DATA MINING PROJECT

Phase 1: Supervised Techniques

Each function of the project is explained comprehensively and the reasons why we used them are fully covered.

Ashok Kumar J (862467018)

Dan Shay (862546326)

November 04, 2024

Contact details:

ajakk002@ucr.edu

dan.shay@email.ucr.edu

GitHub Link: https://github.com/REGATTE/DMT_project

?contentsname?

1 Abstract

In this project, we applied several data mining techniques to the "Wisconsin Breast Cancer Diagnostic" dataset. Since there are more healthy cases than malign cases, the dataset is imbalanced; a problem we solved. We implemented classifiers from scratch, studied the effects of dimensionality reduction; and we also did feature selection, and applied data augmentation with SMOTE to improve classifier performance. We imported the following libraries:

- **Matplotlib**: For visualization purposes.
- **Seaborn**: For visualization purposes.
- **Numpy**: For basic data manipulation.
- **Pandas**: For basic data manipulation.
- **StratifiedKFold**: For cross-validation.
- **F1_score**: For performance metrics.
- **TruncatedSVD**: For dimensionality reduction.

2 Preliminaries

2.1 Data Loading

We fetched the dataset and stored features and target labels. Additionally, since our labels are either "malignant" or "benign", which is a binary approach, 1 and 0 are considered for them respectively.

2.2 Exploratory Data Analysis

In this part, we can see the correlation matrix for the features and visualization of a heatmap to identify features that are correlated, to see if there is redundancy and the possibility of feature selection or other dimensionality reduction strategies.

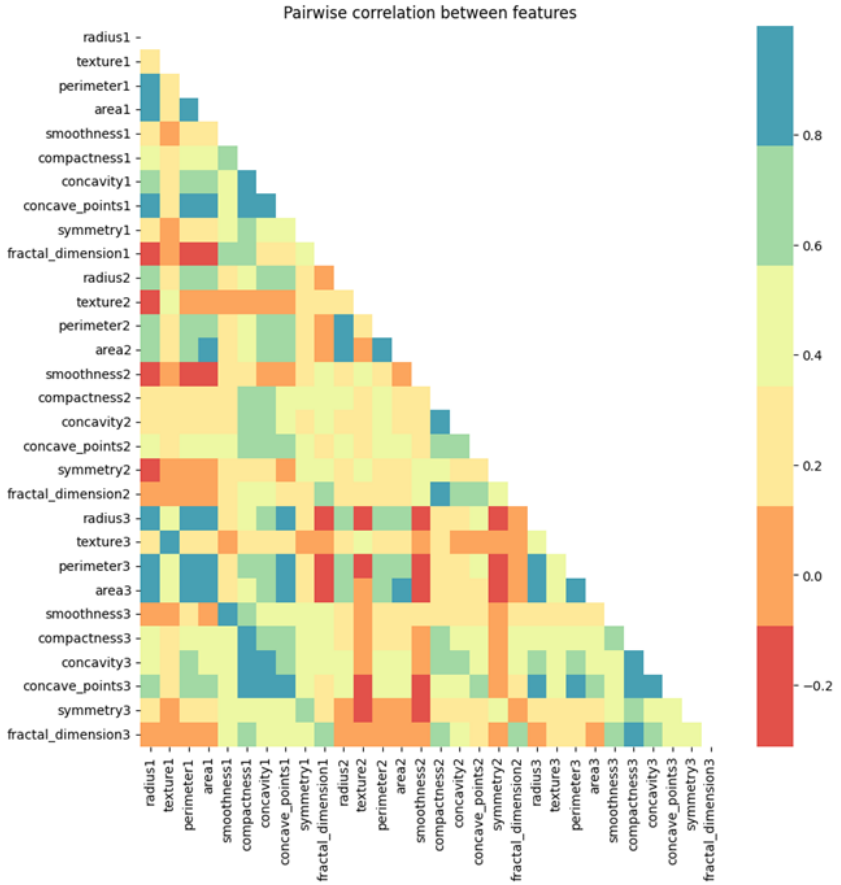

Figure 1: Pairwise correlation between featres

Also, there are 357 benign cases and 212 malign which show our dataset is imbalanced; so we have to balance it to avoid any data bias.

We can see the box plots for each feature separated by being benign or malign, showing the distributions, which is helpful to distinguish between two classes based on their spread and central tendency. For example, features with less overlap between classes might have more decisive power, so will be prioritized during feature selection.

