CSE 601: Data Mining & Bioinformatics

Classification Project Report

SUBMITTED BY:

BHARAT REDDY (50289569)

MALINI ANBAZHAGAN (50289383)

MONISHA BALAJI (50290962)

Table of Contents

DECISION TREE *3*

ALGORITHM 3

IMPLEMENTATION 3

RESULTS 4

RANDOM FOREST *6*

ALGORITHM 6

IMPLEMENTATION 6

RESULTS 8

K-NEAREST NEIGHBORS *10*

ALGORITHM 10

IMPLEMENTATION 10

RESULTS 11

NAIVE BAYES *13*

ALGORITHM 13

IMPLEMENTATION 14

RESULTS 14

KAGGLE InClass PREDICTION COMPETITION *16*

INTRODUCTION 16

APPROACH 16

CHOSEN CLASSIFIER 16

IMPLEMENTATION 16

RESULTS 17

External References *18*

DECISION TREE

ALGORITHM

- Given: a data set with features and labels.
- Divide the dataset into K-folds of train and test data where K is taken as input from the user.
- Check if the dataset has categorical data. If yes, convert categorical data to continuous using a one-hot encoding.
- Calculate GINI index for each feature in the dataset.
- Create and split the tree based on the minimum GINI index found.
- Stop the tree when all labels in the subset are the same or when all features are used for splitting.
- Predict the labels for test data and then check with the actual label.
- Calculate the evaluation metrics: Accuracy, Precision, Recall, and F-1 measure.

IMPLEMENTATION

- The algorithm is implemented using Python3.
- Dataset, number of folds are taken as input from the user.
- The method **create_fold()** is used to create K-folds of test and train data from the given dataset.
- The method **check_string()** is used to check if the given dataset has any categorical data and return the features that contain categorical/string data.
- The method **one_hot_encoding()** is used to convert categorical data to its equivalent one hot encoding value(continuous/number).
- The method **gini_index()** is used to calculate and return the Gini value for each item in dataset.
- The method **getTreeSplit()** is used to split the tree based on a given split index into the left and right subsets.
- The method **Tree()** is used to get minimum GINI value from all GINI values and return the value, it’s index, left & right subsets.
- The method **createTree()** is used to create Decision Tree based on the root, left and right subsets.
- The method **predict()** is used to predict the label of test data from the decision tree created using train data.
- The method **evaluation()** is used to calculate and return performance metrics (Accuracy, Precision, Recall & F-1 measure) for the algorithm.
- Metrics (Accuracy, Precision, Recall & F-1 measure) are calculated using their respective formulae based on True Positive, True Negative, False Positive & False Negative.
- Data structure (Node) for saving the tree is used to create a decision tree where the topmost node is root and it is split into left and right nodes where each node in the tree will have split value and its index.

- Dictionary is used to store the categorical/string data along with its corresponding encoded value which is used later to print the decision tree.
- Categorical features are converted to its equivalent one hot encoding value and then GINI is calculated.
- Gini value is calculated for all Continuous features.
- The best feature is the one that has the minimum Gini value and this feature is selected to create/split the decision tree. The best feature is selected as the Root. Feature values less than that of the best feature are put in the left subset and all feature values greater than the best feature value are put in the right subset.
- Decision Tree is stopped when:
    - All labels in a subset are the same.
    - All features are used for splitting.

External Libraries used:

- pandas as pd: for reading dataset
- numpy as np: for arrays
- mean from statistics: for calculating mean
- preprocessing from sklearn: for one-hot encoding
- Randrange from random: to get random values in a given range

RESULTS

Below is the table depicting performance metrics for the 2 datasets given. The number of folds is 10.

Performance Metrics	Dataset 1	Dataset 2
Accuracy	**0.9285714285714286**	**0.6543478260869565**
Precision	**0.913271582816609**	**0.5230859199280252**
Recall	**0.8943723868723868**	**0.4841275369797041**
F-1 Measure	**0.9029725845210181**	**0.49258343922415115**

The performance of Dataset 1 is found to be higher than that of Dataset 2. Possible reasons for higher performance for Dataset 1 when compared to Dataset 2 are:

- Dataset 1 does not have any categorical features and Dataset 2 has one categorical feature.
- Dataset 1 has more data to train the decision tree compared to Dataset 2.

