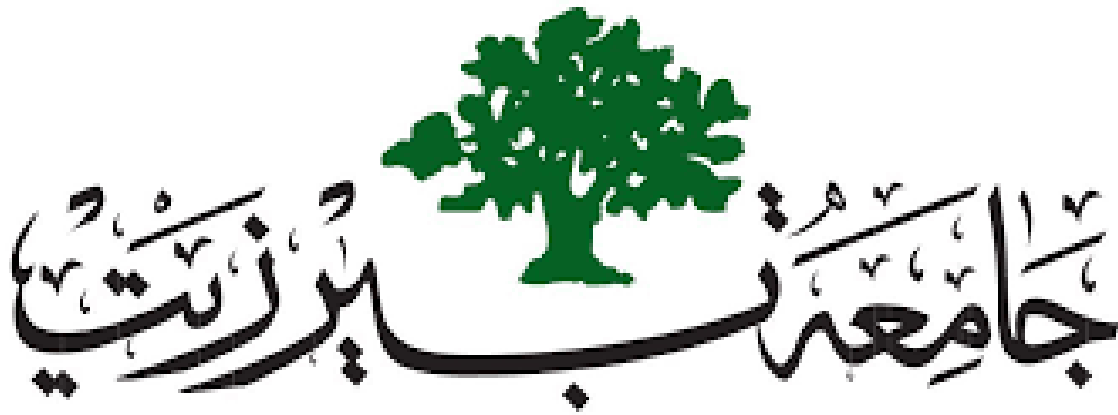


"بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ"



BIRZEIT UNIVERSITY

Faculty of Engineering and Technology.

Department of Electrical and Computer Engineering.

MACHINE LEARNING AND DATA SCIENCE - ENCS5341.

**Assignment #3: KNN, Logistic Regression, SVM, Kernel
Methods, and Ensemble Methods.**

Students name (Prepared by): Razan Abdelrahman –1200531.

Hidaya Mustafa - 1201910.

Instructor: Dr. Ismail Khater.

Section: 2.

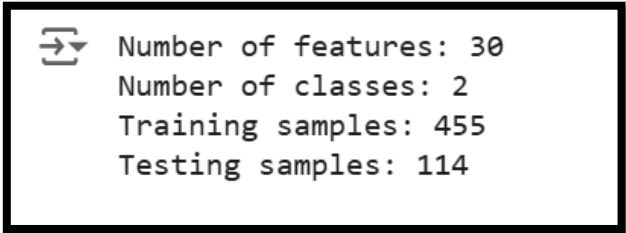
Date: 28th | December.

1. Introduction.

In this assignment, we explore fundamental machine learning algorithms, including K-Nearest Neighbors (KNN), Logistic Regression, Support Vector Machines (SVM), and ensemble methods like Boosting and Bagging. These algorithms are essential tools for tackling classification problems and offer diverse approaches to model building and decision-making.

2. Dataset.

- **Dataset Description:** The Breast Cancer dataset is a widely used benchmark in machine learning and medical research. It consists of clinical and pathological features to predict whether a tumor is malignant (cancerous) or benign (non-cancerous).
- **Dataset information:**
 - The dataset includes 30 features, such as mean radius, mean texture, and mean perimeter, which describe various clinical and pathological characteristics of tumors.
 - The dataset contains two target classes: 0: Benign and 1: Malignant.
 - Training Samples: 80% of the dataset (used for training the models).
 - Testing Samples: 20% of the dataset (used for evaluating model performance).



```
➡ Number of features: 30
   Number of classes: 2
   Training samples: 455
   Testing samples: 114
```

The dataset contains 30 features that represent clinical and pathological characteristics of tumors, including [*mean radius, mean texture, mean perimeter, mean area, mean smoothness, mean compactness, mean concavity, mean concave points, mean symmetry, mean fractal dimension, radius error, texture error, perimeter error, area error, smoothness error, compactness error, concavity error, concave points error, symmetry error, fractal dimension error, worst radius, worst texture, worst perimeter, worst area, worst smoothness, Worst compactness, worst concavity, worst concave points, worst symmetry, and worst fractal dimension*]. Additionally, there is a target column indicating whether a tumor is benign (0) or malignant (1).

