Machine learning

Abstract:

The aim of the project is to identify the best machine learning algorithm for breast cancer classification using the Breast Cancer Wisconsin dataset. Logistic regression, random forest, decision tree, bayesian classification and k-nearest neighbour were chosen as candidate algorithms. Before anything, due to an initial imbalance of the dataset, oversampling was performed to achieve a balanced dataset.Then for each algorithm, to achieve the optimum performance of the models, hyperparameter tuning was conducted with k fold cross validation. After collecting the best hyperparameters, new model sets were trained for each algorithm using the best hyperparameters. The test set was then used to collect predictions of the model for performance evaluation. Evaluation metrics including accuracy, precision, recall, and F1 score were employed. The results indicated that Logistic Regression has the best performance among the five selected classification algorithms, making it a promising choice for breast cancer classification tasks.

Introduction:

Aim

The aim of the project is to find the optimal machine learning algorithm for precise classification of breast cancer for the Breast Cancer Wisconsin dataset. The goal is driven by the importance of the algorithm in this context, as its performance could have a direct impact on patients' lives. Therefore, It is crucial to find the most accurate and reliable algorithm, as misclassification must be avoided. In addition, to bring the best performance out from the classifying models, hyperparameter tuning will be conducted.

Literature Review

In the study conducted by Kadhim and Kamil (2022), they addressed that breast cancer is the second major cause of death in women. [2] Breast cancer occurs when cancer cells in the breast multiply inexorably. As there is still no definitive cure to stop the growth of breast cancer, it is advised to have regular check-ups and always be on the lookout for any abnormal tumour in the breast.

There are several methods to detect this disease, including biopsy, ultrasound, thermography, and fine needle aspiration biopsy [2]. However, the most effective of these methods is biopsy [2]. In a biopsy, a needle-like device is inserted into the breast to take a sample of the suspected tumour for analysis. Analysing the tissue sample can take several days, as many factors such as consistency, location and size have to be taken into account with this small sample. Even after the analysis, the decision between the experts may conflict and this is where the ML could approach. These machine learning models could help the decision process of finalising whether they are malignant or not. The models are not

biased but rather relying purely on their training data and looking at data trends so it is trustful and could reduce the risk of human error.
This project’s aim was to build a machine learning model that could be applied in real situations for breast cancer diagnosis. For this analysis they used the Wisconsin diagnostic breast cancer dataset. They evaluated using eleven different machine learning classifiers : decision tree, quadratic discriminant analysis, AdaBoost, Bagging meta estimator, Extra randomised trees, Gaussian process classifier, Ridge, Gaussian naive Bayes, k-Nearest neighbours, multilayer perceptron, and support vector classifier [2].

The project's conclusion highlighted that, among the various classifiers, the extremely randomised tree has the best performance of F1-score of 96.77%. Beyond just presenting the result, they talked about the characteristics of this classifier that particularly make it effective. They emphasised that ensemble methods, like extremely randomised trees, are ideal for cancer detection tasks due to their effectiveness in minimising model variability and enhancing prediction accuracy. Moreover, the sequential arrangement of weak learners in ensemble methods enables them to learn from previous ones, contributing to the overall improvement of prediction models [2].

Inspired by the findings of the Kadhim and Kamil (2022) project, I am motivated to investigate whether employing the widely used RandomForestClassifier, instead of the extremely randomised tree classifier, on the identical dataset, can perform as high as f1-score as 96.77%.

Dataset

The dataset used is Breast Cancer Wisconsin dataset from scikit-learn, it contains 569 entries , each consisting of 31 columns containing features extracted from a digitised image of a fine needle aspirate (FNA) of a breast mass [1]. These describe characteristics of the cell nuclei present in the image [1], providing valuable information for the analysis and classification of breast cancer.

The dataset consists of two classes: benign and malignant. However, it displays an imbalanced distribution, with 62.7% benign and 37.3% malignant entries. To tackle this class imbalance, the original dataset underwent random oversampling to achieve a balanced dataset. Balancing the dataset is crucial because training a model on imbalanced data can result in a biassed model that leans towards the majority class. Therefore, ensuring the balance of the dataset is essential to enhance the model's ability to make accurate predictions across all classes.

