Jasmeen Kaur
BDS-HW5

# Feature Selection

---

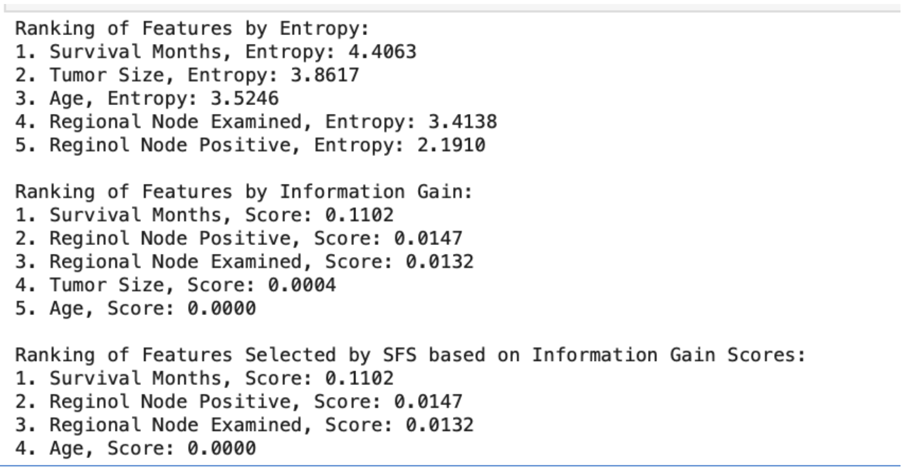
**For feature engineering and preprocessing, I used the following steps:**

1. **Handling Missing Values:** The first step in the preprocessing involves addressing missing data within the dataset. For numerical columns, missing values are replaced with the median of each column, which is less sensitive to outliers compared to the mean. For categorical columns, the most frequent value (mode) is used as the imputation strategy. Even though the Breast Cancer dataset doesn't have a lot of missing values, this step ensures that the dataset is complete, helping prevent errors during model training and improve model accuracy.
2. **Outlier Detection and Handling:** The code identifies outliers in the numerical data based on the Interquartile Range (IQR) method. Outliers can skew the results of many algorithms by pulling the model's focus away from the central trend of the data. By calculating the first and third quartiles (Q1 and Q3), and defining outliers as those values that lie beyond 1.5 times the IQR from these quartiles, the method systematically removes these outliers from the dataset.
3. **Standardization of Features:** To ensure that all numerical features contribute equally to the analysis, standardization is applied using StandardScaler() function from the sklearn.preprocessing library, which transforms the data to have zero mean and unit variance. This normalization step is crucial, especially for models that rely on distance calculations(such as KNN) or regularization, as it prevents features with larger scales from dominating the learning process.
4. **Dimensionality Reduction with PCA:** Finally, Principal Component Analysis (PCA) is applied to reduce the dimensionality of the data while retaining 95% of the variance. Even though there wasn't a large number of features in the Breast Cancer dataset, this step reduces the computational complexity and potential overfitting by summarizing several correlated features into a smaller set of uncorrelated components, thus simplifying the model's structure without significant loss of information.

For Feature Selection/Feature Ranking I used Entropy, Information Gain and Sequential Forward Selection techniques.

Finally, the **Status** column(Alive or Dead) was labeled as the target value to be predicted.

**Results from all the models and the feature selection/ranking in the form of a table**



```
Ranking of Features by Entropy:
1. Survival Months, Entropy: 4.4063
2. Tumor Size, Entropy: 3.8617
3. Age, Entropy: 3.5246
4. Regional Node Examined, Entropy: 3.4138
5. Reginol Node Positive, Entropy: 2.1910

Ranking of Features by Information Gain:
1. Survival Months, Score: 0.1102
2. Reginol Node Positive, Score: 0.0147
3. Regional Node Examined, Score: 0.0132
4. Tumor Size, Score: 0.0004
5. Age, Score: 0.0000

Ranking of Features Selected by SFS based on Information Gain Scores:
1. Survival Months, Score: 0.1102
2. Reginol Node Positive, Score: 0.0147
3. Regional Node Examined, Score: 0.0132
4. Age, Score: 0.0000
```

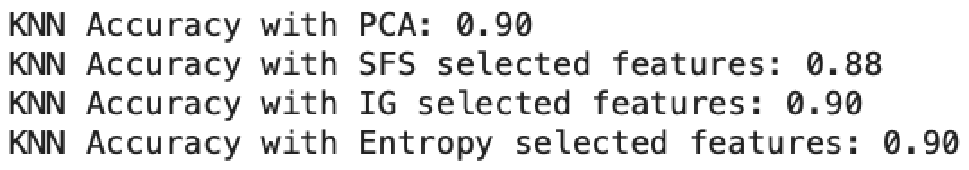
1. **KNN** is a simple, instance-based learning algorithm where the class of a sample is determined by the majority class among its k nearest neighbors.

