fulfills all the requirements.

Down the line, with more data Neural Networks can also be used. (Future Work)

Model Name	Pros	Cons
Logistic Regression	• High Explainability • Good Accuracy	• Have to normalize the data • Computationally expensive • Used in binary classification
Neural Networks (ANN/DNN)	• Maximum accuracy	• Computationally expensive • Less Data • Less Explainable
Linear SVM	Same as Logistic Regression	Same as Logistic Regression
Kernel SVM	• High accuracy	• Have to normalize the data • Computationally expensive • Used in binary classification
Decision Trees / Random Forest	• Fast processing • Suitable for multiclass classification • Bagging/Boosting can increase accuracy • Data can be used “as is”, less pre-processing.	• Less explainable

Step 2 - Model Selection and Parameter Tuning

Model Benchmarking.
A model is trained using the basic parameters of random forest. This will serve as the benchmark of all the models that follow.

Notice that there are many parameters to take care of.
Out of these, the most important ones are- n_estimators, max_features, max_depth, min_samples_leaf, and min_samples_split.

For a more comprehensive guide on random forest- https://towardsdatascience.com/an-implementation-and-explanation-of-the-random-forest-in-python-77bf308a9b76

2. **Initial Benchmark Dataset** - A single RF model was chosen and was trained on the default parameters.

```
# Import the model we are using
from sklearn.ensemble import RandomForestClassifier
# Instantiate model
rf = RandomForestClassifier(n_estimators= 10, random_state=42)
# Train the model on training data
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
accuracy = accuracy_score(y_test,y_pred)
print('Accuracy:', round(accuracy, 2), '%.')
# THIS IS THE BASELINE MODEL
```

```
Accuracy: 0.78 %.
```

```
Parameters currently in use:

      {'bootstrap': True,
      'ccp_alpha': 0.0,
      'class_weight': None,
      'criterion': 'gini',
      'max_depth': None,
      'max_features': 'auto',
      'max_leaf_nodes': None,
      'max_samples': None,
      'min_impurity_decrease': 0.0,
      'min_impurity_split': None,
      'min_samples_leaf': 1,
      'min_samples_split': 2,
      'min_weight_fraction_leaf': 0.0,
      'n_estimators': 100,
      'n_jobs': None,
      'oob_score': False,
      'random_state': 42,
      'verbose': 0,
      'warm_start': False}
```

Step 2 - Model Selection and Parameter Tuning

Parameter Tuning - Random Search

The most important parameters are each provided with a list of common values. A random search is partaken between all the parameters and the model with the best parameters are chosen. That model was then evaluated the test dataset.

An improvement of 0.95% was shown by the new random searched model parameters..

```
from sklearn.model_selection import RandomizedSearchCV

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
```

```
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}

pprint(random_grid)
```

```
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
```

These following parameters were provided by the random search as the best parameters.

```
rf_random.best_params_
```

```
{'bootstrap': True,
 'max_depth': 90,
 'max_features': 'sqrt',
 'min_samples_leaf': 4,
 'min_samples_split': 2,
 'n_estimators': 800}
```

```
best_random = rf_random.best_estimator_
random_accuracy = evaluate(best_random, X_test, y_test)
```

```
Model Performance
Accuracy = 0.78%.
```

```
print('Improvement of {:0.2f}%.'.format( 100 * (random_accuracy - base_accuracy) / base_accuracy))
```

```
Improvement of 0.95%.
```

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Step 2 - Model Selection and Parameter Tuning

Parameter Tuning - Grid Search

The random parameters were fine tuned using finer values and a linear search was done to find the model with the maximum accuracy.

The final model is shown with these parameters and all the evaluation metrics are shown of the model.