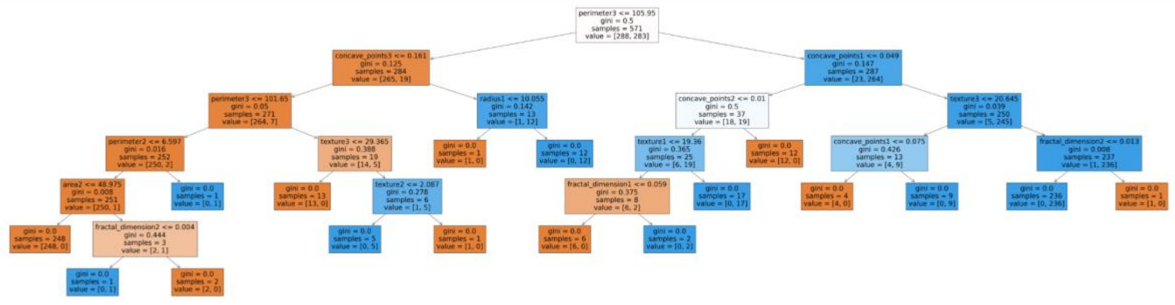
Background

When a machine learning algorithm undergoes training with a dataset and gains the ability to recognize data patterns, it is now a machine learning model [3]. Especially in the context of classification algorithms, the outcome is a classification model. When a classification model receives new data, it tries to predict the correct label for the data. This is exactly what we try to do by creating models of different classification algorithms and finding the algorithm with the best performance. The algorithms that are considered for this task are decision tree, logistic regression, naive Bayes, random forest and k-nearest neighbours.

Decision tree

A decision tree is a tree structured classifier, where internal nodes represent features of the dataset. The branches represent the decision rules, the tree starts with the basic decision rule (root node) a possible example could be ‘ Is compactness1 larger than 0.15 ? ‘. From there more decision rules follow, splitting the dataset. Each question helps to arrive at the final decision called a leaf node. After reaching a leaf node, the prediction for a new data point is determined by the majority class of the training instances that satisfy the conditions of that leaf node. This majority class serves as the final predicted outcome for the given set of feature conditions.

To have a better idea of how the decision tree works, Fig 1.1 has been plotted. The basic decision rule in this case is ‘perimeter3 larger or equal to 105.95’ from then continues to the next decision rule continuously splitting the dataset.



perimeter3 <= 105.95 gini = 0.5 samples = 571 value = [288, 283]
concave_points3 <= 0.161 gini = 0.125 samples = 284 value = [265, 19]
concave_points1 <= 0.049 gini = 0.147 samples = 287 value = [23, 264]
perimeter3 <= 101.65 gini = 0.05 samples = 271 value = [264, 7]
radius1 <= 10.055 gini = 0.142 samples = 13 value = [1, 12]
concave_points2 <= 0.01 gini = 0.5 samples = 37 value = [18, 19]
texture3 <= 20.645 gini = 0.039 samples = 250 value = [5, 245]
perimeter2 <= 6.597 gini = 0.016 samples = 252 value = [250, 2]
texture3 <= 29.365 gini = 0.388 samples = 19 value = [14, 5]
gini = 0.0 samples = 1 value = [1, 0]
gini = 0.0 samples = 12 value = [0, 12]
texture1 <= 19.36 gini = 0.365 samples = 25 value = [6, 19]
gini = 0.0 samples = 12 value = [12, 0]
concave_points1 <= 0.075 gini = 0.426 samples = 13 value = [4, 9]
fractal_dimension2 <= 0.013 gini = 0.008 samples = 237 value = [1, 236]
area2 <= 48.975 gini = 0.008 samples = 251 value = [250, 1]
gini = 0.0 samples = 1 value = [0, 1]
gini = 0.0 samples = 13 value = [13, 0]
texture2 <= 2.087 gini = 0.278 samples = 6 value = [1, 5]
fractal_dimension1 <= 0.059 gini = 0.375 samples = 8 value = [6, 2]
gini = 0.0 samples = 17 value = [0, 17]
gini = 0.0 samples = 4 value = [4, 0]
gini = 0.0 samples = 9 value = [0, 9]
gini = 0.0 samples = 236 value = [0, 236]
gini = 0.0 samples = 1 value = [1, 0]
gini = 0.0 samples = 248 value = [248, 0]
fractal_dimension2 <= 0.004 gini = 0.444 samples = 3 value = [2, 1]
gini = 0.0 samples = 5 value = [0, 5]
gini = 0.0 samples = 1 value = [1, 0]
gini = 0.0 samples = 6 value = [6, 0]
gini = 0.0 samples = 2 value = [0, 2]
gini = 0.0 samples = 1 value = [0, 1]
gini = 0.0 samples = 2 value = [2, 0]