   **Pros**: Easy to understand and simple to implement

   **Cons**: Computationally expensive as it requires distance computation between each query instance and all training samples

   **Main Hyperparameters**:

   - **'k'**: number of nearest neighbors to select for final classification
   - **'Distance metric'** such as Euclidean etc.



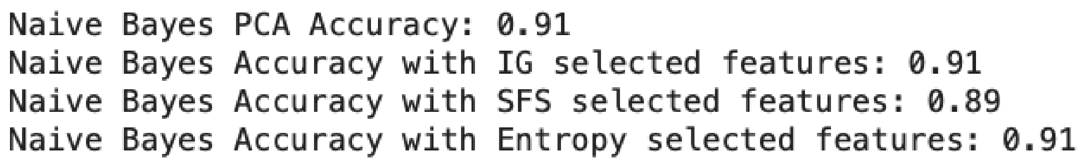
```
KNN Accuracy with PCA: 0.90
KNN Accuracy with SFS selected features: 0.88
KNN Accuracy with IG selected features: 0.90
KNN Accuracy with Entropy selected features: 0.90
```

2. **Naive bayes** classifier is an algorithm based on applying Bayes' theorem with "naive" assumption of conditional independence between every pair of features

   **Pros**: Performs well with a large number of features and works well with both categorical and numerical data just like our Breast Cancer dataset. Additionally, it's extremely fast as compared to other classification algorithms

   **Cons**: Poor estimator as it assumes feature independence

   Main Hyperparameters: N/A



```
Naive Bayes PCA Accuracy: 0.91
Naive Bayes Accuracy with IG selected features: 0.91
Naive Bayes Accuracy with SFS selected features: 0.89
Naive Bayes Accuracy with Entropy selected features: 0.91
```

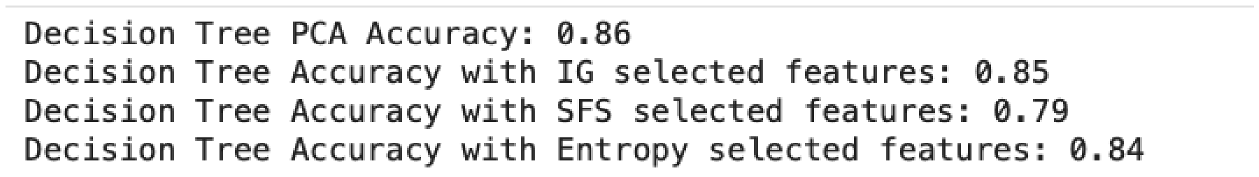
3. **C4.5 Decision Tree** is an extension of the ID3 algorithm and can handle both continuous and discrete data. It builds decision trees using a set of rules that minimally classify a set of data

   **Pros**: Easy to interpret, doesn't require scaling and can handle both numerical and categorical data

   **Cons**: Sensitive to noisy data hence prone to overfitting

   **Main Hyperparameters**:

   - **'max_depth'**: maximum depth of the tree
   - **'min_sample_split'**: minimum number of samples required to split an internal node.



```
Decision Tree PCA Accuracy: 0.86
Decision Tree Accuracy with IG selected features: 0.85
Decision Tree Accuracy with SFS selected features: 0.79
Decision Tree Accuracy with Entropy selected features: 0.84
```

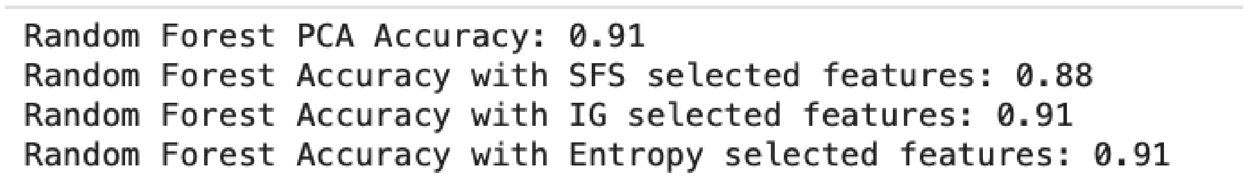
4. **Random Forest** is an ensemble method that constructs a multitude of decision trees at training time and outputs the class that is the mode(highest frequency) of the classes of the individual trees.

