Mobile Prices Classification

Jason Shawn D' Souza

December 8, 2019

Abstract

The *Mobile Prices Classification Dataset* contains data that ultimately classifies the **price range** of a given mobile device. The objective was to study the effects of various feature selection techniques on the prediction ability of a model. When testing between 8 different classification models,*SVM Classifier*(Support Vector Machines), *Random Forest Classifiers* and *XGBoost* performed the best **after** feature selection. When implementing the **Greedy** feature selection technique, the **SVM** model recorded a high cross fold accuracy as supported by its confusion matrix. **Tree-based** feature selection also improved the acccuracies, however **Embedded** feature selection techniques performed the weakest.

1 Introduction

The dataset chosen was Mobile Prices Classification. The classification values of the dataset lie in the last column (**price range**). The price range of a given mobile could be

- 0 - Phones at a **low** cost
- 1 - Phones at a **medium** cost (like lower end Huawei,Xiaomi phones)
- 2 - Phones at a **high** cost(like OnePlus and other flagship phones)
- 3 - Phones at a **very high** cost(like Samsung and Apple flagship phones).

The **objectives** of this study are

1. To find the best possible models for the Mobile Prices Classification Dataset
2. To study in detail the effects of **feature selection** on these models .

2 Research

The topic of research chosen is **Feature Selection**.

2.1 Types of Feature Selection Algorithms

There are **three** broad types of feature selection algorithms

2.1.1 Filter methods

Ranking methods (using statistics like **Chi-squared**) to score each feature [1]. **Univariate** nature results in choosing features either **independently** or with **regard to the target value**.

2.1.2 Wrapper methods

Using **objective functions**/**heuristics** to choose the relevant features (figuring out the best combination of features). [1]. An example is the **RFE**(Recursive Feature Elimination).

2.1.3 Embedded Methods

Using the **model** training time itself to learn which features contribute towards a better target label prediction. Embedded methods were introduced to **tackle the computational time** taken by Wrapper methods [1].

2.2 Correlation Matrix

```python
def correlation_matrix(dataset):
    # Develop a correlation between the various features
    correlation_mat = dataset.corr()
    correlation_features = correlation_mat.index
    plt.figure(figsize=(20, 20))
    # Plotting the heatmap
    sns.heatmap(dataset[correlation_features].corr(), annot=True, cmap="RdYlGn")
    plt.show()
```

The snippet of code above is responsible for creating a correlation between the features in the dataset. The output is a heatmap as seen in Figure 1

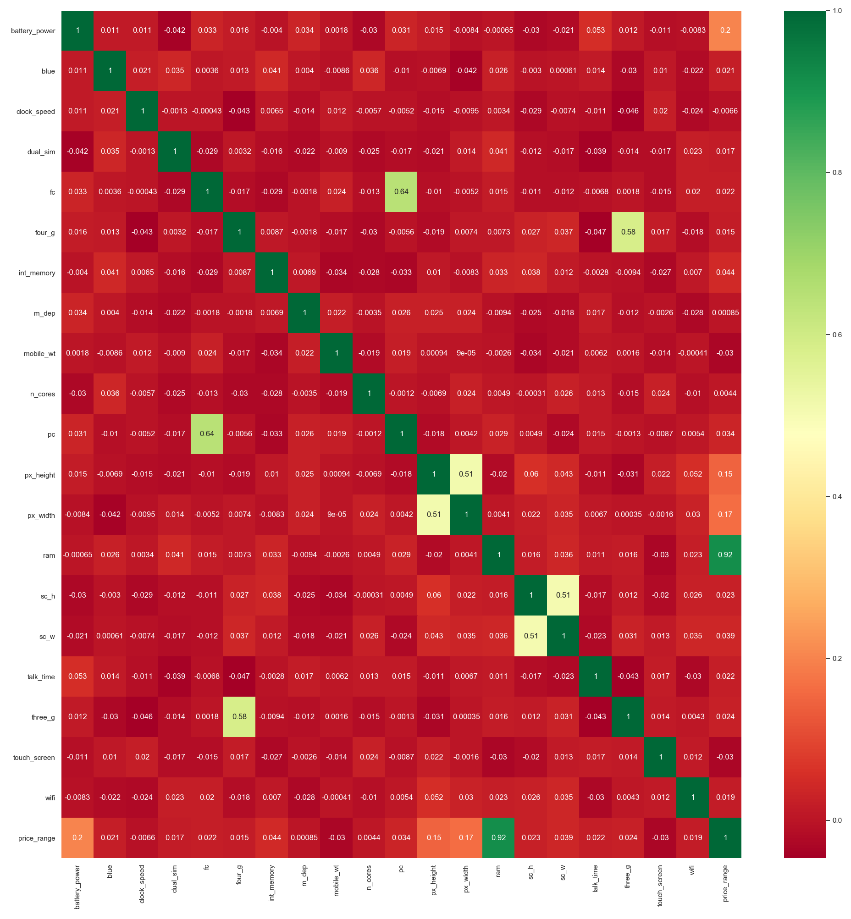


Figure 1: The Correlation matrix

We are interested in the **last** row/column of the heatmap, which shows the relation of the price range with the other features in the dataset. It can be obeserved that the **ram** feature has the greaest relation with the final price range a mobile phone belongs to. The *other* features that have some correlation (to a lesser extent) with the target values are the **battery power**,**pixel width** and **pixel height** in that particular order. The other 16 features in the dataset have a **minor correlation** with the price range

2.3 Feature Importances

```
def feature_importance(dataset, data_train, class_train):
    """
    :param dataset: the pandas instance of the dataset
    :param data_train: the training features of the dataset
    :param class_train: the target labels of the dataset
    :return: A graph showing importances of the features
    """
    # Use a base random forest classifier
    model = return_base_model("Random Forest")
    # fit it with the features and label
    model.fit(data_train, class_train)
    dataset_copy = dataset.copy()
    # Drop the target label
    dataset_copy = dataset_copy.drop(["price_range"], axis=1)
    # Using the copy without the targets
    feature_importances = pd.Series(model.feature_importances_,
    index=dataset_copy.columns)
    feature_importances.nlargest(20).plot(kind='bar')
    plt.show()
```

The code snippet above is responsible for judging the importance of features, This is done by using the *feature importances attribute of the random forest classifier.*The ouput graph as shown in Figure 2.