Fig 1.1

K - Nearest Neighbours

In K-Nearest Neighbors, data points are represented in a feature space, where each data point becomes a coordinate based on its features. The algorithm determines the similarity between data points using distance metrics such as Euclidean distance, Manhattan distance. KNN operates by selecting the k nearest neighbours to a given data point, and the given point is classified as the majority class of these neighbours through a majority vote.

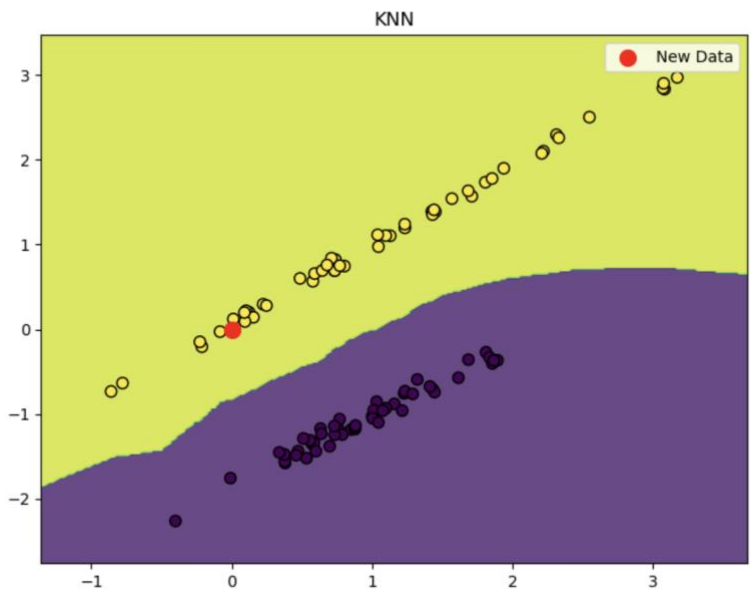


Fig 1.2

For better understanding, I have plotted a scatter diagram with two classes (blue and yellow) in Fig. 1.2. Employing KNN with a k-value of 3. The red dot is a new value to be classified. So to classify the new data, it is converted into coordinates. Then the distance between all available points is calculated and the 3 closest points are found. Through majority vote, the new data is classified. As shown, the 3 closest points to the red point are yellow, therefore this data is classified as yellow.

KNN is categorised as a lazy learner algorithm, meaning it does not actively learn from the training data during the training phase but instead stores all available data. When new data is presented for testing, the algorithm classifies it based on the similarity to the stored data points. This approach allows KNN to make predictions without a formal training phase and adapt to changes in the dataset during runtime.

For this algorithm various distance metrics could be used, the most commonly used metrics include Euclidean distance, Manhattan distance and Minkowski distance . For this project, to optimise the performance of the KNN model, all three of them were implemented and only one metric was chosen for final retraining through hyperparameter tuning.

The Euclidean distance calculates the distance between two points by:

$$Euclidean\ distance\ = \sqrt{(x_2 - x_1\ \ )^2 + (y_2 - y_1\ \ )^2}$$

The Manhattan distance calculates the distance between two points by:

$$Manhattan\ distance\ = |\ x1\ -\ x2\ |\ +\ |\ y1\ -\ y2\ |$$

The Minkowski distance calculates the distance between two points by:

$$Minkowski\ Distance = (\sum_{i=1}^{n}\ \ |\ x_{1i} - x_{2i}|^{p})^{\frac{1}{p}}$$

Bayesian Classification

Bayesian classification is based on Bayes' theorem, which aims to calculate the probability that a given hypothesis is true when a data is given. Identify the particular class the data belongs to using this equation :

$$P(H/X)\ = \frac{P(X/H)\ P(H)}{P(X)}$$

The algorithm assumes independence among features, and it calculates the probability of each class for a given data point, selecting the class with the highest probability.
In Bayesian classification, there are two types of probability used: posterior probability and prior probability.