Pros for Decision Tree:

- Decision Tree considers all possible outcomes of a particular decision.
- Decision Tree can be expressed in a readable form for users.
- Decision Tree is not affected by outliers and hence normalization of data is not required.
- Decision Tree can generate rules

Cons for Decision Tree:

- Decision Tree has low bias and high variance(overfitting), i.e., performance on new data might not be as good as seen on training data.
- Decision Tree especially a large one can be convoluted and have limited value.
- Accuracy may decrease as the depth of the tree increases.
- Small changes in data can result in a big change in the model.

RANDOM FOREST

ALGORITHM

- Given: a data set with features and labels.
- Divide the dataset into K-folds of train and test data where K is taken as input from the user.
- Take input from the user for the number of trees to be created
- Repeat the below steps until the given number of trees are created:
  - Check if the dataset has categorical data. If yes, convert categorical data to continuous using a one-hot encoding.
  - For each fold, create bag datasets (equal to the number of trees)
  - Bagging involves creating datasets by randomly selecting data from training data which has the same length as that of training dataset but each time will have different individual data
  - For each tree, randomly select a smaller subset of features (square root of the total number of features in dataset)
  - Calculate GINI index for each feature in the dataset.
  - Create and split the tree based on the minimum GINI index found.
  - Stop the tree when all labels in the subset are the same or when all features are used for splitting.
  - Predict the labels for test data in each tree.
- Select the predicted label by taking the maximum count (majority voting) and then check with the actual label.
- Calculate final evaluation metrics: Accuracy, precision, Recall and F-1 measure by taking the mean of evaluation metrics calculated for each tree.

IMPLEMENTATION

- The algorithm is implemented using Python3.
- Dataset, number of folds & number of trees are taken as input from the user.
- The method **create_fold()** is used to create K-folds of test and train data from the given dataset.
- The method **create_bag()** is used to create a bagging dataset from the training dataset. Each bag might contain repeated data from the original dataset.
- The method **check_string()** is used to check if the given dataset has any categorical data and return the features that contain categorical/string data.
- The method **one_hot_encoding()** is used to convert categorical data to its equivalent one hot encoding value(continuous/number).
- The method **gini_index()** is used to calculate and return Gini value for each item in dataset.
- The method **getTreeSplit()** is used to split the tree based on a given split index into the left and right subsets.
- The method **Tree()** is used to get minimum GINI value from all GINI values and return the value, it's index, left & right subsets.

- The method **createTree()** is used to create Decision Tree based on the root, left and right subsets.
- The method **predict()** is used to predict the label of test data from the decision tree created using train data.
- The method **evaluation()** is used to calculate and return performance metrics (Accuracy, Precision, Recall & F-1 measure) for the algorithm.
- Metrics (Accuracy, Precision, Recall & F-1 measure) are calculated using their respective formulae based on True Positive, True Negative, False Positive & False Negative.
- Data structure (Node) for saving the tree is used to create a decision tree where the topmost node is root and it is split into left and right nodes where each node in the tree will have split value and its index.
- Dictionary is used to store the categorical/string data along with its corresponding encoded value which is used later to print the decision tree.
- Categorical features are converted to its equivalent one hot encoding value and then GINI is calculated.
- Gini value is calculated for all Continuous features.
- The best feature is the one that has the minimum Gini value and this feature is selected to create/split the decision tree. The best feature is selected as the Root. Feature values less than that of the best feature are put in the left subset and all feature values greater than the best feature value are put in the right subset.
- Decision Tree is stopped when:
    - All labels in a subset are the same.
    - All features are used for splitting.
- Repeat the below steps until the number of given trees are implemented:
    - Check if the dataset has categorical data. If yes, convert categorical data to continuous using one-hot encoding.
    - For each fold, create bag datasets (equal to the number of trees)
    - Bagging involves creating datasets by randomly selecting data from training data which has the same length as that of training dataset but each time will have different individual data
    - For each tree, randomly select a smaller subset of features (square root of the total number of features in dataset)
    - Calculate GINI index for each feature in the dataset.
    - Create and split the tree based on the minimum GINI index found.
    - Stop the tree when all labels in the subset are the same or when all features are used for splitting.
    - Predict the labels for test data in each tree.
- Select the predicted label by taking the maximum count (majority voting) and then check with the actual label.
- Calculate final evaluation metrics: Accuracy, precision, Recall and F-1 measure by taking the mean of evaluation metrics calculated for each tree.