   **Pros**: Reduces overfitting in decision trees by effectively handling data imbalances improving overall accuracy of the decision trees

   **Cons**: Can be computationally intensive and complex to interpret

   **Main Hyperparameters**:

   - **'n_estimators'**: number of trees in the forest
   - **'max_depth'**: maximum depth of the tree
   - **'min_sample_split'**: minimum number of samples required to split an internal node.



```
Random Forest PCA Accuracy: 0.91
Random Forest Accuracy with SFS selected features: 0.88
Random Forest Accuracy with IG selected features: 0.91
Random Forest Accuracy with Entropy selected features: 0.91
```

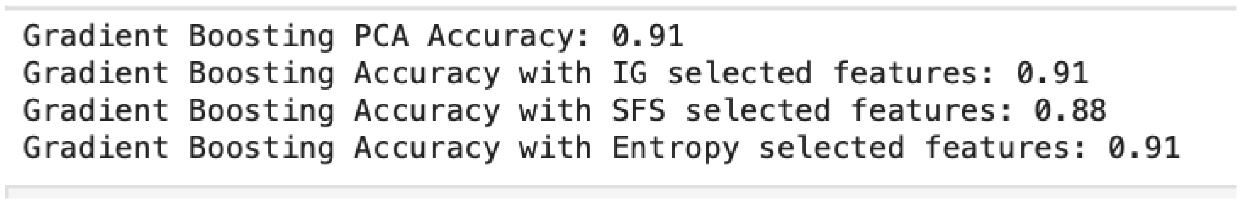
5. **Gradient Boosting** is an ensemble technique that builds multiple weak models sequentially, with each new model being trained to correct the errors made by the previous ones. Combining them all gets better performance as a whole.

   **Pros**: Very effective on a wide range of problems and often provides with a very high predictive accuracy

   **Cons**: Computationally expensive and can be prone to overfitting if not tuned properly.

**Main Hyperparameters:**

- **‘n_estimators’**: number of boosting stages to perform
- **‘max_depth’**: maximum depth of the tree; has a limitation for the number of nodes
- **‘learning rate’**: shrinks the contribution/weight of each tree by this value



```
Gradient Boosting PCA Accuracy: 0.91
Gradient Boosting Accuracy with IG selected features: 0.91
Gradient Boosting Accuracy with SFS selected features: 0.88
Gradient Boosting Accuracy with Entropy selected features: 0.91
```

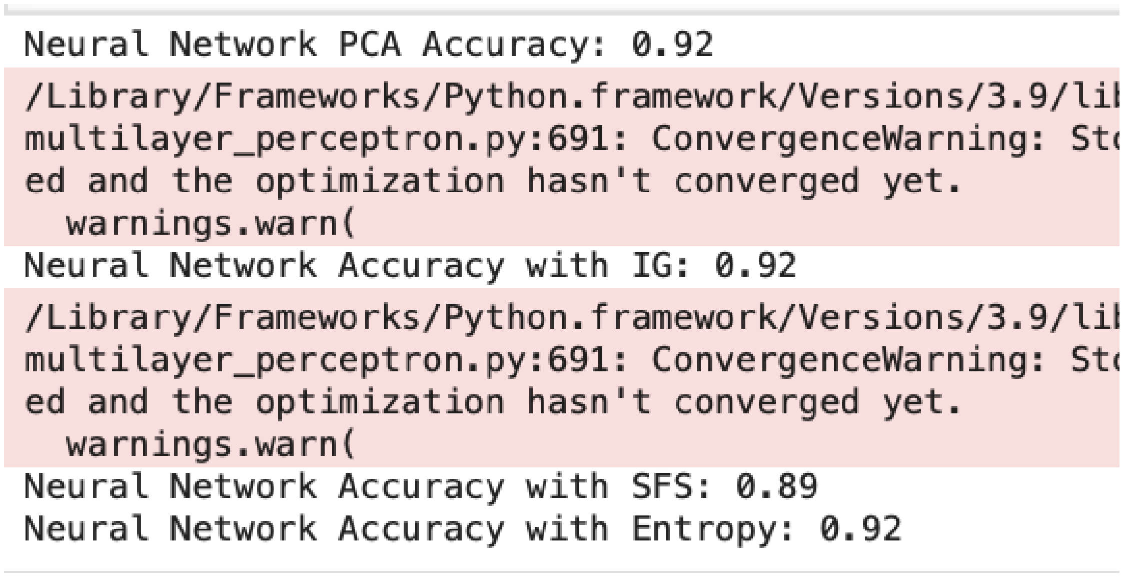
6. **Neural Networks** consist of  layers of interconnected nodes( aka neurons), where each layer performs certain transformations of the inputs received to capture complex patterns in the data.