The following table shows the first five rows of the dataset, with 30 numerical features and a target column indicating tumor classification.

| mean radius | mean texture | mean perimeter | mean area | mean smoothness | mean compactness | mean concavity | mean concave points | mean symmetry | mean fractal dimension | ... | worst texture | worst perimeter | worst area | worst smoothness | worst compactness | worst concavity | worst concave points | worst symmetry | worst fractal dimension | target |
|----------------|-----------------|-------------------|--------------|--------------------|---------------------|-------------------|---------------------------|------------------|------------------------------|-----|------------------|--------------------|---------------|---------------------|----------------------|--------------------|----------------------------|-------------------|-------------------------------|--------|
| 17.99 | 10.38 | 122.80 | 1001.0 | 0.11840 | 0.27760 | 0.3001 | 0.14710 | 0.2419 | 0.07871 | ... | 17.33 | 184.60 | 2019.0 | 0.1622 | 0.6656 | 0.7119 | 0.2654 | 0.4601 | 0.11890 | 0 |
| 20.57 | 17.77 | 132.90 | 1326.0 | 0.08474 | 0.07864 | 0.0869 | 0.07017 | 0.1812 | 0.05667 | ... | 23.41 | 158.80 | 1956.0 | 0.1238 | 0.1866 | 0.2416 | 0.1860 | 0.2750 | 0.08902 | 0 |
| 19.69 | 21.25 | 130.00 | 1203.0 | 0.10960 | 0.15990 | 0.1974 | 0.12790 | 0.2069 | 0.05999 | ... | 25.53 | 152.50 | 1709.0 | 0.1444 | 0.4245 | 0.4504 | 0.2430 | 0.3613 | 0.08758 | 0 |
| 11.42 | 20.38 | 77.58 | 386.1 | 0.14250 | 0.28390 | 0.2414 | 0.10520 | 0.2597 | 0.09744 | ... | 26.50 | 98.87 | 567.7 | 0.2098 | 0.8663 | 0.6869 | 0.2575 | 0.6638 | 0.17300 | 0 |
| 20.29 | 14.34 | 135.10 | 1297.0 | 0.10030 | 0.13280 | 0.1980 | 0.10430 | 0.1809 | 0.05883 | ... | 16.67 | 152.20 | 1575.0 | 0.1374 | 0.2050 | 0.4000 | 0.1625 | 0.2364 | 0.07678 | 0 |

ws x 31 columns

Also the following table summarizes the statistical properties of the dataset's features, including count, mean, standard deviation (std), minimum, maximum, and quartiles (25%, 50%, 75%). These statistics provide an overview of the distribution and range of the 30 numerical features used for tumor classification.


| | mean radius | mean texture | mean perimeter | mean area | mean smoothness | mean compactness | mean concavity | mean concave points | mean symmetry | mean fractal dimension | ... | worst texture | worst perimeter | worst area | worst smoothness | worst compactness | worst concavity |
|-------|----------------|-----------------|-------------------|--------------|--------------------|---------------------|-------------------|---------------------------|------------------|------------------------------|-----|------------------|--------------------|---------------|---------------------|----------------------|--------------------|
| count | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | ... | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.000000 |
| mean | 14.127292 | 19.289649 | 91.969033 | 654.889104 | 0.096360 | 0.104341 | 0.088799 | 0.048919 | 0.181162 | 0.062798 | ... | 25.677223 | 107.261213 | 880.583128 | 0.132369 | 0.254265 | 0.272188 |
| std | 3.524049 | 4.301036 | 24.298981 | 351.914129 | 0.014064 | 0.052813 | 0.079720 | 0.038803 | 0.027414 | 0.007060 | ... | 6.146258 | 33.602542 | 569.356993 | 0.022832 | 0.157336 | 0.208624 |
| min | 6.981000 | 9.710000 | 43.790000 | 143.500000 | 0.052630 | 0.019380 | 0.000000 | 0.000000 | 0.106000 | 0.049960 | ... | 12.020000 | 50.410000 | 185.200000 | 0.071170 | 0.027290 | 0.000000 |
| 25% | 11.700000 | 16.170000 | 75.170000 | 420.300000 | 0.086370 | 0.064920 | 0.029560 | 0.020310 | 0.161900 | 0.057700 | ... | 21.080000 | 84.110000 | 515.300000 | 0.116600 | 0.147200 | 0.114500 |
| 50% | 13.370000 | 18.840000 | 86.240000 | 551.100000 | 0.095870 | 0.092630 | 0.061540 | 0.033500 | 0.179200 | 0.061540 | ... | 25.410000 | 97.660000 | 686.500000 | 0.131300 | 0.211900 | 0.226700 |
| 75% | 15.780000 | 21.800000 | 104.100000 | 782.700000 | 0.105300 | 0.130400 | 0.130700 | 0.074000 | 0.195700 | 0.066120 | ... | 29.720000 | 125.400000 | 1084.000000 | 0.146000 | 0.339100 | 0.382900 |
| max | 28.110000 | 39.280000 | 188.500000 | 2501.000000 | 0.163400 | 0.345400 | 0.426800 | 0.201200 | 0.304000 | 0.097440 | ... | 49.540000 | 251.200000 | 4254.000000 | 0.222600 | 1.058000 | 1.252000 |