External Libraries used:

- pandas as pd: for reading dataset
- numpy as np: for arrays
- mean from statistics: for calculating mean
- preprocessing from sklearn: for one hot encoding
- Randrange from random: to get random values in a given range
- sqrt from math: for calculating square root

RESULTS

In order to determine the optimal number of Trees, we tried from 10 to 50 trees and we found that the best results for Dataset 1 were when we set the number of trees to 40 and for Dataset 2 was when we set the number of trees to 50.

For Dataset 1, metrics for Random Forest is better than Decision Tree metrics. However, for Dataset 2, recall reduces in Random Forest compared to Decision Tree as Random Forests reduce variance as well as bias.

Dataset 1

No. of Trees	10	20	30	40	50
Accuracy	0.95	0.95	0.9535	0.9625	0.9553
Precision	0.9544	0.9392	0.9754	0.9652	0.9463
Recall	0.91	0.9124	0.8997	0.9330	0.9206
F-1 Measure	0.9299	0.9239	0.9332	0.9479	0.9326

Dataset 2

No. of Trees	10	20	30	40	50
Accuracy	0.6760	0.6760	0.6869	0.6934	0.7043
Precision	0.6027	0.6259	0.5666	0.5816	0.5826
Recall	0.4114	0.4149	0.3672	0.4418	0.4498
F-1 Measure	0.47951	0.4891	0.4286	0.4911	0.5024

Below is the table depicting the best performance metrics for the 2 datasets given. The number of folds is 10. The number of Decision Trees per fold is 40 for Dataset 1 & 50 for Dataset 2.

Performance Metrics	Dataset 1	Dataset 2
Accuracy	0.9625	0.7043478260869566
Precision	0.9652210144927537	0.5826053113553114
Recall	0.9330931029754559	0.44989634247761184
F-1 Measure	0.9479679031589048	0.5024285556903534

The performance of Dataset 1 is found to be higher than that of Dataset 2. Possible reasons for higher performance for Dataset 1 when compared to Dataset 2 are:

- Dataset 1 does not have any categorical features and Dataset 2 has one categorical feature.
- Dataset 1 has more data to train the decision tree compared to Dataset 2.

Pros for Random Forest:

- Random Forest improves overall accuracy.
- Random Forest reduces variance by using multiple trees.
- Random Forest reduces overfitting issues seen in Decision Tree.

Cons for Random Forest:

- Random Forest is more complex than Decision tree
- Random Forest is computationally expensive.
- Using Random Forest, it is hard to visualize the model or understand why it predicted something.
- It will be slower if a large number of trees are generated.

K-NEAREST NEIGHBORS

ALGORITHM

- Given: a data set with features and labels.
- Divide the dataset into K-folds of train and test data. which is taken as input from the user.
- Take input from the user for the number of neighbors (K) to be considered.
- Repeat the below steps to predict the output class:
    - Check if the dataset has categorical data. If yes, convert categorical data to continuous using a one-hot encoding.
    - Once the dataset is set, rescale it to fit in the range of 0 to 1.
    - After normalizing the dataset, for each sample in the test set, compute the distance of the sample from all other samples in the train dataset.
    - Sort the distances to retrieve K number of closest neighbors.
    - The neighbors' output labels are taken into consideration. Predict as output the label that occurs the most number of times amongst the K neighboring samples for the test sample.
    - Repeat for all test samples and predict output for each.
- Check the predicted labels with the actual labels.
- Calculate final evaluation metrics: Accuracy, precision, Recall and F-1 measure by taking the mean of evaluation metrics calculated.