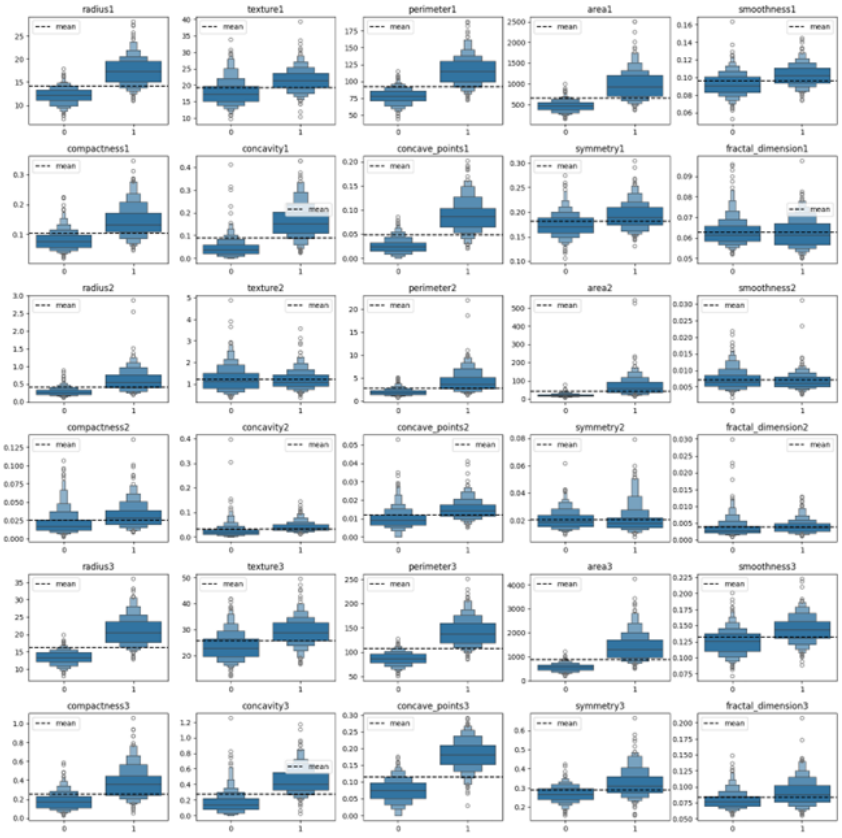

Figure 2: Box Plots

3 Questions

3.1 Q1 – Implementing simple classifiers

We implemented the Decision Tree and Naïve Bayes classifiers from scratch and evaluated them using stratified 10-fold cross-validation, with F1 score as the performance metric.

1. **DecisionTreeClassifier**: This is our Decision Tree classification function with an optional control of depth.

    - We also have defined the **"Entropy"** method measures the impurity of a given set of labels that is useful when deciding to split a decision tree.
    - To find the best feature to split on at each node, we also defined the **"Information_gain"** method that computes the difference in entropy before and after a potential split.
    - We implemented other simple needed methods like **"split"**, **"best_split"**, **"build**$_{tree}$"$whosename fullyrepre$
    - The **"predict_sample"** function traverses the tree from root and moves left to right until it reaches a leaf node to predict for a single instance.

- The **"predict"** function uses "predict_sample" function for every instance in the input data.

2. **NaiveBayesClassifie**r: This is our Naïve Bayes classification function.

   - The "fit" method computes the mean, variance, and class priors for each feature and class. The Gaussian Naïve Bayes formula uses these statistics to calculate probabilities for prediction.
   - The **"guassian_pdf"** method computes the Gaussian Probability Density Function for a given feature value. This function is central to Gaussian Naive Bayes, which models each feature as a Gaussian distribution conditioned on the class label.
   - The **"predict**$_{s}ample"functionallowsforpredictiononasingleinstancebycalculatingtheposteriorprob$
   - The **"evaluate_classifier"** function computes the F1 scores for each fold and returns the mean and standard deviation of F1 scores. This function allows us to estimate the model's performance since the dataset is imbalanced.

After implementing these classifiers, we instantiated both of them and evaluated them using the "evaluate_classifier" method which assesses the F1 score means and standard deviation of each of them. Ultimately, we visualized their bar chart to compare their F1 scores. Since the F1 score for Decision Tree is **0.92** and for Naïve Bayes is **0.91**, we can conclude that **both models have strong performance**.