8 rows x 31 columns

3. Part 1: K-Nearest Neighbors (KNN).

3.1 Implement the KNN algorithm using APIs.

To implement the K-Nearest Neighbors (KNN) algorithm, we used the *KNeighborsClassifier* from the *Scikit-learn* library. This section explores the performance of the KNN model with different values of k (number of neighbors): 2, 5, 10, and 15.

 Accuracy of the KNN model using APIs with k = 2 : 92.98%
 Accuracy of the KNN model using APIs with k = 5 : 95.61%
 Accuracy of the KNN model using APIs with k = 10 : 97.37%
 Accuracy of the KNN model using APIs with k = 15 : 96.49%


The results indicate that the accuracy of the KNN model improved as the number of neighbors increased up to $k=10$, achieving the highest accuracy of **97.37%**. However, further increasing k to 15 resulted in a slight decrease in accuracy, suggesting that an excessively high k value may lead to under fitting.

3.2 Impact of Distance Metrics on KNN Performance.

To evaluate the effect of different distance metrics on the performance of the K-Nearest Neighbors (KNN) algorithm, we implemented the model using three commonly used metrics:

1. **Euclidean Distance:** Measures the straight-line distance between two points.
2. **Manhattan Distance:** Measures the sum of absolute differences between coordinates.
3. **Cosine Distance:** Measures the cosine of the angle between two vectors, focusing on their orientation rather than magnitude.

The model was trained and tested using $k = 10$ and performance metrics such as Accuracy, Precision, Recall, F1-Score, and ROC-AUC were computed.

| | |
|--|--|
|  Euclidean Distance: Accuracy: 97.37% Precision: 97.22% Recall: 98.59% F1-Score: 97.90% ROC-AUC: 96.97% | Manhattan Distance: Accuracy: 97.37% Precision: 97.22% Recall: 98.59% F1-Score: 97.90% ROC-AUC: 96.97% |
| Cosine Distance: Accuracy: 95.61% Precision: 94.59% Recall: 98.59% F1-Score: 96.55% ROC-AUC: 94.64% | |

The results show that Euclidean and Manhattan distances perform equally well, achieving the highest accuracy (97.37%) and consistent metrics across Precision, Recall, F1-Score, and ROC-AUC. In contrast, Cosine distance produced slightly lower performance (Accuracy: 95.61%, ROC-AUC: 94.64%), likely because it focuses on vector orientation rather than magnitude, which may be less suitable for this dataset.

Overall, Euclidean and Manhattan distances are more effective for this task, while Cosine distance remains a viable but slightly less optimal choice.

3.3 Determine the optimal value of K (number of neighbors) using cross-validation.

To determine the optimal number of neighbors (k) for the K-Nearest Neighbors (KNN) algorithm, we performed a GridSearch with cross-validation. A range of k values from 1 to 20 was tested using 5-fold cross-validation to evaluate the model's performance for each k .

```
➡ Optimal k (number of neighbors): 9  
Accuracy of the KNN model with optimal k=9: 95.61%
```

Cross-validation ensured a robust selection of k by evaluating model performance across multiple data splits. The optimal value of k balances the trade-off between overfitting (low k) and under fitting (high k), ensuring reliable predictions on unseen data. While the accuracy of **95.61%** is slightly lower than the maximum observed in earlier experiments with $k=10$, the use of cross-validation adds confidence to the choice of $k=9$ as the optimal parameter for this dataset.