IMPLEMENTATION

- The algorithm is implemented using Python3.
- Dataset, number of folds & number of neighbors are taken as input from the user.
- The method **create_fold()** is used to create K-folds of test and train data from the given dataset.
- The method **check_string()** is used to check if the given dataset has any categorical data and return the features that contain categorical/string data.
- The method **one_hot_encoding()** is used to convert categorical data to its equivalent one hot encoding value(continuous/number).
- The method **eucl_dist()** is used to calculate the Euclidean distance between any two samples that are given as input.
- The method **predict_class()** is used to retrieve the K closest neighbors for the given test sample by sorting the computed distances. The predicted label is given by taking the maximum count (majority voting) of the labels of the neighboring samples.
- The method **KNN()** is used to predict the labels of the test samples.
- The method **evaluation()** is used to calculate and return performance metrics (Accuracy, Precision, Recall & F-1 measure) for the algorithm.
- Metrics (Accuracy, Precision, Recall & F-1 measure) are calculated using their respective formulae based on True Positive, True Negative, False Positive & False Negative.

External Libraries used:

- pandas as pd: for reading dataset
- numpy as np: for arrays
- mean from statistics: for calculating mean
- preprocessing from sklearn: for one-hot encoding
- Randrange from random: to get random values in a given range
- sqrt from math: for calculating square root

RESULTS

Given two datasets, each is validated by varying the value of K i.e. the number of Neighbors from 5 to 25. The corresponding results are recorded below and it shows that the highest accuracy for Dataset 1 is reached at K=20 whereas for Dataset 2, it is reached at K=10.

Dataset 1

No. of Neighbors	**5**	**10**	**12**	**14**	**16**	**18**	**20**	**22**
Accuracy	**0.93214**	**0.92142**	**0.89642**	**0.925**	**0.93928**	**0.9107**	**0.9375**	**0.92142**
Precision	**0.94041**	**0.97648**	**0.93414**	**0.92392**	**0.95488**	**0.9335**	**0.9589**	**0.94810**
Recall	**0.86975**	**0.80908**	**0.76131**	**0.86122**	**0.87744**	**0.8198**	**0.8415**	**0.84851**
F-1 Measure	**0.90058**	**0.88141**	**0.83583**	**0.88861**	**0.91374**	**0.8703**	**0.8928**	**0.89251**

Dataset 2

No. of Neighbors	**5**	**10**	**12**	**14**	**16**	**18**	**20**	**22**
Accuracy	**0.63478**	**0.65652**	**0.65**	**0.64130**	**0.62173**	**0.6347**	**0.6195**	**0.65217**
Precision	**0.38342**	**0.58904**	**0.55308**	**0.43357**	**0.42407**	**0.4126**	**0.5327**	**0.57283**
Recall	**0.29009**	**0.14951**	**0.16742**	**0.16504**	**0.10967**	**0.1511**	**0.1389**	**0.14094**
F-1 Measure	**0.31972**	**0.22651**	**0.24316**	**0.22533**	**0.16641**	**0.2123**	**0.2074**	**0.21601**

Below is the table depicting the best performance metrics for the 2 datasets given. The number of folds is 10. Number of Neighbors chosen i.e. K is 20 for Dataset1 & 10 for Dataset2.

Performance Metrics	Dataset 1	Dataset 2
Accuracy	**0.9375**	**0.65652**
Precision	**0.9589**	**0.58904**
Recall	**0.8415**	**0.14951**
F-1 Measure	**0.8928**	**0.22651**

The acquired results show us that the evaluation metrics are higher for Dataset 1 when compared to Dataset 2. This could be due to the absence of categorical fields in Dataset 1.