When setting the scene with the dataset used in this project. Fig1.3 table shows the part of the first data in the dataset.

	radius1	texture1
Patient 1	17.99	10.38

Fig 1.3

- Let the hypothesis be H: ‘patient belongs to a benign class. ‘
- Let X as detail of patient: (radius1 =17.99)
-

Prior probability:

- P(B) - Probability of benign class $= \frac{total\ number\ of\ benign\ data}{total\ number\ of\ data} = \frac{357}{569}$
- P(X) -The total probability of observing same data value X (radius1=17.99)

Posterior probability:

- P(H/X) -The probability that the hypothesis H is true given the data X
- P(X/H) - The probability of observing the same data value X and hypothesis H is true.

These probabilities are calculated individually for each feature, and the process is repeated for the remaining features. Then another round of the same procedure is performed with a changed hypothesis, that the patient belongs to a malignant class. The calculated total probabilities are then collected, and the class with the highest probability is assigned to the data.

Logistic Regression

Logistic regression estimates the probability of the event and predicts the output of categorical dependent variables. So in this context, it would be benign and malignant but instead of giving distinct 0 and 1 , it will give probability values which are between 0 and 1. The sigmoid function, a mathematical transformation is applied in logistic regression, it would map these predicted values to probabilities. This function ensures that the output remains within the 0 to 1 range, forming an S-shaped curve. In logistic regression, the concept of a threshold value is employed, where values above the threshold tend to be classified as 1, and values below the threshold tend to be classified as 0. This threshold defines the point at which the predicted probability is either class benign or malignant.

Random Forest

The random forest algorithm works by combining the outputs of several decision trees to arrive at the final result. Unlike a single decision tree that considers all available factors, a random forest utilises bagging and features randomness to create an uncorrelated collection of decision trees. Each tree only examines a subset of random factors, ensuring that they don't depend on the same set of information. It is like getting diverse opinions to make more reliable decisions. By considering a variety of viewpoints it aims to make more accurate choices. This helps the random forest make more accurate predictions compared to a decision tree alone, because it avoids overfitting.

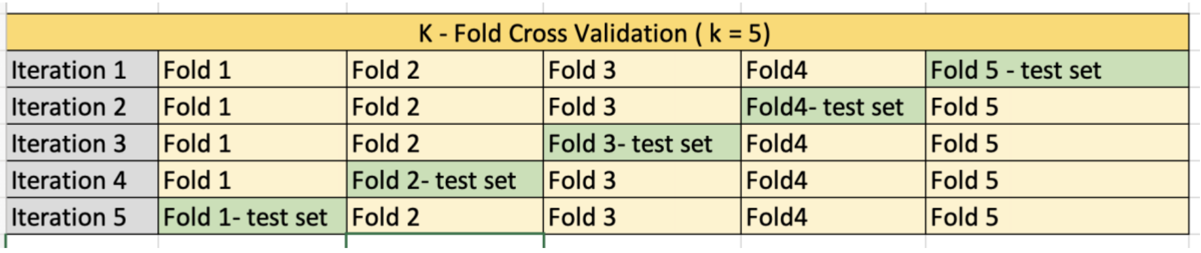
Methodology

Cross-validation was carried out in this project. This was necessary because the algorithm must perform well over the entire data set. The reason for this is that the algorithm may perform differently based on different training sets, e.g. using test set 1 may result in high accuracy, while accuracy may decrease when there is change in test set. To avoid this inconsistency and to find the algorithm that really performs well throughout the entire data set, a K-fold cross-validation was performed.

K - Fold cross validation

K-fold cross-validation is a method of splitting the dataset into ‘k’ folds, allowing each fold to take turns serving as both the training and test sets. For example, the value of ‘k’ is set to 5, the dataset is divided into 5 folds. In the first iteration, the initial four folds become the training set, while the fifth fold serves as the test set. In the subsequent iteration, the second last fold becomes the test set, and the remaining folds act as the training set, this process is repeated k times, ensuring each fold has its chance as both training and test data. Finally, the accuracy is determined by computing the mean accuracy across all the folds.

The fig 1.4 demonstrates how k fold cross validation is conducted.