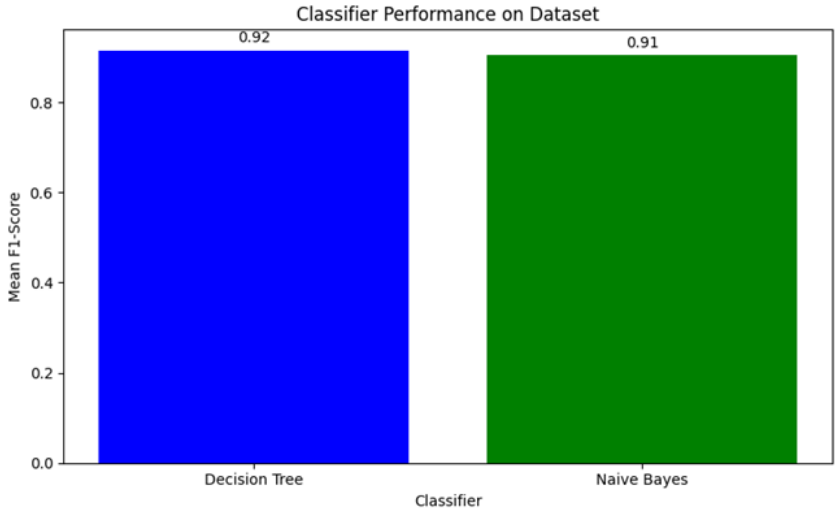

Figure 3: Classifier Performance on Dataset

3.2 Q2 - Dimensionality reduction with the Singular Value Decomposition

In this part, we try to apply SVD to reduce feature space dimensionality and examine the impact on classifier performance that we earlier calculated. These are the steps followed:

1. Reducing dimensionality with SVD, testing various ranks.
2. Evaluating classifiers at each rank using stratified cross-validation.
3. Plotting F1 scores as a function of SVD rand to visualize performance changes.

This is how functions work:

1. The **"apply_svd"** method transforms the training and test sets while ensuring no data leakage happens by only fitting on the training set and then transforming both training and test sets.
2. The **"evaluate_with_svd"** function returns a list of average F1 scores across SVD ranks allowing for analysis of the trade-off between dimensionality reduction and classifier accuracy. More thoroughly, it evaluates the classifier performance over a range of SVD ranks by:
   - Splitting the dataset with cross-validation
   - Applying SVD to each train-test split
   - Training the classifier on reduced-dimensional data
   - Recording F1 scores for each SVD rank

Finally we evaluated the Decision Tree and Naïve Bayes classifiers using the "evaluate_with_svd" function across **[2, 5, 10, 20, 30]** SVD ranks to see how dimensionality reduction impacts the performance of both of the classifiers. The plot is shown below:

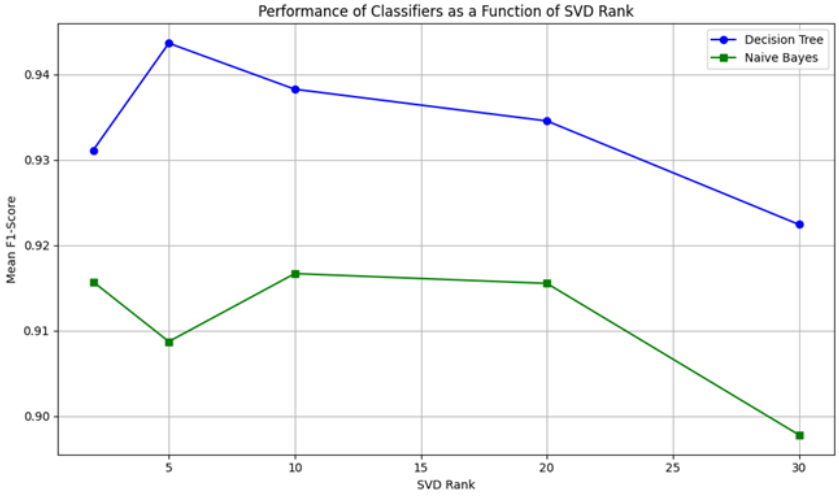

Figure 4: Performance of Classifiers as a Function of SVD Rank

As we can see, while both of the classifiers' performance slightly declines as SVD rank increases, the Decision Tree performs better than the Naïve Bayes.