Pros for K-Nearest Neighbors:

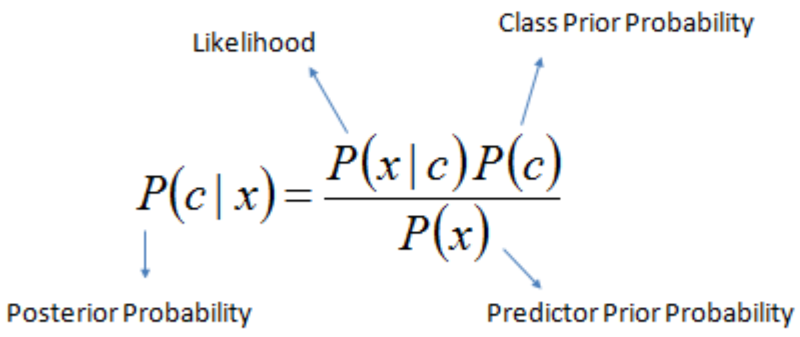
- KNN is simple to implement and also very intuitive.
- It does not involve explicitly building any model. Hence no training steps involved.
- KNN extends its application to both Classification and Regression.
- It helps to easily implement multi-class problems as well.

Cons for K-Nearest Neighbors:

- Although it might work well on small datasets, as the size of the dataset increases, the efficiency of the algorithm decreases.
- It does not work well with imbalanced datasets.
- The value of k has to be chosen with thought.

NAIVE BAYES

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.


Likelihood
Class Prior Probability
$P(c \mid x) = \frac{P(x \mid c)P(c)}{P(x)}$
Posterior Probability
Predictor Prior Probability

$$P(c \mid \mathrm{X}) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Where,

- *P*(*c*|*x*) is the posterior probability of *class* c given *predictor* (*features*).
- *P*(*c*) is the probability of *class*.
- *P*(*x*|*c*) is the likelihood which is the probability of *predictor* given *class*.
- *P*(*x*) is the prior probability of *predictor*.

ALGORITHM

- Given: a data set with features and labels.
- Divide the dataset into K-folds of train and test data where K is taken as input from the user.
- Check if the dataset has categorical data. If yes, convert categorical data to continuous using a one-hot encoding.
- Separate the training data by class/label.
- Summarize the dataset: Calculate the mean and standard deviation of the dataset.
- Summarize data by class: Summarize the separated dataset by the label.
- Calculate the Gaussian probability density function: Calculate the probability or likelihood
- Calculate the class probabilities.
- Calculate the evaluation metrics: Accuracy, Precision, Recall, and F-1 measure.

IMPLEMENTATION

- The algorithm is implemented using Python3.
- Dataset, number of folds are taken as input from the user.
- The method **create_fold()** is used to create K-folds of test and train data from the given dataset.
- The method **check_string()** is used to check if the given dataset has any categorical data and return the features that contain categorical/string data.
- The method **one_hot_encoding()** is used to convert categorical data to its equivalent one hot encoding value(continuous/number).
- The method **summarize_dataset()** is used to get the mean and standard deviation.
- The method **summarize_by_class() is** used to get the mean and standard deviation of columns of training data using summarize_dataset().
- The method **class_probability()** is used to calculate the Gaussian probability of the dataset of each class.
- The method **predict_class()** is used to get the maximum probability of the dataset of each class from class_probability().
- The method **naive_bayes()** is used to predict the labels for the test samples.
- The method **evaluation()** is used to calculate and return performance metrics (Accuracy, Precision, Recall & F-1 measure) for the algorithm.

External Libraries used:

- pandas: for reading the dataset
- numpy as np: for arrays
- mean from statistics: for calculating mean
- preprocessing from sklearn: for one-hot encoding
- Randrange from random: to get random values in a given range
- sqrt from math: for calculating the square root
- exp from math: for calculating the exponent
- pi from math: to use pi value
- reader from csv: to read CSV files

RESULTS

Given two datasets, each is validated by varying the value of fold i.e k_fold. The results are displayed below for k_fold =10. The first table compares the evaluation metrics of Dataset 1 and Dataset 2.

Dataset	Accuracy	Precision	Recall	F-1
Dataset 1	0.9411	0.9287	0.8892	0.9081
Dataset 2	0.7021	0.5324	0.6316	0.5621

The second table shows a comparison in terms the confusion matrix parameters.

