CSE 601: Data Mining and Bioinformatics Project-3
Classification Algorithms

1. **K Nearest Neighbors**
2. **Decision Tree**
3. **Random Forrest**
4. **Naïve Bayes**
5. **Kaggle Competition**

Professor
Jing Gao

Sai Kumar Aindla (50321372)
Swathi Ravindran (50314300)
Akshara Santharam (50313950)

AIM

1. The aim of this project is to implement: Nearest Neighbor, Decision Tree and Naïve Bayes.
2. Implement Random Forest algorithm based on implementation of Decision Tree.
3. Adopt 10-fold Cross Validation to evaluate the performance of all methods on the provided datasets in terms of Accuracy, Recall, F-1 Measure and Precision.
4. Kaggle Competition to improve the performance using given training features, training labels and testing features.

Evaluation Metrics

The performance can be determined by using True Positives, True Negatives, False Positives, and False Negatives.

Accuracy:

Accuracy is one of the performance measures and it is a ratio of correctly predicted observations to the total observations.

Accuracy = True Positive + True Negative / True Positive + False Positive + False Negative + True Negative

Precision:

Precision is the ratio of correctly predicted positive observations to the total number of positive predicted observations.

Precision = True Positive / True Positive + False Positive

Recall:

Recall is the ratio of correctly predicted positive observations to all the observations in actual class.

Recall = True Positive / True Positive + False Negative

F-1 Measure:

F-1 Measure can be found by finding the mean of precision and recall.

F-1 Score = 2 * (Recall * Precision) / Recall + Precision

Dataset

Dataset contains features and labels (0 or 1). The last column has the labels and remaining columns has the feature values. Features can be nominal or categorical value.

10 fold Cross Validation

The datasets are split into 90% training data and the rest 10% as the testing data. The average accuracy is calculated by using all the folds.

K Nearest Neighbors

When there is a little or no prior knowledge of distribution of data, K-NN is used. K in KNN refers to the number of nearest neighbors. Distance metrics used in KNN is Euclidean distance, Manhattan distance, Humming distance etc. In our implementation we have used Euclidean distance.

Algorithm:

- Decide the value for K.
- Find the distance of the new point to each and every point in the training data, when a new point is found. Find the K nearest neighbors to the new data point.
- For classifying the point, we count the number of data points and the new data point will belong to the class that has a greater number of neighbors.
- The value of new data point will be the average of k neighbors.
- The choice of K will have a major impact on the results of KNN Algorithm. Choosing better value of K will improve accuracy and precision.

Implementation

- File name and the K value is given by the user. Once the file is obtained, it is converted into a numpy array using np.asarray() function.
- Pre-processing of the data is performed and while pre-processing, the categorical data is checked, and columns are stored separately. While pre-processing we also normalize the data using Min Max normalization technique.
- After pre-processing, the data is split according to the 10-fold cross validation technique, where 90% is used for training and 10% is used for testing.
- Then we implement KNN(k, train_p,test_p) Algorithm on the split data.
    - For each train point get the calculate the distance with the test point.
    - Use the k nearest train points labels to label the test point.
- Once the test labels are predicted, we compare it with the actual test labels.
- Performance is checked by using Accuracy, Precision, F-1 Measure and Recall.
- Again, another dataset is picked from the remaining datasets and the same steps as above is performed.

Parameters Description

- K value is the most important while implementing the KNN algorithm.
- Very high and very low K values will result in the misclassification rate.
- Hence, before chosing the K value we need to perform the algorithm multiple times and chose the one with best accuracy. We have performed with k values 9 and 5 for both the datasets.
- Apart from that, we have used distance metric as the Euclidean distance since it gives better results when compared with minkowski and absolute distance measures.

Output

- For project3_dataset1.txt:
    - For k = 9,

```
Enter the Number of Neighbors : 9
K Nearest Neighbors Results for the file project3_dataset1.txt --------------
Accuracy : 96.66353383458647
Precision: 98.54219948849105
Recall : 92.58877591865794
F-1 Measure: 95.29281749376221
```

    - For k = 5,

```
Enter the Number of Neighbors : 5
K Nearest Neighbors Results for the file project3_dataset1.txt --------------
Accuracy : 96.66353383458647
Precision: 97.82791377420536
Recall : 92.80553645799912
F-1 Measure: 95.06992132434624
```