K - Fold Cross Validation (k = 5)					
Iteration 1	Fold 1	Fold 2	Fold 3	Fold4	Fold 5 - test set
Iteration 2	Fold 1	Fold 2	Fold 3	Fold4- test set	Fold 5
Iteration 3	Fold 1	Fold 2	Fold 3- test set	Fold4	Fold 5
Iteration 4	Fold 1	Fold 2- test set	Fold 3	Fold4	Fold 5
Iteration 5	Fold 1- test set	Fold 2	Fold 3	Fold4	Fold 5

* yellow folds are train set   fig 1.4

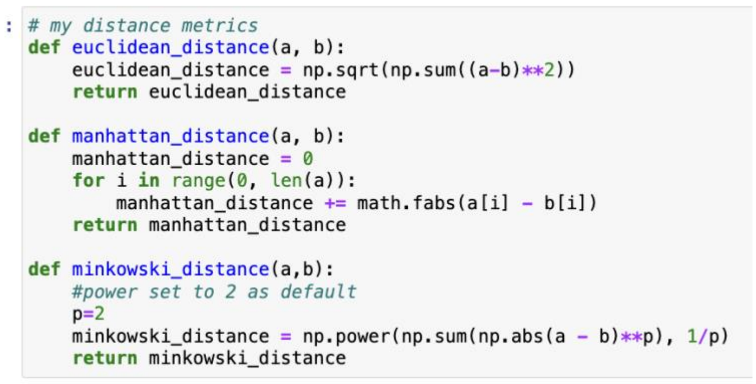
Hyperparameter tuning

As part of the project, hyperparameter tuning was performed using k-fold cross-validation, an essential step when comparing models with different algorithms. The reason for this is that different models may have different parameters that impact on performance of the model,and even can lead to overfitting or underfitting. However, hyperparameter tuning involves systematically examining a range of hyperparameter values to determine the combination that optimises the performance of the particular model. This optimisation helps to eliminate overfitting issues by preventing overly complex models through regularisation techniques and can adjust model complexity to reduce underfitting issues. Furthermore, the optimised performance resulting from hyperparameter tuning enables a fair and meaningful comparison of the performance of different models.

In the actual implementation, for each algorithm model hyperparameter tuning was done using GridSearchCV. Then when the optimal combination of hyperparameters were found for all five classifiers, a new set of models was retrained, incorporating these best hyperparameters. Only then the performance of these models was compared to find the most effective model.

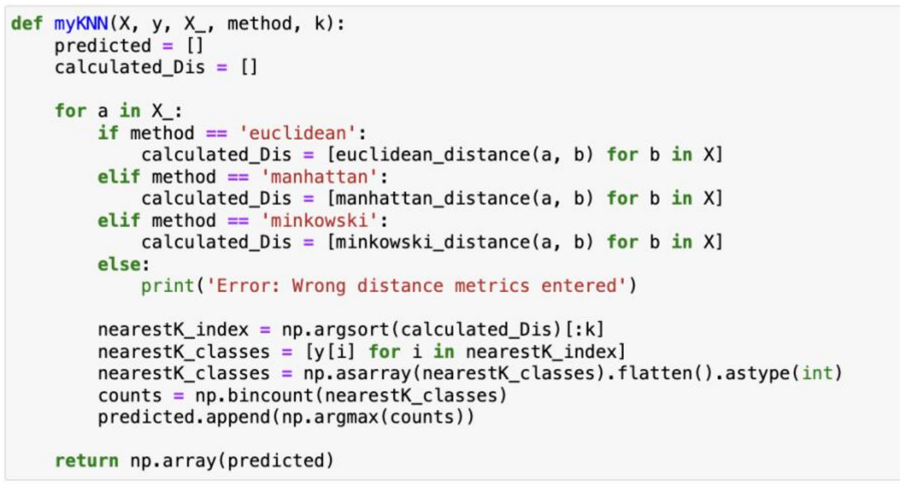
Implementation of ML algorithm from scratch

For implementation of one machine learning algorithm from scratch, I have implemented KNN on my own. Firstly, the three distance metrics were written.