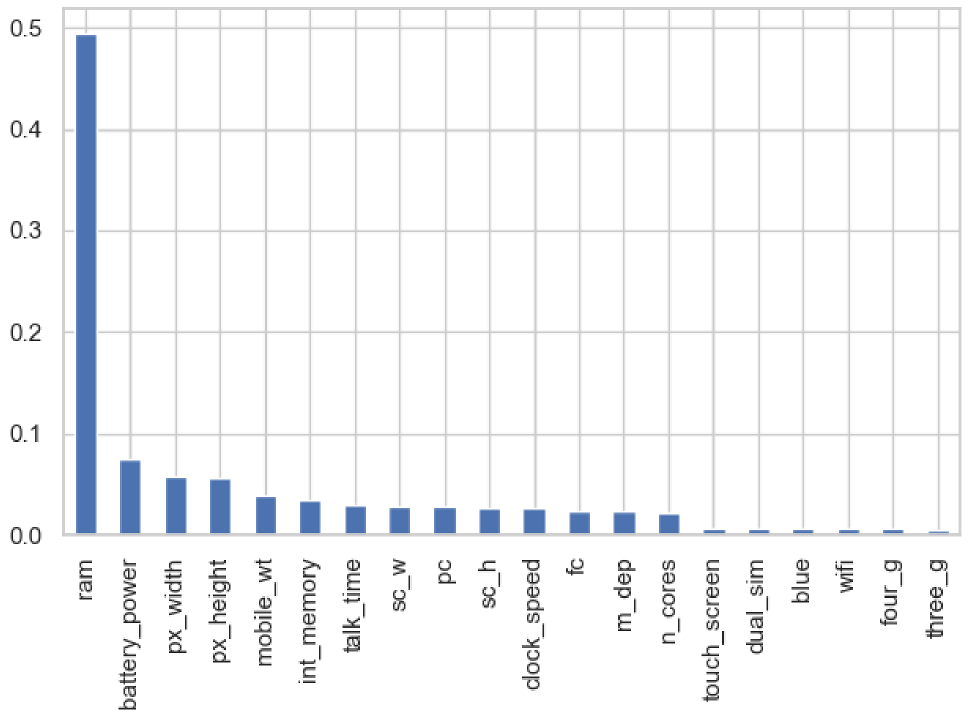


Figure 2: Feature Importances graph

The graph confirms the results of the correlation matrix in Figure 1 that the **ram**, **battery power**, **pixel width** and **pixel height** are the top 4 features in the dataset.

2.4 Uni-variate Feature Selection

This was done using **Select K-best** feature selection tool provided by **sklearn**. This is a form of **Filter** method of Feature Selection (as statistics is used) The **score** function used was **Chi-squared**/Chi2 as this is a **classification** problem.

2.4.1 Chi-squared

As mentioned in Section 3.4, sklearn's MinMax Scaler is used to **avoid negative features**. This is because chi-squared **compares the stats between each non-negative feature and class** [3].
SelectKBest selects the *k* best features and I set the k value as **4** given the results of the previous sections (4 Features with high scores and correlation with the target label of the price range).

2.5 Greedy Feature Selection

To perform Greedy Feature Selection, **RFECV** class from sklearns feature selection package was used. The results **improved** in all three models with optimized hyperparameters. The results will be discussed in greater detail in Section 4(Evaluation). This is a form of **Wrapper** method of Feature Selection as discussed earlier.

2.6 Embedded Feature Selection

As mentioned earlier, using the **embedded algorithms** within a model. This is achieved using **SelectFromModel** class from sklearn's feature selection package. The procedure is similar to that in Greedy Feature Selection wherein, a **mask** of the important features is used and only the **required** features are kept. This is a type of Embedded Method of feature selection(as can be judged from the name).

3 Methodology

The initial baseline models used were as follows:

- **K-Nearest Neighbour** (**KNN**) Classifier
- **Ridge** Classifier

- **XGBoost** Classifier
- **Support Vector Machines** (**SVM**) Classifier
- **Random Forest** Classifier
- **Decision Tree** Classifier
- **Gaussian Naive Bayes** Classifier
- **Stochastic Gradient Descent** (**SGD**) Classifier

Before using the given dataset, the following pre-processing steps *were taken into consideration* :

1. **Dealing with outliers**
2. **Dealing with missing values**
3. **Handling categorical data**
4. **Scaling the data**
5. **Handling Data imbalance**
6. **Feature Selection**
7. **Dimensionality Reduction**

3.1 Dealing with outliers

This was done by using **seaborn** and **matplotlib** to visualize all the *features*(discarding the target label of price range).

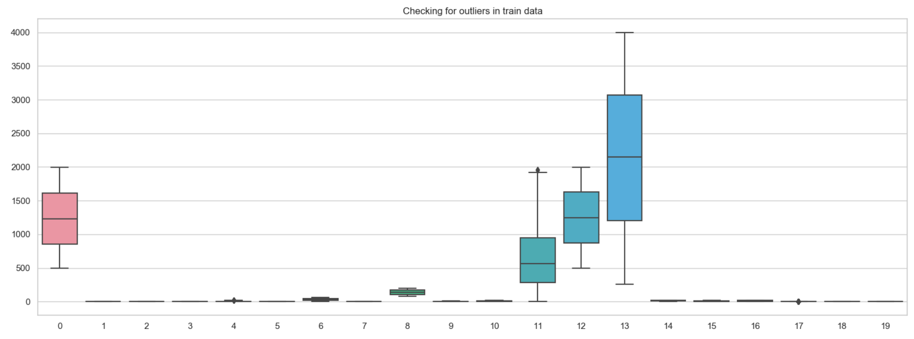
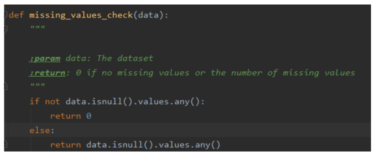


Figure 3: Checking for outliers in train data

As can be seen in the Figure 3 , the only feature that has data points outside the seaborn *whiskers* is in column number 11(**px_height** column,i.e, the column with the screen height)

3.2 Dealing with missing values

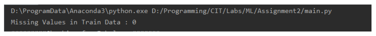
This was tested by using the *missing_values_check* method in the code which basically checks for null values and returns the exact count ***if any***.



```python
def missing_values_check(data):
    """

    :param data: The dataset
    :return: 0 if no missing values or the number of missing values
    """
    if not data.isnull().values.any():
        return 0
    else:
        return data.isnull().values.any()
```

(a) Code that checks for missing values



```
D:\ProgramData\Anaconda3\python.exe D:/Programming/CIT/Labs/ML/Assignment2/main.py
Missing Values in Train Data : 0
```

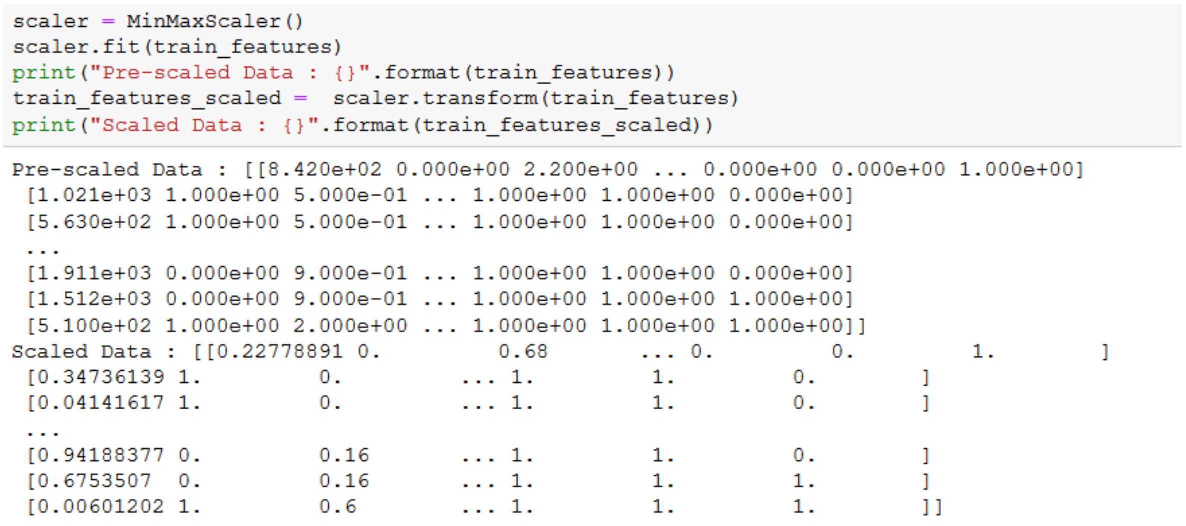
(b) Results

3.3 Dealing with Categorical Data

The dataset **does not** contain any categorical data (just continuous numerical values). Thus, no **encoding** is used.