- For project3_dataset2.txt:
    - For k = 9,

```
Enter the Number of Neighbors : 9
K Nearest Neighbors Results for the file project3_dataset2.txt --------------
Accuracy : 68.3950046253469
Precision: 57.8459595959596
Recall : 37.118794947742316
F-1 Measure: 43.274183605762545
```

    - For k = 5,

```
Enter the Number of Neighbors : 5
K Nearest Neighbors Results for the file project3_dataset2.txt --------------
Accuracy : 65.61054579093434
Precision: 51.80656039866566
Recall : 38.71501305711832
F-1 Measure: 43.06612890201819
```

Observations and Analysis

- KNN performs better with project3_dataset1.txt than project3_dataset2.txt.
- Time complexity increases as the number of records increases. To find k values, sorting on these values has to be performed.
- KNN performs well with the first dataset as the features in first dataset are of continuous values as compared to the second dataset.
- Metric chose to calculate the distance might the effect the performance and accuracy of the algorithm.

Advantages

- Robust to noisy training data.
- No training phase involved.

- Can be used in both classification and regression.
- When dataset is very large, effective algorithm is used.
- Complex models are learnt very easily.

Disadvantages

- Too large k or small k could result in misclassification.
- Expensive.
- False intuition might occur which will affect the accuracy in prediction.
- Huge memory is required to run the algorithm.

Decision Tree

A decision tree is a predictive model which is a mapping from observations about an item to conclusions about its target value. In the tree structures, leaves represent classifications (also referred to as labels), non-leaf nodes are features, and branches represent conjunctions of features that lead to the classifications. We have used Gini Index to check the impurity of the node. A decision tree consists of a node, leaf nodes and non-leaf nodes.

- Node: Attribute
- Leaf: Outcome
- Link: Decision rule

Impurity Measurement

Gini Index is used for measurement of the impurity of the node in the tree.

- It is calculated by subtracting the sum of squared probabilities of each class from one. It favors larger partitions and easy to implement whereas information gain favors smaller partitions with distinct values.
- A feature with a lower Gini index is chosen for a split.
- Classification and Regression trees use Gini method for binary splits.

Implementation

- Initially, give the file name in the main function for which you want to run the decision tree.
- In the first step, we convert our categorical data into numerical values. This can be done by creating a dictionary by using keys and values where keys are representing categorical values and values represents the numerical value
- The dataset can be split into 10 parts to do the 10-fold cross validation
- Gini index is found and at each node, we take the split that gives us the lowest Gini index and tree is built on top of the node.
- To implement decision tree, we use recursion. At each internal node we call the split function to find the attribute that gives us the lowest Gini index and left and right subtrees are built.

- To predict the output for a dataset, we traverse from the root node. We traverse the left subtree if the root value is greater than the values in the left subtree and vice versa.
- If the final terminal node is found, we stop the traversal and return the output.

Parameter Description

- Gini Index is used for the calculating the impurity of the node and least Gini index value is the selected as the best position to split the node.
- When Gini Index value of zero is encounter, splitting of the node is stopped and considered to be the target value to be labeled.
- Categorical data is converted in the initial step to numbers to make the classification easy and build the decision tree.

Output

- For project3_dataset1.txt:

```
Decision Tree Results for the file project3_dataset1.txt --------------
Accuracy : 92.43421052631578
Precision: 91.47086922452317
Recall : 88.53501551356514
F-1 Measure: 89.79424987236926
```

- For project3_dataset2.txt:

```
Decision Tree Results for the file project3_dataset2.txt --------------
Accuracy : 65.1063829787234
Precision: 49.90997564526977
Recall : 49.025341617446884
F-1 Measure: 48.56747168914083
```

Observations and Analysis

- Similar to KNN, decision tree algorithm performed better on project3_dataset1 when compared with project3_dataset2 because of the smaller number of features in the later one.
- Accuracy and other measurements will show the results.
- Also, if we have a greater number of features, we can have more options to split the node using the Gini index.

Advantages

- Inexpensive.
- No need of choosing any K value when compared with Knn.
- Simple and easy to interpret and understand.
- Preprocessing and normalization are not required.

- Handles categorical and nominal values.

Disadvantages

- Time taking when need to construct large decision trees.
- Needs to be pruned to avoid the overfitting of the data.
- Calculations can go far more complex compared to other algorithms.
- Require higher time to train the model.
- Inadequate for applying regression and predicting continuous values.