4. Part 2: Logistic Regression.

To implement the Logistic Regression model, the features were first standardized using the *StandardScaler* from *Scikit-learn*. This ensures all features have a mean of 0 and a standard deviation of 1, which is crucial for algorithms like Logistic Regression to perform optimally. The model was then trained on the scaled training set and evaluated on the scaled test set.

```
➡ Logistic Regression :  
Accuracy: 97.37%  
Precision: 97.22%  
Recall: 98.59%  
F1 Score: 97.90%  
ROC-AUC: 96.97%
```

The Logistic Regression model demonstrated excellent performance, achieving high values across all metrics. This indicates that the model is effective in classifying tumors

as benign or malignant. The high Recall (98.59%) is particularly noteworthy for medical datasets, as it reflects the model's ability to correctly identify malignant tumors, minimizing false negatives.

4.1 Impact of Regularization Techniques in Logistic Regression.

To explore the effect of regularization on the performance of Logistic Regression, two regularization techniques were tested:

- 1. L1 Regularization (Lasso):** Encourages sparsity by shrinking some feature coefficients to zero, effectively performing feature selection.
- 2. L2 Regularization (Ridge):** Penalizes large coefficients, preventing overfitting by distributing the penalty across all features.

Both models were trained with the *liblinear* solver, appropriate for smaller datasets and for handling L1 regularization.



L1 Regularization (Lasso) Performance:

Accuracy: 95.61%
Precision: 94.59%
Recall: 98.59%
F1 Score: 96.55%
ROC-AUC: 94.64%

L2 Regularization (Ridge) Performance:

Accuracy: 95.61%
Precision: 94.59%
Recall: 98.59%
F1 Score: 96.55%
ROC-AUC: 94.64%

Both L1 and L2 regularization yielded identical performance metrics, suggesting that regularization did not significantly impact the model's predictive power for this dataset. However, L1 regularization offers the additional benefit of feature selection, which could improve interpretability and efficiency, particularly for high-dimensional datasets. L2 regularization, on the other hand, is more effective in preventing overfitting by balancing the influence of all features.

4.2 Compare the performance of Logistic Regression with KNN.

Logistic Regression outperforms KNN in terms of accuracy, precision, F1-score, and ROC-AUC. Logistic Regression shows a higher overall performance, especially in precision and F1-score, indicating that it is better at correctly identifying positive instances and achieving a balanced performance.

KNN (using Euclidean or Manhattan distance) provides strong recall and accuracy, especially for classifying positive instances (as indicated by the high recall of 98.59%),

but its precision and F1-score are slightly lower than Logistic Regression. Cosine distance leads to lower overall performance across all metrics, with significantly lower accuracy and ROC-AUC.

5. Part 3: Support Vector Machines (SVM).

5.1 Implement the SVM algorithm using APIs.

To implement the Support Vector Machines (SVM) algorithm, we used the *Scikit-learn* *SVC* class with the default kernel (RBF). The model was trained on the dataset and evaluated using common classification metrics. The following performance was observed:

```
SVM Performance with Default (RBF) Kernel:  
Accuracy: 94.74%  
Precision: 92.21%  
Recall: 100.00%  
F1-Score: 95.95%  
ROC-AUC: 99.34%
```

The SVM with the default RBF kernel demonstrated strong performance, particularly with a perfect recall score of 100.00%, ensuring all malignant cases were correctly identified. The high F1-Score (95.95%) reflects a balance between precision and recall, while the ROC-AUC (99.34%) highlights its exceptional ability to distinguish between classes. This makes the RBF kernel a robust choice for datasets where minimizing false negatives is critical, such as medical diagnosis, although slightly lower precision (92.21%) indicates the presence of some false positives.