3.4 Scaling the data

Since the dataset contains numerical values, it made more sense use **MinMaxScaler** (from *scikit-learn*) as opposed to Standard Scaler, which could throw out ***negative values*** which could not be used in **univariate feature selection** (Select K-Best with chi2 **needs** positive numbers). The results of the scaling are as seen in Figure 5 (ran on *ipynb* with same code as in the *main.py* file)

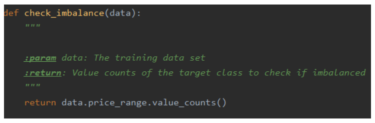


```python
scaler = MinMaxScaler()
scaler.fit(train_features)
print("Pre-scaled Data : {}".format(train_features))
train_features_scaled =  scaler.transform(train_features)
print("Scaled Data : {}".format(train_features_scaled))
```

```
Pre-scaled Data : [[8.420e+02 0.000e+00 2.200e+00 ... 0.000e+00 0.000e+00 1.000e+00]
 [1.021e+03 1.000e+00 5.000e-01 ... 1.000e+00 1.000e+00 0.000e+00]
 [5.630e+02 1.000e+00 5.000e-01 ... 1.000e+00 1.000e+00 0.000e+00]
 ...
 [1.911e+03 0.000e+00 9.000e-01 ... 1.000e+00 1.000e+00 0.000e+00]
 [1.512e+03 0.000e+00 9.000e-01 ... 1.000e+00 1.000e+00 1.000e+00]
 [5.100e+02 1.000e+00 2.000e+00 ... 1.000e+00 1.000e+00 1.000e+00]]
Scaled Data : [[0.22778891 0.         0.68       ... 0.         0.         1.        ]
 [0.34736139 1.         0.         ... 1.         1.         0.        ]
 [0.04141617 1.         0.         ... 1.         1.         0.        ]
 ...
 [0.94188377 0.         0.16       ... 1.         1.         0.        ]
 [0.6753507  0.         0.16       ... 1.         1.         1.        ]
 [0.00601202 1.         0.6        ... 1.         1.         1.        ]]
```

Figure 5: Scaling Data using MinMax Scaler

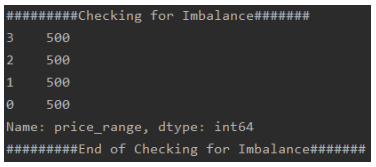
3.5 Handling Data Imbalance



```
def check_imbalance(data):
    """

    :param data: The training data set
    :return: Value counts of the target class to check if imbalanced
    """
    return data.price_range.value_counts()
```

(a) Code that checks for data imbalance



```
#########Checking for Imbalance#######
3    500
2    500
1    500
0    500
Name: price_range, dtype: int64
#########End of Checking for Imbalance#######
```

(b) Results

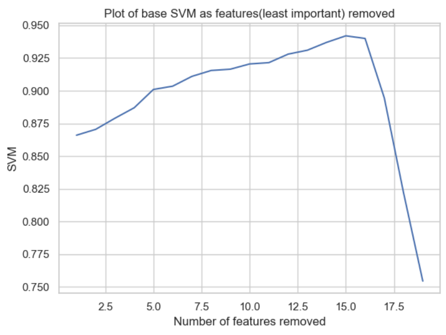
Figure 6a shows the code that checks the count of the target labels (in the column ***price range***. As can be seen in Figure 6b the data contains equal number of points for **each** label (500 for each label/class.)

3.6 Feature Selection

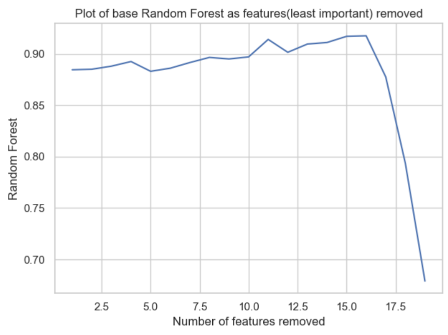
This was done in many steps (basically steps of **Tree Based Feature Selection**):-

1. Build a baseline random forest classifier and fit it with the **train features and class/labels** (obtained **after** splitting the training dataset)
2. Using the *feature_importances_* attribute of the random forest classifier (which returns a numpy array of the features from **best to worst**)
3. Applying *np.argsort* to sort from worst to best.
4. From this *sorted* numpy array, recursively removing each feature (Until 1 feature is left)
5. Each iteration results in *that* many features being removed (i.e, for the first iteration only one feature will be removed, for the second iteration two features will be removed and so on.)

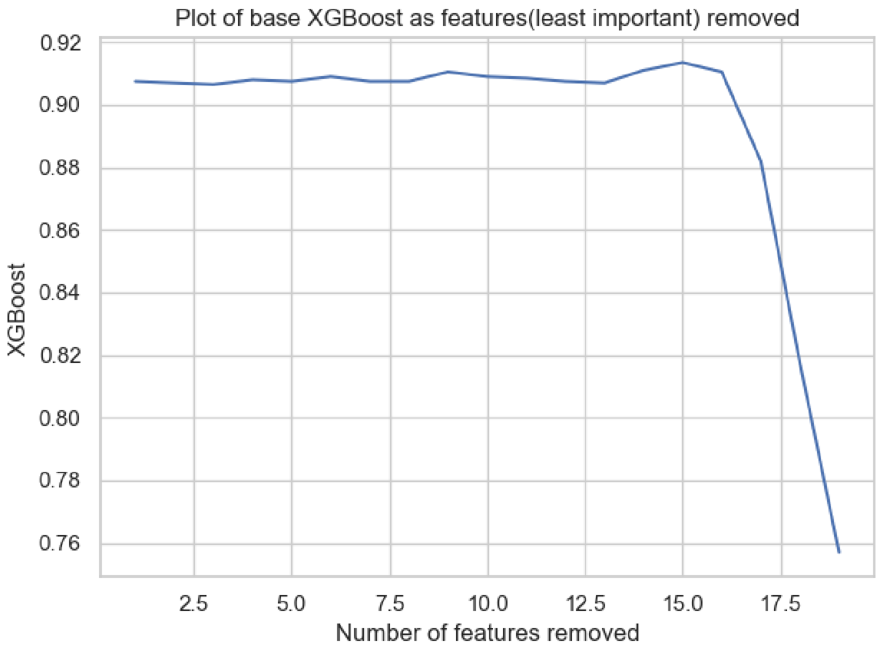
So each **iteration's** accuracy is stored along with the **number of features** removed. This is to find the highest **cross-validation** for each **feature removal**. The functions responsible for this are *feature_removal* and *remove_feature_and_plot*. The results using this recursive feature removal with the baseline models resulted in better accuracy than without it in most cases (XGBoost for example performed better **before** applying the feature removal. Cross fold mean of **0.9012499999999999** without and max accuracy of **0.8600000000000001** with feature removals). However, the three top models of **SVM**,**Random Forest** and **XGBoost** attained their max accuracies when removing **16** out of the 20 features as shown in Figures 7a, 7b and 7c.



