Project 3: Cancer Diagnosis Using Machine Learning

Objective:
Develop and evaluate machine learning models to predict whether a tumor is **benign (B)** or **malignant (M)** using a dataset of clinical variables.

Dataset Details:

- **Total Samples:** 569
- **Features:** 20 clinical variables
- **Labels:** B (Benign), M (Malignant)
- **Availability:** Dataset is provided on Brightspace.

Requirements:

- Use **Spark MLLib** for model development: Spark MLLib
- Select **two machine learning algorithms** (e.g., Random Forest, etc.) to train and evaluate your models.
- **Compare the performance** of the two models based on evaluation metrics.

Deliverables (Due by 11:59 PM on May 20):

1. **Code**
2. **README File**
3. **Report** containing:
   - Justification for the chosen algorithms
   - Description of your training procedure
     - Data splits
     - Cross-validation methods (optional)
   - Testing results
   - Evaluation using:
     - F1 Score
     - Precision
     - Recall
     - Accuracy (% correct predictions)
   - Comparison and analysis of the two algorithms
   - Discussion on:
     - Limitations of your approach
     - Suggestions for future improvements

Support:
For questions or clarifications, please **email to book a Zoom meeting**.

Grading Rubric:

- **Algorithm Justifications:** 30 points
- **Training Procedure:** 30 points
- **Coding Quality:** 20 points
- **Reporting (including evaluation and discussion):** 20 points

Student:
Dorisa Shehi

Code explanation:

A Spark session is created with optimized settings for memory and performance.

The dataset is read from a CSV file.

The id column is dropped (since it's not useful for modeling).

All feature columns are cast to numeric types.

Columns with all missing values are removed.

Missing values in the remaining columns are imputed using the mean

The diagnosis column (categorical) is converted into numeric labels.

All feature columns are combined into a single feature vector.

The dataset is split into **80% training** and **20% testing** data.

Two classifiers are trained:

- Random Forest with 50 trees.
- Gradient Boosted Trees (GBT) with 50 iterations and depth of 3.
- 3-fold cross-validation is used to evaluate each model during training.

Both models are evaluated on the test set using multiple metrics:

- Accuracy
- F1 Score
- Precision
- Recall

Evaluation scores are printed for comparison.

The Spark session is stopped to release resources.

Justification for the chosen algorithms

For this project, we chose two powerful ensemble learning methods to tackle the classification task.

Random Forest (RF) works by building a group of decision trees and combining their results. It’s great at handling large sets of features and tends to avoid overfitting by averaging the predictions from multiple trees.The ensemble nature helps reduce variance in predictions, critical for medical applications where false negatives can be particularly harmful.

Gradient Boosted Trees (GBT), on the other hand, takes a more strategic approach. It builds one tree at a time, with each new tree trying to fix the mistakes made by the previous ones. This method often leads to higher accuracy, especially in complex tasks.The boosting approach potentially offers improved sensitivity for detecting subtle patterns that may indicate malignancy.

Description of your training procedure

Data Split

To train and evaluate our models fairly, we first split the dataset into 80% for training and 20% for testing, using a fixed random seed (42) to make sure the results are consistent every time the code runs.

Cross-Validation

- Implemented 3-fold cross-validation to:
- Mitigate overfitting risk
- Ensure model generalizability
- Optimize hyperparameters

Testing Results

Model Evaluation:

Accuracy	Random Forest: 0.9535	Gradient Boosting: 0.9767
F1 Score	Random Forest: 0.9532	Gradient Boosting: 0.9766
Precision	Random Forest: 0.9540	Gradient Boosting: 0.9776
Recall	Random Forest: 0.9535	Gradient Boosting: 0.9767

The results clearly show that Gradient Boosting performed better than Random Forest on all major evaluation metrics. This is likely because Gradient Boosting improves predictions step by step, learning from previous mistakes along the way.Both models achieved >96% accuracy, indicating strong predictive power

Evaluation using:

To assess model performance, we used the following metrics:

- **Accuracy**: Measures the percentage of correct predictions.
- **F1 Score**: Balances precision and recall, especially useful when classes are imbalanced.
- **Precision**: Indicates how many of the predicted positives were actually correct.
- **Recall**: Shows how many of the actual positives were correctly identified.

Comparison and analysis of the two algorithms

Performance Analysis

Overall performance: Both algorithms demonstrated excellent classification capabilities with accuracy above 95%

GBT advantage: Gradient Boosting showed consistently superior performance across all metrics, with approximately 2.3 percentage point improvement in accuracy (97.67% vs 95.35%)

Stability: Both models exhibited remarkable consistency across different evaluation metrics, with minimal variation between accuracy, F1, precision, and recall

Algorithmic Differences Impact

Gradient Boosting's sequential correction of errors effectively captured subtle patterns in the dataset, leading to its superior performance

Both algorithms maintained excellent balance between precision and recall, indicating effective classification across both classes

The high F1 scores (95.32% for Random Forest, 97.66% for Gradient Boosting) indicate strong performance regardless of class distribution

While Random Forest was likely faster to train due to its parallel nature, the performance advantage of Gradient Boosting (2.3 percentage points) justifies its additional computational cost

Discussion on:

- Limitations of your approach

  - Limited hyperparameter tuning due to time and performance constraints.
  - Used default settings for many parameters to ensure quicker training time.
  - Spark cluster resources were modest (`2g` executor memory and 8 partitions), which could impact scalability.

- Suggestions for future improvements

  - Perform grid search over more hyperparameters (e.g., `maxDepth`, `minInstancesPerNode`).

  - Use more folds in cross-validation for better generalization.

  - Deploy the model as a real-time prediction service.