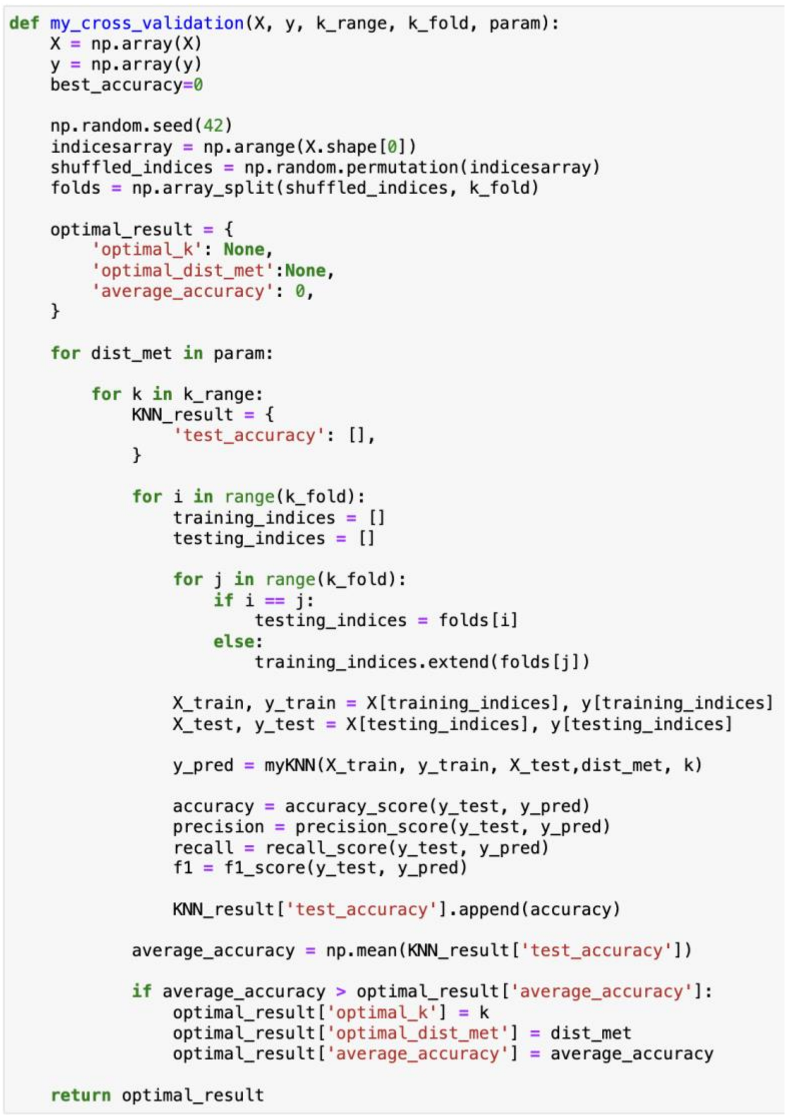
```
: # my distance metrics
  def euclidean_distance(a, b):
      euclidean_distance = np.sqrt(np.sum((a-b)**2))
      return euclidean_distance

  def manhattan_distance(a, b):
      manhattan_distance = 0
      for i in range(0, len(a)):
          manhattan_distance += math.fabs(a[i] - b[i])
      return manhattan_distance

  def minkowski_distance(a,b):
      #power set to 2 as default
      p=2
      minkowski_distance = np.power(np.sum(np.abs(a - b)**p), 1/p)
      return minkowski_distance
```

Secondly, I implemented the myKNN function. For each data point for classification, the function finds the k-nearest neighbours based on the specified distance metric. Then the majority class among these neighbours is predicted as the assigned class for the given data point and stored in an array to be returned.



```
def myKNN(X, y, X_, method, k):
    predicted = []
    calculated_Dis = []

    for a in X_:
        if method == 'euclidean':
            calculated_Dis = [euclidean_distance(a, b) for b in X]
        elif method == 'manhattan':
            calculated_Dis = [manhattan_distance(a, b) for b in X]
        elif method == 'minkowski':
            calculated_Dis = [minkowski_distance(a, b) for b in X]
        else:
            print('Error: Wrong distance metrics entered')

        nearestK_index = np.argsort(calculated_Dis)[:k]
        nearestK_classes = [y[i] for i in nearestK_index]
        nearestK_classes = np.asarray(nearestK_classes).flatten().astype(int)
        counts = np.bincount(nearestK_classes)
        predicted.append(np.argmax(counts))

    return np.array(predicted)
```