5.2 Performance Comparison of SVM Kernels.

To evaluate the performance of Support Vector Machines (SVM), the model was trained using three different kernels: linear, polynomial (poly), and radial basis function (RBF). These kernels determine the decision boundary's shape and allow the SVM



```
Kernel: linear  
Accuracy: 95.61%  
Precision: 94.67%  
Recall: 98.61%  
F1-Score: 96.60%  
ROC-AUC: 99.64%
```

```
Kernel: poly  
Accuracy: 92.11%  
Precision: 90.91%  
Recall: 97.22%  
F1-Score: 93.96%  
ROC-AUC: 96.63%
```

```
Kernel: rbf  
Accuracy: 92.98%  
Precision: 92.11%  
Recall: 97.22%  
F1-Score: 94.59%  
ROC-AUC: 96.96%
```

algorithm to adapt to different data distributions. Metrics such as Accuracy, Precision, Recall, F1-Score, and ROC-AUC were computed to compare their performance.

The **linear kernel** outperformed the other kernels in terms of Accuracy (95.61%), F1-Score (96.60%), and ROC-AUC (99.64%), indicating that the data is likely linearly separable. The **polynomial kernel** showed the lowest performance across metrics, which might suggest overfitting or that it struggled to model the data effectively. The **RBF kernel**, while slightly better than the polynomial kernel, also underperformed compared to the linear kernel, possibly due to the dataset's simplicity favoring linear decision boundaries.

5.3 Impact of Kernel Choice on Model Performance.

The choice of kernel significantly impacts SVM performance. For this dataset, the linear kernel achieves the highest accuracy, ROC-AUC, and F1-Score, making it the most suitable choice due to the dataset's relatively simple, linear decision boundary. While the polynomial and RBF kernels demonstrate strong recall, their slightly lower precision and F1-scores suggest they may be better suited for more complex, non-linear datasets.

6. Part 4: Ensemble Methods.

6.1 Comparison of Ensemble Methods: AdaBoost vs. Random Forest.

To evaluate the performance of ensemble methods, we implemented two models: AdaBoost (Adaptive Boosting) and Random Forest (Bagging). Both models were trained on the dataset to compare their effectiveness using various classification metrics, including Accuracy, Precision, Recall, F1-Score, and ROC-AUC.

1. **AdaBoost:** Combines weak learners (decision stumps) sequentially, focusing on misclassified samples in each iteration to improve performance.

2. **Random Forest:** Uses a bagging approach with multiple decision trees, aggregating their outputs for robust predictions.

| | | |
|---------------------------------|----------|---------------|
| Comparison of Ensemble Methods: | | |
| Metric | AdaBoost | Random Forest |
| Accuracy | 95.61% | 95.61% |
| Precision | 94.67% | 95.89% |
| Recall | 98.61% | 97.22% |
| F1-Score | 96.60% | 96.55% |
| ROC-AUC | 98.25% | 99.37% |

Both AdaBoost and Random Forest achieved identical accuracy (95.61%) and similar F1-scores, reflecting comparable overall performance. AdaBoost excelled in Recall (98.61%), making it more effective at identifying true positive cases, which is vital in medical datasets. In contrast, Random Forest showed higher Precision (95.89%), reducing false positives, and outperformed AdaBoost in ROC-AUC (99.37%), indicating better class distinction in probabilistic predictions.

6.2 Performance Evaluation of Ensemble Methods vs. Individual Models.

- Which Ensemble Method Performed Better and Why?

Random Forest performed slightly better than AdaBoost in precision, ROC-AUC, and overall robustness, leveraging multiple decision trees to reduce overfitting and handle noisy data effectively. However, AdaBoost excelled in recall, making it ideal for applications prioritizing the identification of hard-to-classify positive instances.

- Comparison with Individual Models (KNN, Logistic Regression, SVM).

Ensemble methods like Random Forest and AdaBoost outperform KNN, which struggles with decision boundary misclassifications, particularly in high-dimensional spaces. While Logistic Regression and SVM with linear kernels provide strong performance and are competitive, ensemble methods excel in handling non-linear scenarios and offer more stable and robust precision and recall across various data types.

7. Collaboration and Contributions.

This assignment was completed collaboratively, we are equally contributing to the development of the code and the preparation of the report. All steps were discussed and implemented during team meetings. These collaborative sessions ensured that both members actively participated in writing, debugging, and refining the code, as well as drafting and structuring the report.