Dataset	TP	TN	FP	FN
Dataset 1	20	34	0	2
Dataset 2	8	28	7	3

Pros for Naive Bayes Classification

- It is easy and fast to predict the class of the test data set. It also performs well in multi-class prediction.
- Naive Bayes classifier performs better compared to other models for independent data like logistic regression and you need less training data.
- It performs well in case of categorical data compared to numerical data. For numerical data, normal distribution is assumed.

Cons for Naive Bayes Classification

- If the categorical variable has a category (in the test data set), which was not observed in the training data set, then the model will assign a 0 (zero) probability and will be unable to make a prediction.
- Another limitation of Naive Bayes is the assumption of independent predictors.
- Naive Bayes is also known for bad estimation.

KAGGLE InClass PREDICTION COMPETITION

INTRODUCTION

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

APPROACH

- We have tried different classifiers and checked which classifier gives the maximum accuracy.
- We tried different parameter-tuning approaches, tried normalizing the data and scaling the data
- Under Random Forest, different parameters were changed to see if accuracy improved.
    - Extracted **feature_importances_** from RandomForestClassifier model. Selected the top features and modified the train_features and test_features datasets to include only these features. And then ran the model again to predict the labels. However, Accuracy decreased. Most likely reason being that most features importance was evenly divided. Top most feature was 0.5 and 93 features had 0.1 importance.
    - **Adaboosting** was also tried along with RandomForestClassifier model and this also didn't yield in a better result.

CHOSEN CLASSIFIER

- The chosen classifier is Random Forest.
- Random forests achieve competitive predictive performance and are computationally efficient to train and test.

IMPLEMENTATION

- Read the input files which are *trian_features.csv, test_features.csv, trian_label.csv*
- Used sklearns Random Forest Classifier.
- Train/Fit the model with training dataset and labels
- Predict the class with testing data using the model.
- The parameter tuning was done on the following parameters of the classifier:
    - **n_estimators**
    - **bootstrap**
    - **max_features**

    - **max_depth**
    - **min_samples_leaf**
    - **min_samples_split**
- **n_estimators** is the number of trees to be built in the forest. By default, it is set to 10.
- **bootstrap** if set to true, bootstrap samples of the dataset will be used to build trees. If set to false, whole dataset will be used to build trees. By default, bootstrap is set to true.
- **max_features** is the number of features to consider when looking for best split. By default, it is set as square root of total number of features in dataset.
- **max_depth** is the maximum depth of the tree. By default, it is set to None which means that nodes are expanded until all leaves are pure or until all leaves contain less than **min_samples_split** samples.
- **min_samples_leaf** is the minimum number of samples to be present in each of left and right branches after split at any depth. By default, it is set to 1.
- **min_samples_split** is the minimum number of samples required to split an internal node. By default, it is set to 2.
- With trial and error method, we found the values that gave a good accuracy:
    - **n_estimators = 400**
    - **bootstrap = False**
    - **max_features = 30**
    - **max_depth = None**
    - **min_samples_leaf = 2**
- Using the default values of some of these parameters gave more accuracy:
    - **max_depth***:* various values were tried however default value of None returned the best result.
    - **min_samples_split***:* various values were tried, however default value of 2 returned the best result.
- Extract the labels predicted for testing data and append it to label field of *submission.csv* file.

RESULTS

- All the classifiers used are from sklearn library.
- Gaussian Naive Bayes with bagging: The accuracy yielded was 0.82397. The Gaussian Naive Bayes without bagging yielded 0.82889 accuracy.
- The KNN with bagging yielded an accuracy of 0.78810.
- The Decision tree yielded accuracy 0.85618
- The Random Forest gave an accuracy of 0.86805. This classifier yielded the best result for the given test data.

External References

1. https://machinelearningmastery.com/implement-decision-tree-algorithm-scratch-python/
2. https://machinelearningmastery.com/implement-random-forest-scratch-python/
3. https://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/
4. https://www.fromthegenesis.com/pros-and-cons-of-k-nearest-neighbors/
5. https://machinelearningmastery.com/naive-bayes-classifier-scratch-python/
6. https://towardsdatascience.com/all-about-naive-bayes-8e13cef044cf
7. https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74
8. https://www.analyticsvidhya.com/blog/2015/06/tuning-random-forest-model/