(a)



(b)



(c)

Figure 7: Accuracies of SVM, Random Forest and XGBoost after feature removals

Graphs of all other models will be available in Section 6(The Appendix).

3.7 Dimensionality Reduction

As the dataset contains a manageable number of rows **and** features, there is no need to perform dimensionality reduction. (Steps are performed in the *ipython* notebook provided and provide bad results)

3.8 Techniques

The technique of hyper-parameter optimization used was sklearn's **GridSearchCV**. The best performing models (i.e, SVM, Random Forest and XGBoost) were tuned as can be seen in the following subsections. (**Please note** all models that accept the **random_state** param was set to a number of 42 to keep readings and performance uniform and with little to no variation)

3.8.1 SVM

- **C**
  The Regularization parameter, which takes a default float value of **1.0**. The list of params for this **key** (C/Regularization param) were **0.001, 0.01, 0.1, 1, 10**.

- **kernel**
  The type of kernel to be used, takes a default of **rbf**(String). The list of params for this key were **rbf** and **linear**

- **gamma**
  The kernel coeff for rbf, poly and sigmoid. The default value is **auto**. Since we included linear kernel , the inclusion of **auto** in the list of params is justified. The full list of params for this key were **auto, 0.001, 0.01, 0.1, 1** .

- **decision_function_shape** Checks if a One vs Rest (**ovr**) or a One vs One(**ovo**) decision shape should be used (Default is **ovr**). The list of params checked were **ovo, ovr**.

3.8.2 Random Forest

- **n_estimators**
  The number of trees in the random forest, with a default value of **100**. The list of params for this key were **100, 500, 1000, 1500, 2000**.

- **max_depth**
  The maximum depth of the tree, which defaults to **None** (keeps on searching till all nodes are searched or contain less than **min_samples_split samples**. The list of params for this key were **10, 25, 40, 55, 70, 85, 100** [2].

3.8.3 XGBoost

- **min_child_weight**
  Minimum sum of weight needed in a child node, which defaults to a value of **1** [4]. The list of params tested for this key were **1, 5** and **10**.

- **gamma**
  Minimum loss reduction required to make another partition on a leaf node and larger the gamma more conservative the algorithm will be [4]. Default value is 0. The list of params tested for this key were **0.5, 1, 1.5, 2** and **5**.

- **nthread**
  Number of parallel threads required to run the algorithm. Default is **maximum** number of threads if not set. This key was tested with **4** threads

- **subsample**
  This is the subsample ratio of the training instances [4]. This parameter tends to **prevent overfitting** by sampling the specified ratio of training data **before** growing the trees. The values lie between (and include) 0 and 1 with 1 being the default value. The list of params tested for this key were **0.6, 0.8** and **1.0**.

- **colsample_bytree**
  Similar to subsample, this parameter involves subsamples of columns. Just like the subsample param, its default value is also 1. The list of params tested for this key were **0.6, 0.8** and **1.0**

- **max_depth**
  As mentioned in Section 3.8.2, maximum depth of tree. The default value is set to 6. The list of params tested for this key were **3, 4** and **5**

4 Evaluation

NOTE: All program logs can be found in Section 6.1 of the Appendix. All Data in this section is taken from the logs (obtained from the script).

4.1 Base models

Without Feature Selection (*Rounded to 5 Digits)		
Model	Cross Fold Accuracy (**Mean**)	Cross Fold Accuracy (**Standard Deviation**)
KNN	0.40050	0.02115
Random Forest	0.87850	0.01950
SVM	0.86400	0.02634
Ridge	0.60150	0.02169
SGD	0.762	0.01900
Decision Tree	0.83450	0.02593
Gaussain NB	0.812	0.01487
XGBoost	0.90900	0.02010

With Feature Selection (Tree-based Approach)	
Model	Max. Crossfold Accuracy (**Mean**)
KNN	0.8975
Random Forest	0.91750
SVM	0.942
Ridge	0.6015
SGD	0.7665
Decision Tree	0.869
Gaussain NB	0.81500
XGBoost	0.9135

4.2 Hyper-parameter optimization

4.2.1 SVM

Best Parameters with SVM:
C=10, decision_function_shape='ovo', gamma='auto',kernel='linear' with a score of **0.9630** (after rounding off)

4.2.2 Random Forest

Best Parameters with Random Forest:
max_depth=25, n_estimators=1500 with a score of **0.8915**

4.2.3 XGBoost

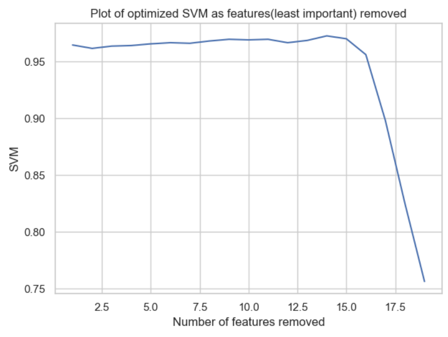
Best Parameters with XGBoost:
colsample_bytree=1.0, gamma=0.5, max_depth=4, min_child_weight=1, nthread=4, subsample=0.6 with a score of **0.9235**

4.3 Models with optimized hyper-parameter

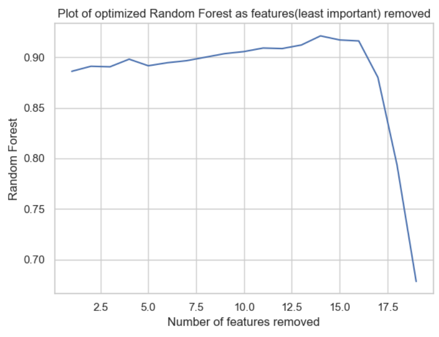
4.3.1 Tree-Based Feature Selection

Optimized Hyper-parameters with Feature Selection (Tree-based Approach)	
Model	Max. Crossfold Accuracy (**Mean**)
SVM	0.97250
Random Forest	0.9205
XGBoost	0.92800

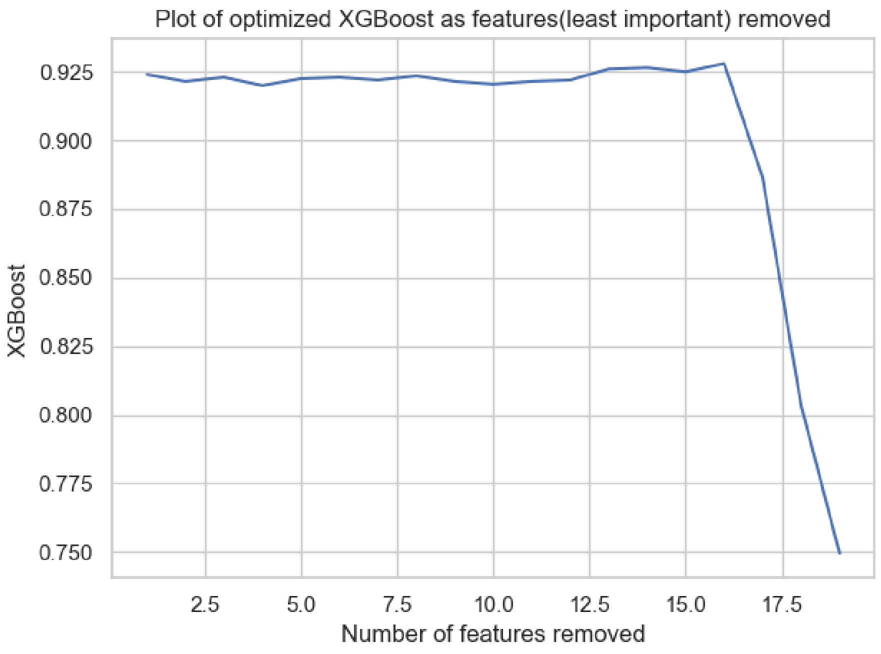
The graphs produced from this tree based approach for each model are shown in Figures 8a, 8b, 8c