   **Pros**: Can model non-linear relationships hence effective for a wide range of problems

   **Cons**: Prone to overfitting and requires a lot of data along with tuning many hyperparameters

   **Main Hyperparameters:**

   - **‘hidden_layer_sizes’**: the number of nodes/neurons in each layer
   - **‘activation’**: the activation function of the hidden layers
   - **‘solver’**: the algorithm for weight optimization
   - **‘alpha’**: L2(learning rate) regularization term that helps prevent overfitting



```
Neural Network PCA Accuracy: 0.92
/Library/Frameworks/Python.framework/Versions/3.9/li
multilayer_perceptron.py:691: ConvergenceWarning: Sto
ed and the optimization hasn't converged yet.
  warnings.warn(
Neural Network Accuracy with IG: 0.92
/Library/Frameworks/Python.framework/Versions/3.9/li
multilayer_perceptron.py:691: ConvergenceWarning: Sto
ed and the optimization hasn't converged yet.
  warnings.warn(
Neural Network Accuracy with SFS: 0.89
Neural Network Accuracy with Entropy: 0.92
```

**Results from the hyperparameter search**

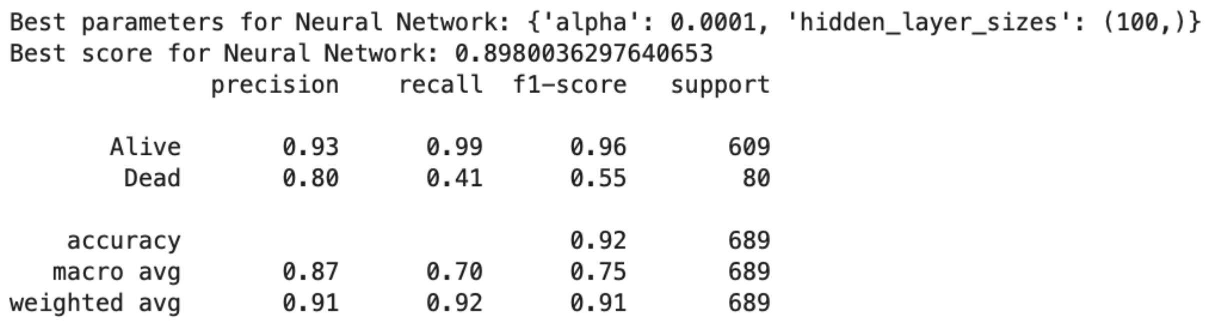
Two algorithms that offer interesting hyperparameter tuning options are **Random Forest** and **Neural Networks (MLPClassifier)**. I used Grid Search from scikit-learn to find the best hyperparameters for each, as it methodically tests combinations within a specified range to determine which configuration yields the best model performance. This method also performs cross-validation to ensure the model's robustness.

**1. For Random Forest Hyperparameter Tuning,** I tuned the following hyperparameters:

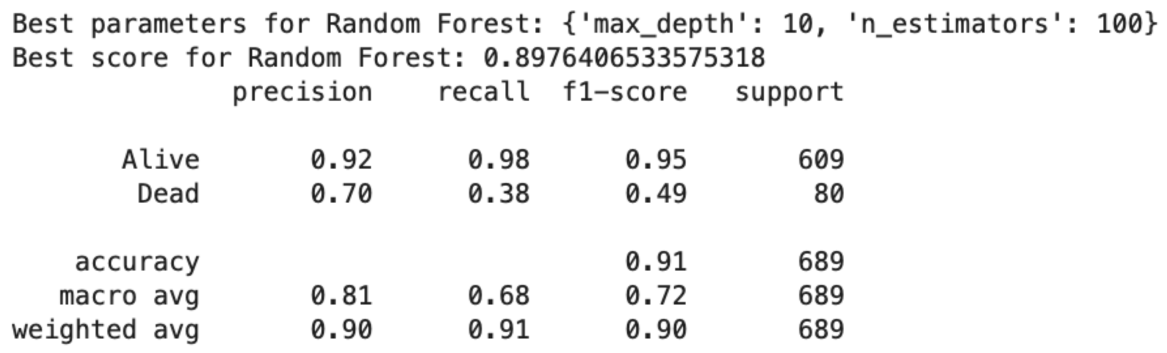
- **n_estimators**: Number of trees in the forest.
- **max_depth**: Maximum depth of each tree.