Random Forest

Random forest is a supervised learning algorithm. The "forest" it builds, is an ensemble of decision trees. Random forest has nearly the same hyperparameters as a decision tree. Classification is done very accurately since more than one decision tree is used.

Implementation

- Initially, the file name and number of trees and number of features are given by the user.
- Initially, the categorical data is converted to numbers by creating a dictionary by using keys and values where keys represents categorical values and values will be corresponding numerical values.
- Next, user can enter the number of trees and number of features to split. The decision trees can be built using the same implementation as the above decision tree method when we split the subsampled dataset.
- We select only a subset of the features while splitting the node. For this, we create a random number and select features based on this.
- The above steps are repeated for all the trees and a dictionary is created to store the result of the trees. The key holds the indices of the test data and value holds a list containing the prediction of each tree.
- The final prediction of random forest will be the maximum vote for each index in the dataset.

Parameter Description

- Gini Index is used as the same as the above decision tree results.
- Each individual tree in the random forest splits out a class prediction.
- The class with the most votes becomes our model’s prediction.
- Number of trees to generate is to be chosen via and number of features to split are also to be chosen by keeping in mind about the time complexity to train the algorithm.
- We have ideally chosen number of trees to generate using values 4,8 and number of features to split using the 2,5.

- We have chosen those values because the algorithm is giving better values than normal decision tree and ideally these are used for the better run time.

Output

- For project3_dataset1.txt:
    - For tree count = 4, number of features to split = 2

```
Enter the number of trees : 4
Enter the number of features to split : 2
Random Forest Results for the file project3_dataset1.txt --------------
Accuracy : 94.19486215538846
Precision: 96.79485398278304
Recall : 87.05475129544928
F-1 Measure: 91.5474556031203
```

    - For tree count = 8, number of features to split = 5

```
Enter the number of trees : 8
Enter the number of features to split : 5
Random Forest Results for the file project3_dataset1.txt --------------
Accuracy : 95.0689223057644
Precision: 96.85347985347987
Recall : 90.02721894855382
F-1 Measure: 93.26854878849618
```

- For project3_dataset2.txt:
    - For tree count = 4, number of features to split = 2

```
Enter the number of trees : 4
Enter the number of features to split : 2
Random Forest Results for the file project3_dataset2.txt --------------
Accuracy : 62.76133209990749
Precision: 43.42207792207792
Recall : 30.26132420211367
F-1 Measure: 35.324488337415985
```

    - For tree count = 8, number of features to split = 5

```
Enter the number of trees : 8
Enter the number of features to split : 5
Random Forest Results for the file project3_dataset2.txt --------------
Accuracy : 60.36077705827937
Precision: 38.791269841269845
Recall : 30.30983636246794
F-1 Measure: 33.108101611076435
```

Observations and Analysis

- Random forest performed better than the above decision tree algorithm.
- Also, project3_daatset1 has better accuracy when compared with project3_dataset2.
- As the number of trees input is increased, the running time of the algorithm is increased.

Advantages

- Solve both classification as well as regression problems
- Works well with both categorical and continuous variables
- Automatically handle missing values
- Algorithm is very stable
- Less impacted by noise.

Disadvantages

- Random Forest require much more time to train as compared to decision trees as it generates a lot of trees.
- This algorithm requires much more computational power and resources. On the other hand, decision tree is simple and does not require so much computational resources.
- Misclassification is possible since there are many classes.

Naive Bayes

Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e., every pair of features being classified is independent of each other.
It is a supervised machine learning algorithm and is one of the most interesting probabilistic classifier-based algorithms on Bayes Theorem.
Naïve Bayes algorithm has an assumption that it is most unlikely in real data. But this Naïve Bayes approach works very well on data where this assumption is not applicable.
Naïve Bayes is a classification algorithm for multi-class and binary classification problem. The calculation of probabilities for each hypothesis are simplified to make calculation tractable.

Bayes Theorem is stated using the following equation:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Algorithm

- Naïve Bayes starts off with some initial confidence in the labels 0 and 1. Confidence levels can be adjusted when a new information is available.
- The final confidence levels are the probabilities of labels 0 and 1 and it changes when all the available information is gone through.

Implementation

- File name is inputted by the user. Once the file is obtained, it is converted into a numpy array using np.asarray() function.
- Pre-processing of the data is performed and while pre-processing, the categorical data is checked, and columns are stored separately. While pre-processing we also normalize the data using Min Max normalization technique.