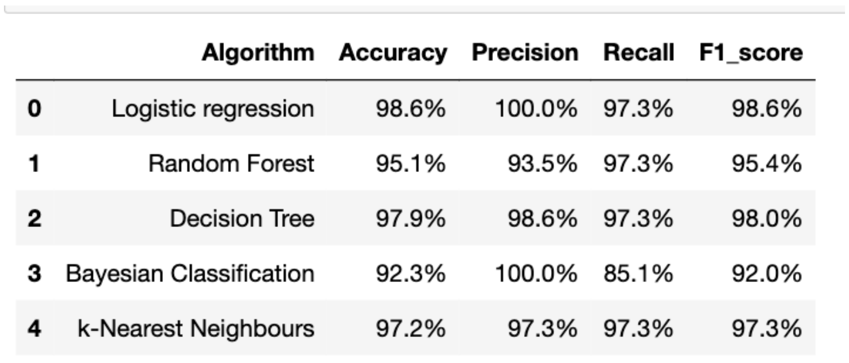
Lastly, to optimise the KNN model, I performed K-fold cross-validation to find the best hyperparameters. For this purpose, I wrote a function called my_cross_validation specifically written for KNN cross-validation. It iterates over different distance metrics and a range of values for the number of neighbours. The function evaluates the model's performance using accuracy across multiple folds, aiming to identify the best combination of k and distance metric. The optimal parameters are then returned.



```python
def my_cross_validation(X, y, k_range, k_fold, param):
    X = np.array(X)
    y = np.array(y)
    best_accuracy=0

    np.random.seed(42)
    indicesarray = np.arange(X.shape[0])
    shuffled_indices = np.random.permutation(indicesarray)
    folds = np.array_split(shuffled_indices, k_fold)

    optimal_result = {
        'optimal_k': None,
        'optimal_dist_met':None,
        'average_accuracy': 0,
    }

    for dist_met in param:

        for k in k_range:
            KNN_result = {
                'test_accuracy': [],
            }

            for i in range(k_fold):
                training_indices = []
                testing_indices = []

                for j in range(k_fold):
                    if i == j:
                        testing_indices = folds[i]
                    else:
                        training_indices.extend(folds[j])

                X_train, y_train = X[training_indices], y[training_indices]
                X_test, y_test = X[testing_indices], y[testing_indices]

                y_pred = myKNN(X_train, y_train, X_test,dist_met, k)

                accuracy = accuracy_score(y_test, y_pred)
                precision = precision_score(y_test, y_pred)
                recall = recall_score(y_test, y_pred)
                f1 = f1_score(y_test, y_pred)

                KNN_result['test_accuracy'].append(accuracy)

            average_accuracy = np.mean(KNN_result['test_accuracy'])

            if average_accuracy > optimal_result['average_accuracy']:
                optimal_result['optimal_k'] = k
                optimal_result['optimal_dist_met'] = dist_met
                optimal_result['average_accuracy'] = average_accuracy

    return optimal_result
```

Result

In this project, I evaluated five different machine learning algorithms for breast cancer classification to find the best algorithm for breast cancer classification. For each algorithm model, the optimal hyperparameters were found by K-fold cross-validation. Then, the new set of algorithms were retrained with their best hyperparameters to compare their performance. After the retraining phase, the evaluation was conducted, the evaluation metrics including accuracy, precision, recall and F1-score were used to determine the performance of each model. To clearly visualise the result, the table in Fig. 1.5 is drawn to show the results.



	Algorithm	Accuracy	Precision	Recall	F1_score
0	Logistic regression	98.6%	100.0%	97.3%	98.6%
1	Random Forest	95.1%	93.5%	97.3%	95.4%
2	Decision Tree	97.9%	98.6%	97.3%	98.0%
3	Bayesian Classification	92.3%	100.0%	85.1%	92.0%
4	k-Nearest Neighbours	97.2%	97.3%	97.3%	97.3%

Fig 1.5

The evaluation of five machine learning algorithms using the given data set showed that all five algorithms were very efficient, as the accuracy was above 90 % for all of them. Logistic regression had the best performance with an impressive overall accuracy of 98.6%, accompanied by perfect precision and high recall, resulting in an F1 score of 98.6%. Decision Tree also performs strongly across all metrics, achieving an accuracy of 97.9%, a precision of 98.6% and a balanced recall of 97.3%, resulting in an F1 score of 98.0%. Random Forest follows close behind with an accuracy of 95.1% and F1 score of 95.4%. While k-Nearest Neighbours shows a consistently robust performance with an accuracy of 97.2% and precision, recall and F1 score of 97.3%, Bayesian Classification lags slightly behind with an accuracy of 92.3%, a perfect precision but a lower recall of 85.1% and an F1 score of 92.0%.