**2. For Neural Networks Hyperparameter Tuning**, I adjusted:

- **hidden_layer_sizes**: Number of neurons in the hidden layers.
- **alpha**: L2 penalty (regularization term) parameter.



```
Best parameters for Neural Network: {'alpha': 0.0001, 'hidden_layer_sizes': (100,)}
Best score for Neural Network: 0.8980036297640653
              precision    recall  f1-score   support

       Alive       0.93      0.99      0.96       609
        Dead       0.80      0.41      0.55        80

    accuracy                           0.92       689
   macro avg       0.87      0.70      0.75       689
weighted avg       0.91      0.92      0.91       689
```

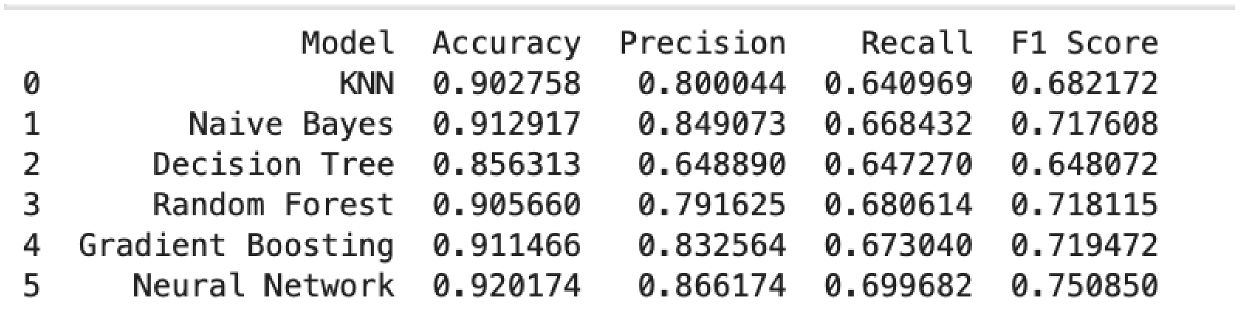


```
Best parameters for Random Forest: {'max_depth': 10, 'n_estimators': 100}
Best score for Random Forest: 0.8976406533575318
              precision    recall  f1-score   support

       Alive       0.92      0.98      0.95       609
        Dead       0.70      0.38      0.49        80

    accuracy                           0.91       689
   macro avg       0.81      0.68      0.72       689
weighted avg       0.90      0.91      0.90       689
```

The Hyperparameter search for the Neural Network has yielded ‘alpha: 0.0001’ and ‘hidden_layer_sizes: (100,)’ as the best parameters, achieving an overall score approximately 0.898, with notably high precision and recall for the ‘Alive’ class, but a lower recall for the ‘Dead’ class. On the other hand, the Random Forest’s best parameters were ‘max_depth: 10’ and ‘n_estimators: 100,’ with a similar overall score of approximately 0.897. However, the Random Forest model showed a slightly lower precision and recall for ‘Alive’ class and a modestly improved recall for the ‘Dead’ class compared to the Neural Network which is a little surprising for me.  Given the results, the Neural Network performed slightly better overall due to its higher overall score and substantially better performance on the ‘Alive’ class.

**Conclusions**

The preprocessing of the Breast Cancer dataset, including imputation, outlier removal, standardization, and PCA, effectively prepared the data for various machine learning models, ensuring a level playing field for features and reducing complexity. Among the evaluated models, the Neural Network excelled with the highest accuracy of 92%, benefiting from feature preprocessing, especially given its sensitivity to feature scale. It outperformed the Random Forest model, which also showed robust results with 91% accuracy, particularly in identifying the 'Dead' class. In terms of feature selection techniques, Information Gain consistently enhanced model performances, particularly in complex models like Neural Networks and Gradient Boosting, by effectively capturing the most relevant features. This suggests that IG is particularly useful in scenarios where inter-feature dependencies are significant.The choice between these models would thus hinge on the specific predictive needs of the task, with the Neural Network favored for overall performance and the Random Forest for its slightly better handling of the minority class.



```
               Model  Accuracy  Precision    Recall  F1 Score
0                KNN  0.902758   0.800044  0.640969  0.682172
1        Naive Bayes  0.912917   0.849073  0.668432  0.717608
2      Decision Tree  0.856313   0.648890  0.647270  0.648072
3      Random Forest  0.905660   0.791625  0.680614  0.718115
4  Gradient Boosting  0.911466   0.832564  0.673040  0.719472
5     Neural Network  0.920174   0.866174  0.699682  0.750850
```