- After pre-processing, the data is split according to the 10-fold cross validation technique, where 90% is used for training and 10% is used for testing.
- The next step after splitting the data is to calculate class posterior probability for continuous data and categorical data.
- To calculate Categorical probability for the categorical columns, at first the class prior probability P(Hi) is obtained from the training data by dividing the number of training sample by total number of training records. The, descriptor prior probability P(X) is found by getting the count of records of the data point by total number of training records. The Descriptor Posterior Probability P(X|Hi) is calculated by obtaining the count of training records of that data point by total number of records of that respective class. In the final step, class posterior probability (Phi|X) is calculated by multiplying the class prior probability P(Hi) and Descriptor Posterior probability P(X | Hi).
- The above steps are repeated for all categorical columns.
- At the end of the implementations, we have values of test records being a zero probability and one probability.
- After iterating, we find the result by multiplying the continuous and categorical probability. We compare it with the actual test labels when the test labels are predicted.
- The performance is found by using Accuracy, Recall, Precision and F-1 measure.
- Again, another dataset is picked from the remaining datasets and the same steps as above is performed.

Parameter Description

- Converting the categorical values to numerical values for the best classification.
- Gaussian density function is used for calculating the probability for the naïve bayes calculation.
- Summarizing the dataset using the two probabilistic statistic parameters mean and standard deviation for the calculation of the probability.

Output

- For project3_dataset1.txt:

```
Naive Bayes Results for the file project3_dataset1.txt --------------
Accuracy : 93.48997493734333
Precision: 91.78709362532891
Recall : 90.44426356826324
F-1 Measure: 91.00318381263209
```

- For project3_dataset2.txt:

```
Naive Bayes Results for the file project3_dataset2.txt --------------
Accuracy : 70.34690101757631
Precision: 57.09275640080594
Recall : 61.59396451501715
F-1 Measure: 58.673964581381455
```

Observations and Analysis

- Performs better for project3_dataset1 because it contains only continuous features and project3_dataset2 has both continuous as well as categorical features.

Advantages

- Easy and fast to predict a test data set.
- When assumption of independent predictors holds true, a Naive Bayes classifier performs better as compared to other models.
- Naive Bayes requires a small amount of training data to estimate the test data. So, the training period is less.
- Handles both continuous and discrete data.

Disadvantages

- Independent predictor assumption.
- Sometimes called as a bad estimator.
- Zero frequency issues can use Laplacian estimator or adding 1 for simple cases to avoid dividing by zero.

Kaggle Competition

Aim of this Kaggle competition is to apply various tricks on top of any classification algorithm (including nearest neighbor, decision tree, Naïve Bayes, SVM, logistic regression, bagging, AdaBoost, random forests) and tune parameters using training data.

Submissions

We have performed 8 submissions in the Kaggle, each with different classification algorithm knn, decision tree, random forest, naïve bernoulli, adaboost, adaboost regressor, gridsearchcv, bagging classifiers.

Normalization

We performed normalization on the training features and testing features before training and predicting the outcomes using preprocessing normalization package.

Implementation, Parameter Setting and Results

For each classification algorithm we have first divided the training set further into training set and validation set. Later we predicted the accuracy, precision for the validation set and based upon those we made our submissions in the Kaggle.

```
#Train and Test Validation Accuracy
X_train, X_val, y_train, y_val = train_test_split(train_features, train_labels, test_size=0.25, random_state=1)
```

1. Knn-

   ```
   knn = KNeighborsClassifier(n_neighbors=13)
   ```

   Using the k value as 13 we got the,
   We got the following results

   ```
   Accuracy for training and validation sets for Knn : 69.0
   F1-score for training and validation sets for Knn : 80.98159509202453
   ```

2. Bagging –

   ```
   bag = BaggingClassifier(n_estimators=8, max_samples=1.0, max_features=1.0)
   ```

   Number of estimators as 8 and max samples as 1.0, max features as 1, we got the following results.

   ```
   Accuracy for training and validation sets for Bagging : 75.0
   F1-score for training and validation sets for Bagging : 82.75862068965517
   ```

3. Decision Tree –

   ```
   tree = DecisionTreeClassifier(criterion = 'gini')
   ```

   Using the Gini Index as the impurity measurement, we got the following results

   ```
   Accuracy for training and validation sets for decision tree : 77.0
   F1-score for training and validation sets for decision tree : 84.35374149659864
   ```

4. Random Forest –

   ```
   rfc = RandomForestClassifier(criterion='gini', n_estimators = 10)
   ```

   Using gini index as the measurement and number of estimators, we got the following results.