For a more comprehensive guide on evaluation metrics - https://medium.com/thalus-ai/performance-metrics-for-classification-problems-in-machine-learning-part-i-b085d432082b

```
param_grid = {
    'bootstrap': [True],
    'max_depth': [180,200],
    'max_features': [3, 4],
    'min_samples_leaf': [8,9,10],
    'min_samples_split': [7,8,10],
    'n_estimators': [50,80, 100, 120,180]
}

# Create a base model
rf = RandomForestClassifier(random_state = 42)

# Instantiate the grid search model
grid_search_second = GridSearchCV(estimator = rf, param_grid = param_grid,
                                  cv = 3, n_jobs = -1, verbose = 2, return_train_score=True)

grid_search_second.fit(X_train, y_train)
grid_search_second.best_params_
```

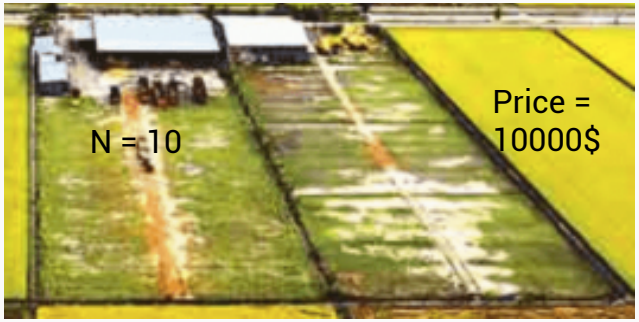
```
Fitting 3 folds for each of 180 candidates, totalling 540 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  37 tasks       | elapsed:    8.7s
[Parallel(n_jobs=-1)]: Done 158 tasks       | elapsed:   38.5s
[Parallel(n_jobs=-1)]: Done 361 tasks       | elapsed:  1.5min
[Parallel(n_jobs=-1)]: Done 540 out of 540 | elapsed:  2.3min finished
{'bootstrap': True,
 'max_depth': 180,
 'max_features': 4,
 'min_samples_leaf': 9,
 'min_samples_split': 7,
 'n_estimators': 100}
```

```
'Grid Search 2': 0.7867647058823529 - ACCURACY SCORE
'Grid Search 2': 0.5173868518889556 - F1 SCORE
'Grid Search 2': 0.6773402212457021 - ROC AUC SCORE
'Grid Search 2': 0.3546804424914043 - GINI SCORE
'Grid Search 2':array([[  0,   0,   0], - CONF MATRIX
                       [ 11, 157,  26],
                       [  0,  21,  57]])
'Grid Search 2': 0.5133491933 - SENSITIVITY SCORE
'Grid Search 2': 0.8521024767 - SPECIFICITY SCORE
```

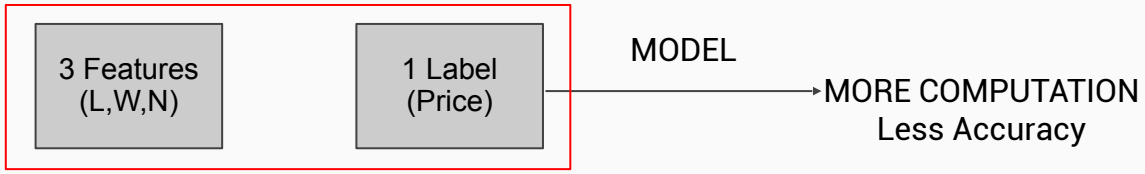
Step 3 - Feature / Model Evolution (Engineering)

Feature and Model evaluation engineering is a very important component of Machine Learning. An example on the right explains so.

Let's say that there is a case where you have to predict the Price (label) and the features given to you are- Length of the Land(L), Breadth of the Land (W), and Number of people living (N).