(a)



(b)



(c)

Figure 8: Accuracies of optimized SVM, Random Forest and XGBoost after feature removals

As can be observed from the graphs, the max. accuracy is reached when removing **sixteen** features and keeping **four** features (As mentioned in Section 2.2, features **ram**, **battery power**, **px height** and **px width**)

4.3.2 Univariate Feature Selection

Optimized Hyper-parameters Univariate Feature Selection	
Model	Max. Crossfold Accuracy (**Mean**)
SVM	0.9630
Random Forest	0.8915
XGBoost	0.9195

The mean of crossfold accuracies is slightly less in this case (compared to tree-based feature removal/search) of all three models. To gain further insights we look into the confusion matrix and classification report of each model
NOTE: All Incorrect predictions in the confusion matrix are marked in red (These are the false positive or false negative values).

SVM Univariate Feature Selection Evaaluation

		Predicted			
		0	1	2	3
Actual	0	490	10	0	0
	1	14	473	13	0
	2	0	13	472	15
	3	0	0	9	491

As can be observed from the confusion matrix, the classification for price ranges 0 and 3 are better than those of 1 and 2 (only few price ranges of 0 were classified as 1 and 3 classified as 2). However, there were more instances that belonged to class 1 but were classified by the model as 0 or 2 (and instances that belonged to class 2 but predicted as 1 or 3). This is proven in the Table 1 given below (higher precision, accuracy and f1-scores for classes 0 and 3 than for classes 1 and 2).

Table 1: SVM Univariate Classification report

	precision	recall	f1-score	support
0	0.97	0.98	0.98	500
1	0.95	0.95	0.95	500
2	0.96	0.94	0.95	500
3	0.97	0.98	0.98	500
accuracy			0.96	2000
macro avg	0.96	0.96	0.96	2000
macro avg	0.96	0.96	0.96	2000

Random Forest Univariate Feature Selection Evaaluation

		Predicted			
		0	1	2	3
Actual	0	473	27	0	0
	1	37	424	39	0
	2	0	48	423	29
	3	0	0	37	463

Observations are similar to those seen in the case of SVM's Confusion Matrix (with slightly less correct predictions). This can be confirmed in Table 2 given below. (Better classification for classes 0 and 3 than 1 and 2).

Table 2: Random Forest Univariate Classification report

	precision	recall	f1-score	support
0	0.93	0.95	0.94	500
1	0.85	0.85	0.85	500
2	0.85	0.85	0.85	500
3	0.94	0.93	0.93	500
accuracy			0.89	2000
macro avg	0.89	0.89	0.89	2000
macro avg	0.89	0.89	0.89	2000

XGBoost Univariate Feature Selection Evaaluation

		Predicted			
		0	1	2	3
Actual	0	482	18	0	0
	1	21	456	23	0
	2	0	35	439	26
	3	0	0	30	470

Similar to the previous models, slightly better classification for classes 0 and 3 when compared to 1 and 2 (However performs better than random Forests). This can be confrimed by the Classification Report seen in Table 3

Table 3: XGBoost Univariate Classification report

	precision	recall	f1-score	support
0	0.96	0.96	0.96	500
1	0.90	0.91	0.90	500
2	0.89	0.88	0.89	500
3	0.95	0.94	0.94	500
accuracy			0.92	2000
macro avg	0.92	0.92	0.92	2000
macro avg	0.92	0.92	0.92	2000

Univariate Feature Selection Inference

Overall, the SVM model outperforms both XGBoost and Random Forests. The ranking of effectiveness of prediction *for this dataset* using univariate feature selection would be

1. SVM
2. XGBoost
3. Random Forest

4.3.3 Greedy Feature Selection

Optimized Hyper-parameters Greedy Feature Selection (*Rounded to 4 Digits)		
Model	Number of features selected	Cross Fold Accuracy (**Mean**)
SVM	7	0.9740
Random Forest	6	0.9205
XGBoost	5	0.9280

SVM Greedy Feature Selection Evaluation

		Predicted			
		0	1	2	3
Actual	0	494	6	0	0
	1	7	484	9	0
	2	0	13	476	11
	3	0	0	6	494

Greedy Feature Selection for SVM performs better and reports better predictions than **Univariate** feature selection. This is when **seven** features are chosen. Furthermore, this method of feature selection *slightly* outperforms the **Tree based** feature selection (mean average of **0.9740** (approx.) in the case of greedy search, **0.9725** (approx.) in the case of tree- based feature selection). The Classification Table

Table 4: SVM Greedy Feature Selection Classification report

	precision	recall	f1-score	support
0	0.99	0.99	0.99	500
1	0.96	0.97	0.97	500
2	0.97	0.95	0.96	500
3	0.98	0.99	0.98	500
accuracy			0.97	2000
macro avg	0.97	0.97	0.97	2000
macro avg	0.97	0.97	0.97	2000

Random Forest Greedy Feature Selection Evaluation

		Predicted			
		0	1	2	3
Actual	0	479	21	0	0
	1	28	450	22	0
	2	0	30	446	24
	3	0	0	34	466

Accuracies for Random Forest *increases* using this technique when compared to univariate feature selection. This is when **six** features are chosen. It should be noted that accuracies obtained in this method and using tree based approach are the **same**(**both before and after rounding off**).

Table 5: Random Forest Greedy Feature Selection Classification report

	precision	recall	f1-score	support
0	0.94	0.96	0.95	500
1	0.90	0.90	0.90	500
2	0.89	0.89	0.89	500
3	0.95	0.93	0.94	500
accuracy			0.92	2000
macro avg	0.92	0.92	0.92	2000
macro avg	0.92	0.92	0.92	2000

XGBoost Greedy Feature Selection Evaluation

		Predicted			
		0	1	2	3
Actual	0	483	17	0	0
	1	18	465	17	0
	2	0	32	441	27
	3	0	0	33	467

Accuracies XBoost *increases* using this technique when compared to univariate feature selection, just like in the case of Random Forests. These are the results when **five** features are chosen. It should be noted that accuracies obtained in this method and using tree based approach are the **same**(**both before and after rounding off**).