Evaluation

The aim of the project was to find the most effective machine learning algorithm that could classify breast cancer accurately based on given data. The aim has been achieved by finalising that the Logistic regression algorithm is the best algorithm.

The logistic regression model has very high performance, however, this algorithm should not be trusted 100 percent in the case of real-life implementation. The model is not perfect, there is always a chance of making an error. The accuracy of the logistic regression model was 98.6%, which implies there is 1.4% chance of making an error. So, if this logistic regression model were implemented in a real hospital environment, medical staff should always double check and avoid making a final decision based on this model result only. The output of the classifier should only be considered to support the decision-making process of the expert.

In addition, the ambitious goal of the project might be to achieve 100% accuracy, but this is not possible because the characteristics of breast cancer cannot always be constant and there is always an outlier because human diseases do not always have the same symptoms. So even if I achieve a model with 100% accuracy, the model cannot be 100% accurate in real life.

Conclusion

In conclusion, among the five selected algorithms, the logistic regression algorithm is the best algorithm for breast cancer classification using the Breast Cancer Wisconsin dataset. It outperformed with an accuracy of 98.6% and a precision of 100%, making the model highly accurate and trustworthy.

The project conducted by Kadhim and Kamil (2022) achieved an F1 score of 96.77% with an extremely random forest algorithm [2]. I aimed to determine whether a regular random forest could achieve performance comparable to that of the extremely random forest. In my implementation, the random forest achieved an F1 score of 95.4%, quite close to 96.77%. However, it indicates that the Random Forest algorithm has slightly lower performance compared to the extremely random forest with this dataset.

References

[1] Wolberg, W., Mangasarian, O., Street, N., & Street, W. (1995). Breast Cancer Wisconsin (Diagnostic). UCI Machine Learning Repository. Retrieved from https://doi.org/10.24432/C5DW2B.

[2] Kadhim, R., & Kamil, M. (2022). Comparison of breast cancer classification models on Wisconsin dataset. *International Journal of Reconfigurable and Embedded Systems (IJRES)*, 11, 166-174. https://doi.org/10.11591/ijres.v11.i2.pp166-174..

[3] Amann, A. (2022, September 13). Machine Learning Algorithm or Machine Learning Model. Techopedia.Retrieved from https://www.techopedia.com/machine-learning-algorithm-or-machine-learning-model/7/34855

[4] IBM. (n.d.). K-Nearest Neighbors Algorithm. Retrieved from https://www.ibm.com/topics/knn#:~:text=Next%20steps-,K%2DNearest%20Neighbors%20Algorithm,of%20an%20individual%20data%20point.

[5] GeeksforGeeks. (n.d.). Decision Tree. Retrieved from https://www.geeksforgeeks.org/decision-tree/

[6]JavaTpoint. (n.d.). K-Nearest Neighbor Algorithm for Machine Learning. Retrieved from https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning

[7] GeeksforGeeks. (n.d.). K-Nearest Neighbours (KNN). Retrieved from https://www.geeksforgeeks.org/k-nearest-neighbours/

[8]JavaTpoint. (n.d.). Data Mining Bayesian Classifiers. Retrieved from https://www.javatpoint.com/data-mining-bayesian-classifiers

[9]Brownlee, J. (2020, August 26). How to Configure k-Fold Cross-Validation. Machine Learning Mastery. Retrieved from https://machinelearningmastery.com/how-to-configure-k-fold-cross-validation/

[10]GeeksforGeeks. (n.d.). Understanding Logistic Regression. GeeksforGeeks. Retrieved from https://www.geeksforgeeks.org/understanding-logistic-regression/

[11] Brownlee, J. (2020, August 28). Hyperparameters for Classification Machine Learning Algorithms. Machine Learning Mastery. Retrieved from https://machinelearningmastery.com/hyperparameters-for-classification-machine-learning-algorithms/