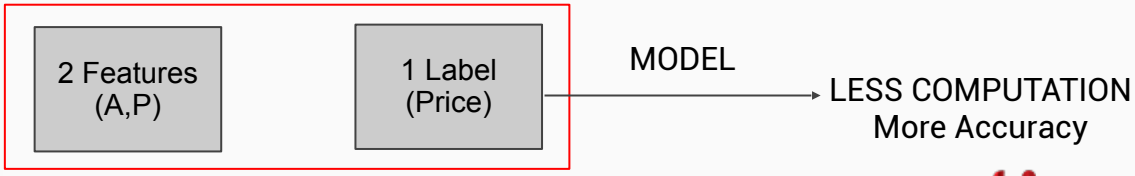


DATASET



In this very case, let us assume that the features are engineered. L*W = Area (A), and Area/N = Population density (P).
This case would definitely prove to have less computation and more accuracy.

DATASET



THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Step 3 - Model / Feature Evolution (Engineering)

In order to proceed with Step 3, 4, and 5, a software is required, because of certain limitations -

1. Lack of time and processing power.
2. Lack of skill and prowess in understanding advanced AI concepts such as bagging/ boosting/ensembling.

Name of the software used- **Driverless.AI**

Model Evolution

From our previous stages, we have concluded that Random Forest classification can act as a baseline model, and using advanced concepts like bagging/boosting one can increase random forest performance. Thus, Driverless.AI was used to train 199 models trained and scored to evaluate features and model parameters.

Model Summary- found the optimal parameters for lightgbm (Bagging) and gbtree (xgboost) (Boosting) models by training models with different parameters.

job order	booster	nfeatures	scores	training times (sec)
14	gbtree	15	0.731742908	30.7760746479
13	gbtree	17	0.7320789229	26.8220162392
7	lightgbm	17	0.7455348029	16.0659251213
4	lightgbm	18	0.7486262132	17.6930742264
15	gbtree	16	0.7488716083	25.5865011215
2	lightgbm	17	0.7493829828	17.9120388031
3	lightgbm	17	0.7495952813	13.9095699787
0	lightgbm	17	0.7497566403	16.9010481834

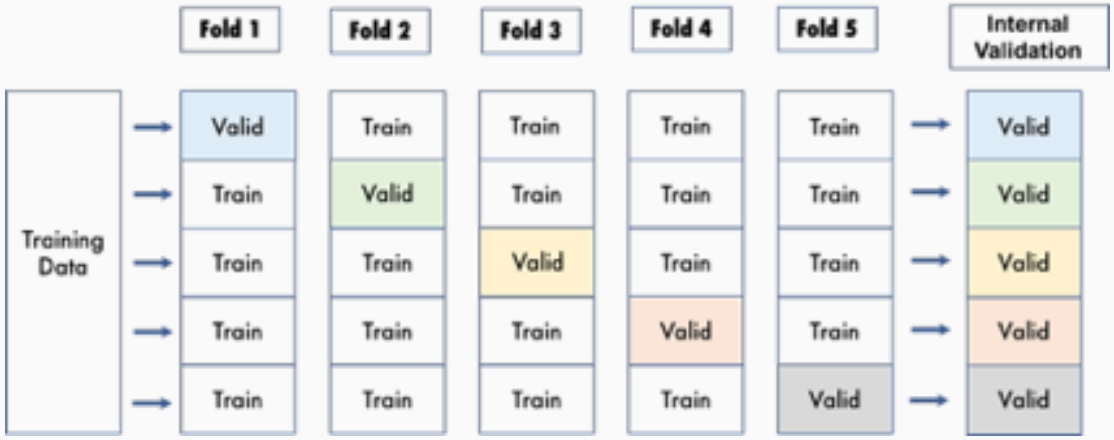
Step 3 - Model / Feature Evolution (Engineering)

In order to validate these selected models even further, model three fold validation is applied. The process is shown in the right.

Model Validation

Driverless AI automatically split the training data to determine the performance of the model parameter tuning and feature engineering steps. For the experiment, Driverless AI randomly split the data into 3 fold cross validation. With cross validation, the whole dataset is utilized by training 3 models where each model is trained on a different subset of the training data.

The visualization below shows how cross validation is utilized to get predictions on hold out data. The visualization shows an example of cross validation with 5 folds. For this experiment, however, 3 folds were created.