3.3 Q3 - Feature selection with randomization

The overview of the steps is as follows:

1. For each feature, we created a copy of the dataset with that feature's values randomized to evaluate its importance.
2. Used 5-fold cross-validation to compare model performance on the original dataset versus the randomized dataset for each feature, calculating the F1 score drop as a measure of feature importance.
3. Determined the importance score, by finding the difference of Original & Randomized f1 score.
4. Ranked features by their importance and plotted a bar chart of importance scores from cross-validation and holdout set

This is how functions work:

1. The **"randomize_feature"** creates a copy of the dataset with one randomly chosen feature column. This disrupts the relationship of the feature with the target which allows us to observe the impact on the classifier's performance.
2. The **evaluate_randomized_feature_on_holdout** method trains on the 20% on both the datasets (with and without randomized features) and evaluates both models on the holdout set, outputs the importance score.
3. The **"evaluate_randomized_feature_cv"** method uses 5-fold stratified cross-validation to compute F1 scores, and evaluates the impact of randomizing a feature on model performance using 5-fold cross-validation.
4. The **"feature_importance_ranking"** function calculates feature importance scores using both cross-validation and holdout set methods.

Then to avoid bias, we split the dataset into a 20% sample for feature selection and 80% for evaluating the models after feature selection.

So, we instantiated the "DecisionTreeClassifier" classifier methods that we defined earlier and ranked features for both of them using the 20% sample. And then trained on the 20% sample and evaluated on the 80% holdout data.

Ultimately, the importance scores for each feature on both corss-validation and holdout, are stored into a tupples, and are plotted using a bar-chart.

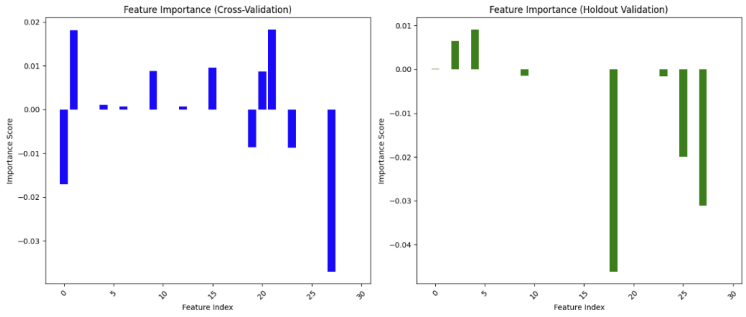

Figure 5: Feature Importance Scores

3.4 Q4 - Data augmentation using SMOTE

The overview of the steps are as follows:

1. Implemented SMOTE from scratch to generate synthetic samples for the minority class, based on a specified number of nearest neighbors.
2. evaluated classifier performance with SMOTE applied at different oversampling ratios (100%, 200%, 300%) and k values (1, 5) using 5-fold cross-validation, calculating the mean F1 score for each configuration.
3. Plotted F1 scores against the oversampling ratios for each k value, showing the effect of SMOTE on model performance for both classifiers.

The **"smote"** function creates synthetic samples based on the following parameters:

1. **x and y**: Feature matrix and labels
2. **minority_class**: Label of minority class to oversample
3. **k**: Number of nearest neighbors to consider for sample generation
4. **oversample_ratio**: Ratio specifying how much to oversample the minority class.

Let's delve deeper into implementing the "smote" function. It separates the minority class samples and calculates the required number of synthetic samples based on the specified oversampling ratio. Then fits a nearest0neighbors model on minority samples and iterates to generate synthetic samples:

1. Randomly select a minority sample
2. Finds its k nearest neighbors

3. Generates a synthetic sample by interpolating between the selected sample and a random neighbor, adding diversity to the augmented dataset.

Then returns the augmented dataset with synthetic minority samples included. Ultimately, we applied the "smote" function to the dataset and then evaluated the Decision Tree and Naïve Bayes classifiers on the augmented data using stratified cross-validation. By assessing the classifiers at various oversampling levels, we can understand how balancing the dataset with synthetic samples affects performance metrics like the F1 score.

This is how functions work:

1. The **"evaluate_smote"** function returns a nested dictionary with F1 scores. It evaluates the classifier performance using different SMOTE oversampling ratios and k values. It runs stratified 5-fold cross-validation for each combination of k and oversample_ratio, and for each fold:
   - SMOTE is applied to the training set to create synthetic samples for the minority class.
   - The classifier is trained on the augmented data and evaluated on the test set.
   - F1 scores are recorded and the mean score is calculated for each k and oversample_ration combination.
2. The **"plot_smote_performance"** function plots the performance across different SMOTE oversampling ratios and k values.