   ```
   Accuracy for training and validation sets for random forest : 77.0
   F1-score for training and validation sets for random forest : 85.16129032258064
   ```

5. AdaBoost Classifier –

```
AdBC= AdaBoostClassifier(n_estimators=5, learning_rate=0.2)
```

Using the number of estimator value as 5 and learning rate as 0.2, we got the following results.

```
Accuracy for training and validation sets for Adaboost Classifier : 71.0
F1-score for training and validation sets for Adaboost Classifier : 83.04093567251462
```

6. AdaBoost GridSearchCV –

Used the grid of number of estimator values as [10, 50, 100, 500]

Learning rate – [0.0001, 0.001, 0.01, 0.1, 1.0], number of k folds as 10

```
grid['n_estimators'] = [10, 50, 100, 500]
grid['learning_rate'] = [0.0001, 0.001, 0.01, 0.1, 1.0]
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
model = AdaBoostClassifier()
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv, scoring='accuracy')
```

Results

```
Accuracy for training and validation sets for Grid Search : 75.0
F1-score for training and validation sets for Grid Search : 83.44370860927152
```

7. AdaBoost Regressor (Majority Voting) –

Ran the Adaboost Regressor classifier for two times and took the majority voting from the two predicted outputs od the regressor by comparing each labels. Got the following results as the training and validation accuracy.

```
AdbcR1  = AdaBoostRegressor(n_estimators =1300,loss='exponential',learning_rate=1)
AdbcR2  = AdaBoostRegressor(n_estimators =1300,loss='square',learning_rate=1)
```

Results

```
Accuracy for training and validation sets for Adaboost Regressor : 78.0
F1-score for training and validation sets for Adaboost Regressor : 85.13513513513513
```

8. Naïve Bayes Bernoulli –

```
nb = BernoulliNB(alpha=1e-02, binarize=0.0, fit_prior=True, class_prior=None)
```

Results

```
Accuracy for training and validation sets for Naive Bayes Bernoulli : 81.0
F1-score for training and validation sets for Naive Bayes Bernoulli : 87.24832214765101
```

Improvements and best algorithms

We have performed improvements extensively on Adaboost and Naïve Bayes Bernoulli by tuning the hyperparameters.

1. For the Adaboost we have observed Adaboost Regressor by taking the majority voting of the regressor we have got the F1-score of 85.135 and accuracy of 75.0 for the training and validation set which is far better than the output of normal Adaboost classifier and Adaboost GridSearchCV. Below table shows the better understanding.

Classifier and Hyperparameters	**Results**
AdaBoostClassifier(n_estimators=5, learning_rate=0.2)	Accuracy – 71 Precision – 83.04
model = AdaBoostClassifier() GridSearchCV(estimator=model)	Accuracy – 75 Precision – 83.44
AdaBoostRegressor(n_estimators =1300,loss='exponential',learning_rate=1) AdaBoostRegressor(n_estimators =1300,loss='square',learning_rate=1)	Accuracy – 75.0 Precision – 85.135

Hence by the above table we concluded with to select Adaboost Regressor as one of our submissions among the three submissions in the Kaggle.
Hyperparameters used for both regressors

a. n_estimators =1300, loss='exponential', learning_rate=1.
b. n_estimators =1300,loss='square',learning_rate=1

2. One more algorithm we have performed improvements is the Naïve Bayes Bernoulli which upon setting the correct set of parameters giving the accuracy and precision better than above Adaboost Regressor. Improvements we made on the Naïve Bayes can be shown by the below table.

Naïve Bayes Hyperparameters Tuning	**Results**
BernoulliNB(binarize=1.0, fit_prior=True, class_prior=None)	Accuracy – 71 Precision – 83.04
BernoulliNB(alpha = 1e-02, binarize=0.0, fit_prior=True, class_prior=None)	Accuracy – 81 Precision – 87.24

Hence, we submitted the outputs of the Naïve Bayes Bernoulli with hyperparameters
alpha = 1e-02,
binarize = 0.0

References:

https://machinelearningmastery.com/naive-bayes-classifier-scratch-python/
https://machinelearningmastery.com/adaboost-ensemble-in-python/