Note: The cross-validation process was repeated 3 times to ensure the validation metrics are robust since the training data was small.

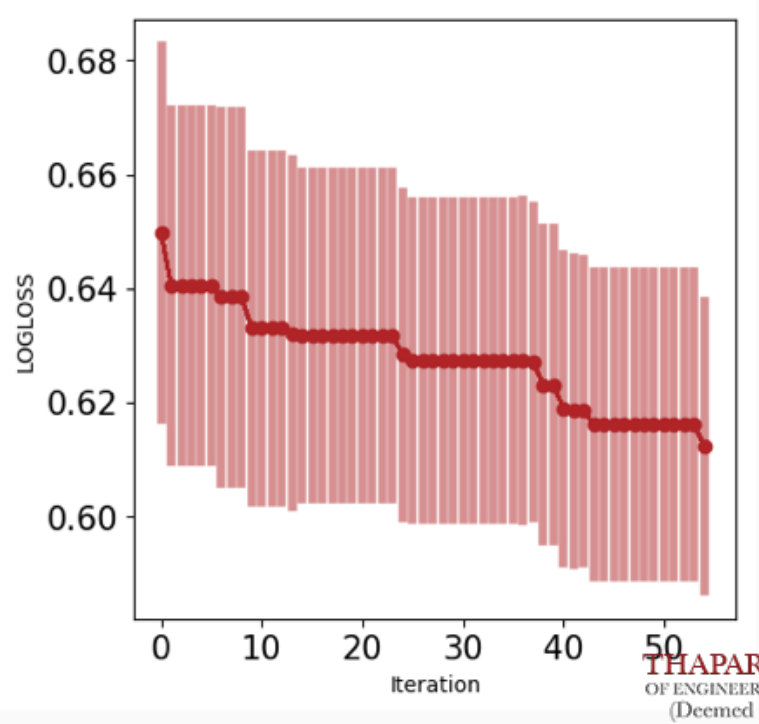
Step 3 - Model / Feature Evolution (Engineering)

As shown in the very first example, Driverless AI uses the genetic algorithm in a very specific way in order to engineer the best features. It is shown in the right.

Feature Evolution

During the Model and Feature Tuning Stage, Driverless AI evaluates the effects of different types of algorithms, algorithm parameters, and features. The goal of the Model and Feature Tuning Stage is to determine the best algorithm and parameters to use during the Feature Evolution Stage. In the Feature Evolution Stage, Driverless AI trained xgboost models (981) where each model evaluated a different set of features. The Feature Evolution Stage uses a genetic algorithm to search the large feature engineering space.

The graph below shows the effect the Model and Feature Tuning Stage and Feature Evolution Stage had on the performance.



Step 3 - Model / Feature Evolution (Engineering)

Using the data of all the (981+199 = 1180) models, Driverless AI presented the most useful features as shown on the right