Ultimately, we initialize both classifiers and evaluate each on the SMOTE-augmented data. This is the Performance plot for the Decision Tree:

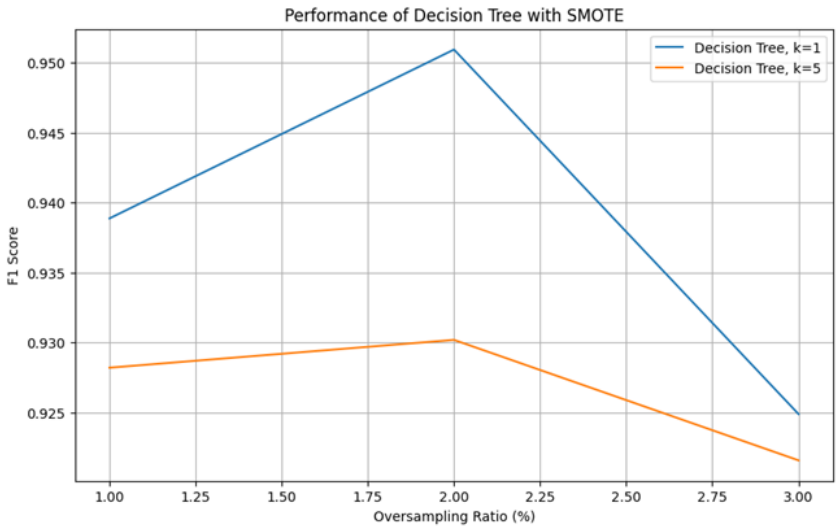

Figure 6: Performance of Decision Tree with SMOTE

It shows the Decision Tree's F1 score peaks around 200% oversampling ratio with k=1, while k=5 yields a different trend. This is the Naïve performance with SMOTE plot:

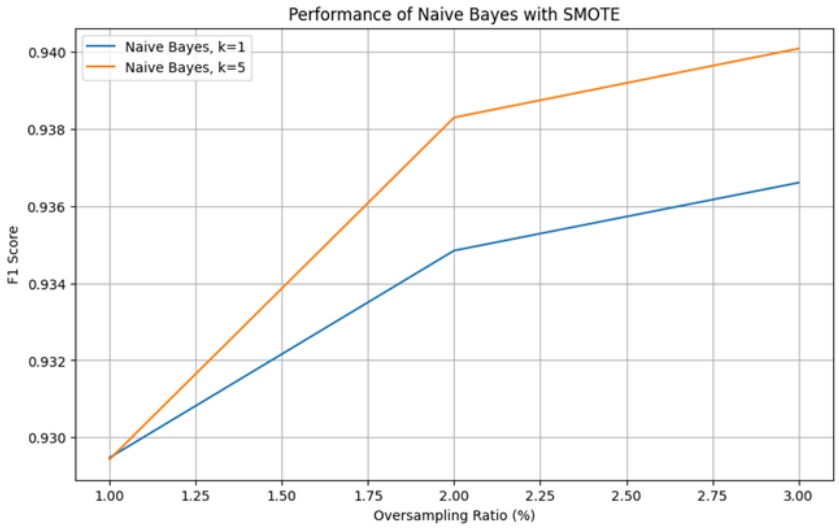

Figure 7: Performance of Naive Bayes with SMOTE

It shows F1 score increases as the oversampling ratios rises, particularly with k=5, indicating the Naïve Bayes classifier benefits from additional synthetic samples that help it to better learn the patterns of the minority class.

4 References

1. https://www.kaggle.com/code/fareselmenshawii/decision-tree-from-scratch
2. https://medium.com/@enozeren/building-a-decision-tree-from-scratch-324b9a5ed836
3. https://github.com/gbroques/naive-bayes/blob/master/naive_bayes/naive_bayes.py
4. https://www.accel.ai/anthology/2022/8/17/svd-algorithm-tutorial-in-python
5. https://www.analyticsvidhya.com/blog/2020/10/feature-selection-techniques-in-machine-learning/
6. https://www.youtube.com/watch?v=oJvjRnuoqQM
7. https://medium.com/@corymaklin/synthetic-minority-over-sampling-technique-smote-7d419696b88c