Table 6: XGBoost Greedy Feature Selection Classification report

	precision	recall	f1-score	support
0	0.96	0.97	0.97	500
1	0.90	0.93	0.92	500
2	0.90	0.88	0.89	500
3	0.95	0.93	0.94	500
accuracy			0.93	2000
macro avg	0.93	0.93	0.93	2000
macro avg	0.93	0.93	0.93	2000

Greedy Feature Selection Inference

Overall, in this method the SVM model outperforms both XGBoost and Random Forests and reports its highest accuracy signalling that this method (greedy feature selection) is a great technique in the given scenario. The ranking of effectiveness of prediction *for this dataset* using greedy feature selection would be

1. SVM
2. XGBoost
3. Random Forest

4.3.4 Embedded Feature Selection

Optimized Hyper-parameters Embedded Feature Selection (*Rounded to 4 Digits)		
Model	Number of features selected	Cross Fold Accuracy (**Mean**)
SVM	4	0.9560
Random Forest	4	0.917
XGBoost	4	0.9260

SVM Embedded Feature Selection Evaluation

		Predicted			
		0	1	2	3
Actual	0	487	13	0	0
	1	10	480	10	0
	2	0	16	464	20
	3	0	0	19	481

The accuracies obtained in this method are the **least** of the three feature selection techniques implemented (accuracies also lesser than the hyper-parameter optimzed SVM model). The number of features chosen were **four**.

Table 7: SVM Embedded Feature Selection Classification report

	precision	recall	f1-score	support
0	0.98	0.97	0.98	500
1	0.94	0.96	0.95	500
2	0.94	0.93	0.93	500
3	0.96	0.96	0.96	500
accuracy			0.96	2000
macro avg	0.96	0.96	0.96	2000
macro avg	0.96	0.96	0.96	2000

Random Forest Embedded Feature Selection

		Predicted			
		0	1	2	3
Actual	0	481	19	0	0
	1	24	455	21	0
	2	0	28	437	35
	3	0	0	39	461

The accuracy obtained for Random Forest is also the least out of the three feature selections implemented. The number of features chosen were also **four**. It should be noted that the accuracy reported (0.9205) is **higher** than the hyper-parameter optimized Random Forest Model(0.8915)

Table 8: Random Forest Embedded Feature Selection Classification report

	precision	recall	f1-score	support
0	0.95	0.96	0.96	500
1	0.91	0.91	0.91	500
2	0.88	0.87	0.88	500
3	0.93	0.92	0.93	500
accuracy			0.92	2000
macro avg	0.92	0.92	0.92	2000
macro avg	0.92	0.92	0.92	2000

XGBoost Embedded Feature Selection

		Predicted			
		0	1	2	3
Actual	0	487	13	0	0
	1	18	465	17	0
	2	0	32	441	27
	3	0	0	33	467

The accuracy obtained for XGBoost is also the least out of the three feature selections implemented (accuracies also lesser than the hyper-parameter optimzed XGBoost model). The number of features chosen were also **four**.

Table 9: XGBoost Embedded Feature Selection Classification report

	precision	recall	f1-score	support
0	0.96	0.97	0.97	500
1	0.90	0.93	0.92	500
2	0.90	0.88	0.89	500
3	0.95	0.93	0.94	500
accuracy			0.93	2000
macro avg	0.93	0.93	0.93	2000
macro avg	0.93	0.93	0.93	2000

Embedded Feature Selection Inference

Overall, in this method the SVM model outperforms both XGBoost and Random Forest again. Although, the general effectiveness and accuracies of **SVM** and **XGBoost** are lowwer (Random Forest still reports a greater accuracy with this method than with the hyper-parameter optimized random forest)

5 Conclusion

Out of the 8 base models tested, **SVM**, **Random Forests** and **XGBoost** were the top three models. In order the best models for the **mobile price classification** dataset were SVM, XGBoost and Random Forest(with respect to the mean cross fold accuracies) These models after tuning and finding the best given hyper-parameters reported higher **cross-fold** accuracies. These were then improved using various **feature selection** techniques and reported an increase in accuracy as seen in Section 4 (**some** exceptions like SVM with Embedded Feature Selection). A main disadvantage of cross-fold validation is observed in the experiments (higher cross-fold validation accuracies). This is because cross-fold validation uses the same data to tune the parameters of the model and then **evaluate** them, creating a *bias*. Also, feature selection to an extent also biases the data (performing feature selection on all of the data and then applying cross-validation. Since the test data in each fold of the cross-validation **was also used** to choose the features, this biases the results to an extent.)
Overall, **Feature** Selection is an important pre-processing step and can be useful in the following ways :-

- Improving the accuracy (as observed in Section 4).
- Visualization of the features importance with respect to the target label/-class.
- Understanding of the models built and the features that matter to the target label/class.

Future Works that could be undertaken with this dataset and premise include implementing **Nested Cross-Fold Validations** or **Stratified K Fold Validations**.

References

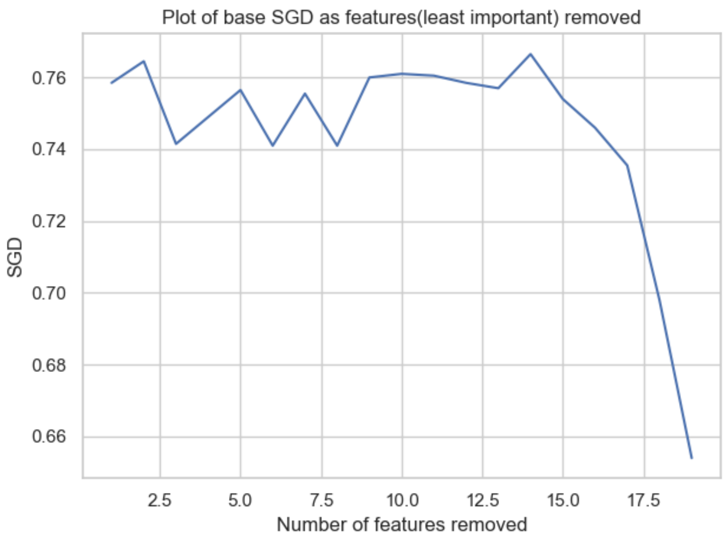
[1] Girish Chandrashekar and Ferat Sahin. “A survey on feature selection methods”. In: *Computers & Electrical Engineering* 40.1 (2014), pp. 16–28.

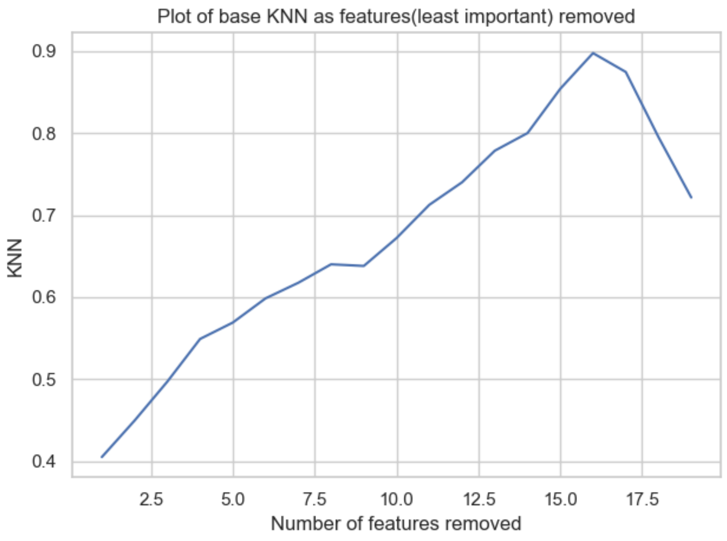
[2] *SKLearn - Random Forest Classifier*. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. Accessed: 2019-12-08.