Feature Importance

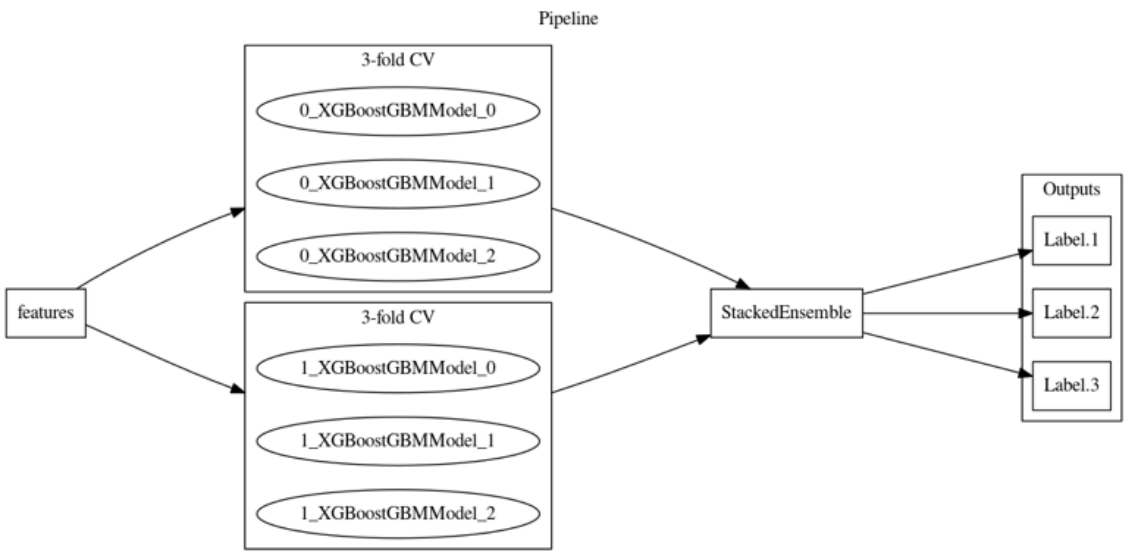
The result of the Feature Evolution Stage is a set of features to use for the final model. The top features used in the final model are shown below, ordered by importance. The features in the table are limited to the top 50, restricted to those with relative importance greater than or equal to 0.003. If no transformer was applied, the feature is an original column.

	Feature	Description	Transformer	Relative Importance
1	6_G	G (Orig)	None	1.0
2	7_H	H (Orig)	None	0.5543
3	5_F	F (Orig)	None	0.4502
4	9_J	J (Orig)	None	0.4375
5	86_I	I (Orig)	None	0.1896
6	13_N	N (Orig)	None	0.1731

Step 4 - Ensembling

Ensembling means, using a group of perfected models to deploy, such that the predictions are done with the maximum accuracy. Driverless used the best models and ensembled them as shown on the right.

For more information about labelEncoder - https://towardsdatascience.com/categorical-encoding-using-label-encoding-and-one-hot-encoder-911ef77fb5bd



Model Index	Type	Model Weight	Num Folds	Fitted features	Target Transformer
0	XGBoostGBMModel	0.9091	3	6	LabelEncoder
1	XGBoostGBMModel	0.0909	3	4	LabelEncoder

Model Index: 0 has a weight of 0.9090909091 in the final ensemble.

Model Index: 1 has a weight of 0.0909090909 in the final ensemble

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Step 5 - Scoring Pipeline and testing

Scoring pipeline was designed in python in order to take in the test dataset values and evaluate the model on the test dataset. The same code will be used to evaluate every reading taken by the spectroscopy sensor, after deployment on the machine.

On the right are the results shown of the testing. On comparison with the Grid Searched RandomForest model, the better evaluation scores are highlighted in green.

Scorer	Better score is	Final ensemble scores on validation data	Final test scores
ACCURACY	higher	0.7434674	0.7993129
AUC	higher	0.8936727	0.9227868
AUCPR	higher	0.789973	0.8391282
F05	higher	0.7434674	0.7823129
F1	higher	0.7434674	0.7823129
F2	higher	0.7438699	0.7823129
GINI	higher	0.7873455	0.8455736
LOGLOSS	lower	0.6121613	0.5582777
MACROAUC	higher	0.7702322	0.9050307
MCC	higher	0.6152012	0.6734694

Evaluation Matrix

Confusion Matrix - Validation

	Predicted: 1	Predicted: 2	Predicted: 3	error
Actual: 1	1	67	1	99%
Actual: 2	5	616	38	7%
Actual: 3	0	124	60	67%

Confusion Matrix - Test

	Predicted: 1	Predicted: 2	Predicted: 3	error
Actual: 1	0	1	0	100%
Actual: 2	0	40	12	23%
Actual: 3	0	19	75	20%

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

~80%

Final Accuracy of Predicting the Ripeness Level of an Apple

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)