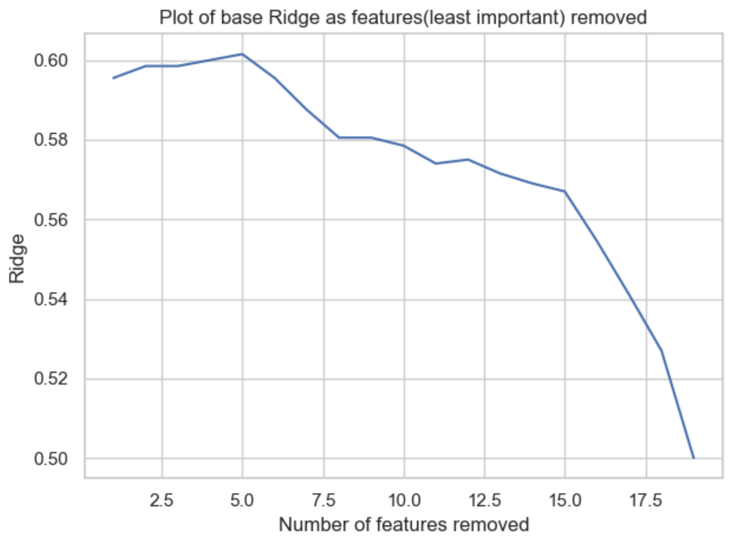
[3] *sklearn.feature_selection.chi2*. https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.chi2.html. Accessed: 2019-12-07.

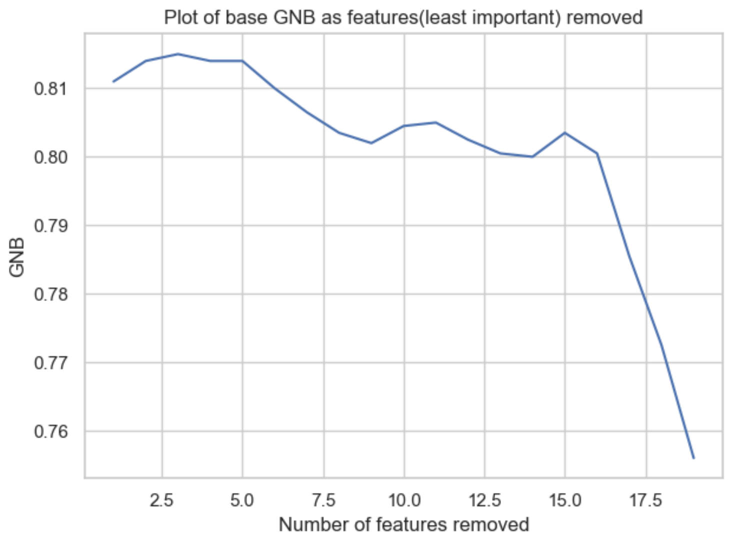
[4] *XGBoost Parameters*. https://xgboost.readthedocs.io/en/latest/parameter.html. Accessed: 2019-12-08.

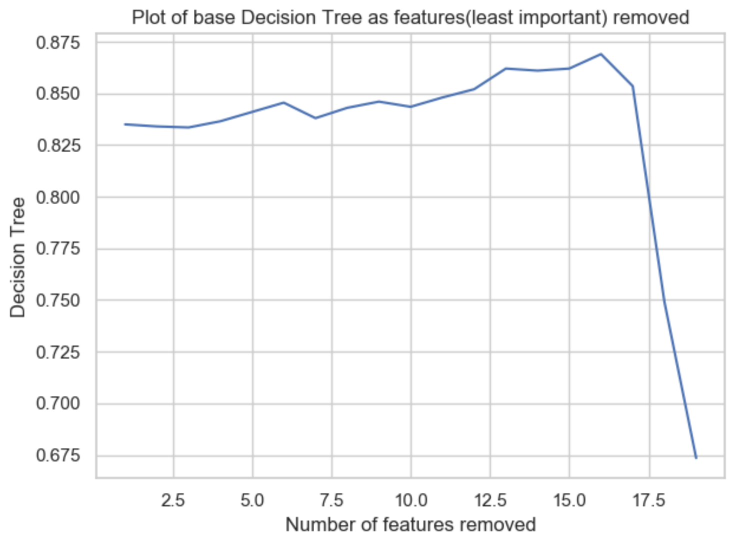
6 Appendix


Plot of base SGD as features(least important) removed
SGD
0.76
0.74
0.72
0.70
0.68
0.66
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Number of features removed


Plot of base KNN as features(least important) removed
KNN
0.9
0.8
0.7
0.6
0.5
0.4
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Number of features removed


Plot of base Ridge as features(least important) removed
Ridge
0.60
0.58
0.56
0.54
0.52
0.50
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Number of features removed


Plot of base GNB as features(least important) removed
GNB
0.81
0.80
0.79
0.78
0.77
0.76
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Number of features removed



6.1 PROGRAM LOGS/OUTPUT

ALL Program output is listed below((taken from log.txt, refer to run() function of python script for details) :

```
Missing Values in Train Data : 0
#########Checking for Imbalance#######
3     500
2     500
1     500
0     500
Name: price_range, dtype: int64
#########End of Checking for Imbalance#######
########## KNN #########

KNN - Cross Val Mean : 0.4005000000000001 , Cross Val Std Dev : 0.02114828598255

########## END OF KNN #########

########## Random Forest #########

Random Forest - Cross Val Mean : 0.8784999999999998 , Cross Val Std Dev : 0.0195

########## END OF Random Forest #########

########## SVM #########

SVM - Cross Val Mean : 0.8640000000000001 , Cross Val Std Dev : 0.02634387974463
```

```
########## END OF SVM #########

########## Ridge #########

Ridge – Cross Val Mean : 0.6014999999999999 , Cross Val Std Dev : 0.021685248442

########## END OF Ridge #########

########## SGD #########

SGD – Cross Val Mean : 0.762 , Cross Val Std Dev : 0.019000000000000017

########## END OF SGD #########

########## Decision Tree #########

Decision Tree – Cross Val Mean : 0.8344999999999999 , Cross Val Std Dev : 0.0259

########## END OF Decision Tree #########

########## GNB #########

GNB – Cross Val Mean : 0.812 , Cross Val Std Dev : 0.014866068747318469

########## END OF GNB #########

########## XGBoost #########

XGBoost – Cross Val Mean : 0.9090000000000001 , Cross Val Std Dev : 0.0200997512

########## END OF XGBoost #########

[17  5 19  1  3 18  9  7  4  2 14 10 15 16  6  8 11 12  0 13]
KNN Max Accuracy : 0.8975
Random Forest Max Accuracy : 0.9175000000000001
SVM Max Accuracy : 0.942
Ridge Max Accuracy : 0.6015
SGD Max Accuracy : 0.7665
Decision Tree Max Accuracy : 0.869
GNB Max Accuracy : 0.8150000000000001
XGBoost Max Accuracy : 0.9135
Models ranked from best to worst : [('SVM', 0.942), ('Random Forest', 0.91750000
Best models : ['SVM', 'Random Forest', 'XGBoost']
$$$$$$$$$$$$$$$$$$$$$ BEGINNING SVM Hyperparameter optimization $$$$$$$$$$$$$$$
Best Parameters with SVM:
```

```
{'C': 10, 'decision_function_shape': 'ovo', 'gamma': 'auto', 'kernel': 'linear'}
0.9629999999999999
Finished running GridSearch on SVM in 21.32828402519226 seconds
SVM Univariate Mean scores : 0.9629999999999999
SVM Univariate Classification Report :
              precision    recall  f1-score   support

         0.0       0.97      0.98      0.98       500
         1.0       0.95      0.95      0.95       500
         2.0       0.96      0.94      0.95       500
         3.0       0.97      0.98      0.98       500

    accuracy                           0.96      2000
   macro avg       0.96      0.96      0.96      2000
weighted avg       0.96      0.96      0.96      2000

SVM Univariate Confusion Matrix :
 [[490  10   0   0]
 [ 14 473  13   0]
 [  0  13 472  15]
 [  0   0   9 491]]
End of SVM univariate feature selection with optimized hyperparameters
The optimized model needs to select 4 features for embedded feature selection

Result after Embedded Feature Selection for SVM :  0.9559999999999998
SVM Embedded Feature Selection Confusion Matrix :
 [[487  13   0   0]
 [ 10 480  10   0]
 [  0  16 464  20]
 [  0   0  19 481]]
SVM Embedded Feature Selection Classification Report :
              precision    recall  f1-score   support

         0.0       0.98      0.97      0.98       500
         1.0       0.94      0.96      0.95       500
         2.0       0.94      0.93      0.93       500
         3.0       0.96      0.96      0.96       500

    accuracy                           0.96      2000
   macro avg       0.96      0.96      0.96      2000
weighted avg       0.96      0.96      0.96      2000

$$$$$$$$$$$$$$$$$$$$$ BEGINNING Random Forest Hyperparameter optimization $$$$$$
Best Parameters with Random Forest:
{'max_depth': 25, 'n_estimators': 1500} with a score of  0.8915
Finished running GridSearch on Random Forest in 157.0145778656006 seconds
```

```
Random Forest Univariate Mean scores : 0.8915
Random Forest Univariate Classification Report :
              precision    recall  f1-score   support

         0.0       0.93      0.95      0.94       500
         1.0       0.85      0.85      0.85       500
         2.0       0.85      0.85      0.85       500
         3.0       0.94      0.93      0.93       500

    accuracy                           0.89      2000
   macro avg       0.89      0.89      0.89      2000
weighted avg       0.89      0.89      0.89      2000

Random Forest Univariate Confusion Matrix :
 [[473  27   0   0]
 [ 37 424  39   0]
 [  0  48 423  29]
 [  0   0  37 463]]
End of Random Forest univariate feature selection with optimized hyperparameters
The optimized model needs to select 4 features for embedded feature selection

 Result after Embedded Feature Selection for Random Forest :
0.917
Random Forest Embedded Feature Selection Confusion Matrix :
 [[481  19   0   0]
 [ 24 455  21   0]
 [  0  28 437  35]
 [  0   0  39 461]]
Random Forest Embedded Feature Selection Classification Report :
              precision    recall  f1-score   support

         0.0       0.95      0.96      0.96       500
         1.0       0.91      0.91      0.91       500
         2.0       0.88      0.87      0.88       500
         3.0       0.93      0.92      0.93       500

    accuracy                           0.92      2000
   macro avg       0.92      0.92      0.92      2000
weighted avg       0.92      0.92      0.92      2000

$$$$$$$$$$$$$$$$$$$$$ BEGINNING XGBoost Hyperparameter optimization $$$$$$$$$$$
Best Parameters with XGBoost:
{'colsample_bytree': 1.0, 'gamma': 0.5, 'max_depth': 4, 'min_child_weight': 1, '
0.9235
Finished running GridSearch on XGBoost in 560.235021352768 seconds
XGBoost Univariate Mean scores : 0.9235
XGBoost Univariate Classification Report :
              precision    recall  f1-score   support

         0.0       0.96      0.96      0.96       500
         1.0       0.90      0.91      0.90       500
         2.0       0.89      0.88      0.89       500
         3.0       0.95      0.94      0.94       500

    accuracy                           0.92      2000
   macro avg       0.92      0.92      0.92      2000
weighted avg       0.92      0.92      0.92      2000

XGBoost Univariate Confusion Matrix :
 [[482  18   0   0]
 [ 21 456  23   0]
 [  0  35 439  26]
 [  0   0  30 470]]
End of XGBoost univariate feature selection with optimized hyperparameters
The optimized model needs to select 4 features for embedded feature selection

Result after Embedded Feature Selection for XGBoost :  0.9279999999999999
XGBoost Embedded Feature Selection Confusion Matrix :
 [[483  17   0   0]
 [ 18 465  17   0]
 [  0  32 441  27]
 [  0   0  33 467]]
XGBoost Embedded Feature Selection Classification Report :
              precision    recall  f1-score   support

         0.0       0.96      0.97      0.97       500
         1.0       0.90      0.93      0.92       500
         2.0       0.90      0.88      0.89       500
         3.0       0.95      0.93      0.94       500

    accuracy                           0.93      2000
   macro avg       0.93      0.93      0.93      2000
weighted avg       0.93      0.93      0.93      2000

Argsort of features : [17 19  5  1  3 18  9  7  4 14  2 15 10 16
6  8 11 12  0 13]
SVM optimized hyperparameter Max Accuracy : 0.9724999999999999
Random Forest optimized hyperparameter Max Accuracy : 0.9205
XGBoost optimized hyperparameter Max Accuracy : 0.9279999999999999
Best number of features for greedy search: 7

Result after Greedy/RFECV for SVM :  0.9739999999999999
```

```
SVM Greedy/RFECV Confusion Matrix :
 [[494   6   0   0]
 [  7 484   9   0]
 [  0  13 476  11]
 [  0   0   6 494]]
SVM Greedy/RFECV Classification Report :
              precision    recall  f1-score   support

         0.0       0.99      0.99      0.99       500
         1.0       0.96      0.97      0.97       500
         2.0       0.97      0.95      0.96       500
         3.0       0.98      0.99      0.98       500

    accuracy                           0.97      2000
   macro avg       0.97      0.97      0.97      2000
weighted avg       0.97      0.97      0.97      2000

################# END OF GREEDY FEATURE SELECTION
Best number of features for greedy search: 6

Result after Greedy/RFECV for Random Forest :  0.9205
Random Forest Greedy/RFECV Confusion Matrix :
 [[479  21   0   0]
 [ 28 450  22   0]
 [  0  30 446  24]
 [  0   0  34 466]]
Random Forest Greedy/RFECV Classification Report :
              precision    recall  f1-score   support

         0.0       0.94      0.96      0.95       500
         1.0       0.90      0.90      0.90       500
         2.0       0.89      0.89      0.89       500
         3.0       0.95      0.93      0.94       500

    accuracy                           0.92      2000
   macro avg       0.92      0.92      0.92      2000
weighted avg       0.92      0.92      0.92      2000

################# END OF GREEDY FEATURE SELECTION
Best number of features for greedy search: 4

Result after Greedy/RFECV for XGBoost :  0.9279999999999999
XGBoost Greedy/RFECV Confusion Matrix :
 [[483  17   0   0]
 [ 18 465  17   0]
 [  0  32 441  27]
 [   0    0   33  467]]
XGBoost Greedy/RFECV Classification Report :
              precision    recall  f1-score   support

         0.0       0.96      0.97      0.97       500
         1.0       0.90      0.93      0.92       500
         2.0       0.90      0.88      0.89       500
         3.0       0.95      0.93      0.94       500

    accuracy                           0.93      2000
   macro avg       0.93      0.93      0.93      2000
weighted avg       0.93      0.93      0.93      2000

################## END OF GREEDY FEATURE SELECTION
```