**NAÏVE BAYES**

**PROJECT-3**

**(CSE-601)**

**SWAPNIKA PEMMASANI | swapnika(50289464)**

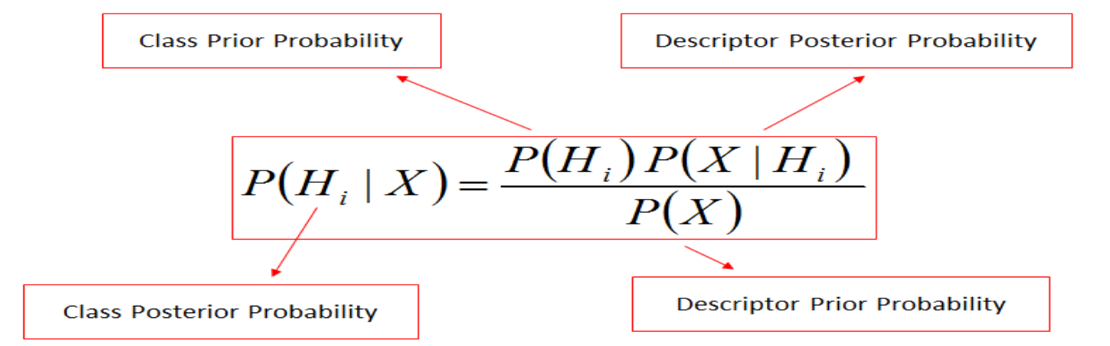
**LAKSHMI VENNELA VEERISETTY | lveerise(50286879)**

**ASISH KAKUMANU | asishkak(50288695)**

Naive Bayes is a probabilistic classification algorithm based on Bayes’ Theorem. Naïve Bayes calculates the class posterior probability in order to classify the given data. It assumes all the attributes are independent and all of the given features have equal probability of occurrence. Naïve Bayes works efficiently on large datasets and is also easy to implement.

Bayes Theorem:

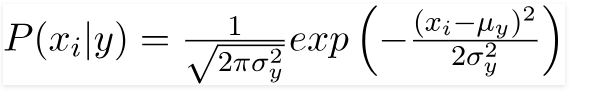
There are various types of Naïve Bayes which are Bernoulli (Binary Features), MultiNomial (Discrete Counts) and Gaussian (Continuous Variables). Bayes Theorem calculates the Class Posterior probabilities of given data with independent variables and classify the data into the class with highest Posterior probability.



**Implementation of Naïve Bayes:**

We have Implemented Naïve Bayes classifier in Python. Our Program performs 10-fold cross validation by choosing different training data and testing data in every iteration and outputs the average accuracy, precision, recall and F-Measure after the number of iterations proportional to the number of Folds.

* 1. Read the input data from the file.
  2. Identify all columns with categorical values and save the indices in a list.
  3. Using the 10-fold cross validation split the data into training and testing data with test data changing in every iteration.
  4. Calculate the class prior probabilities of each class in the training data.
  5. Calculate the Mean and Standard deviation of numerical attributes in the training data for each class.
  6. Calculate the Descriptor posterior probability for numerical attributes of the given test data for each class using probability density function, mean and standard deviation calculate din the previous step.



* 1. Calculate the Descriptor posterior probability for Categorical attributes of the given dataset. If there are no categorical attributes in the data, then probability is considered as One.
  2. Calculate the Posterior probabilities of each class by multiplying class prior probability and descriptor prior probability of each class.
  3. Classify the data to that class which has highest class posterior probability.
  4. Calculate accuracy of the classification by comparing the obtained results with original class of the test data.

**Results:**

**1. project3\_dataset1.txt:**

|  |  |
| --- | --- |
| Average Accuracy: 93.6% | Average Precision: 92.2% |
| Average Recall: 90.44% | Average F-Measure: 91.2 |

**2. project3\_dataset2.txt:**

|  |  |
| --- | --- |
| Average Accuracy: 70.3% | Average Precision: 57.1% |
| Average Recall: 61.5% | Average F-Measure: 58.67 |

**Result Analysis:**

1. By comparing the average results of our model on two datasets we can say that our model performed well on dataset1 when compared with dataset2 which has the accuracy of 70.2%. The reason what we think for the decrease in accuracy of the model is, dataset1 contains only continues values whereas dataset2 contains both continuous and categorical values. This could have been a reason for decrease in accuracy of the model.
2. One more important thing that could be the reason for decrease in accuracy of the model is consideration of independence among all the attributes of the data. There might be some correlated attributes in dataset2 which could have resulted in decrease in accuracy.

**Pros**:

1. Easy to implement and works well on large datasets.
2. Naive Bayes classifier has the performance which is comparable to that of Decision trees.
3. Works on both continuous and categorical data.

**Cons:**

1. Naïve Bayes assumes all the attributes are independent and doesn’t work well on datasets with correlated data.
2. Naïve Bayes fails when the test data has new labels that were not present in the training data.

**K-NEAREST NEIGHBORS**

K- Nearest Neighbors is classification algorithm in which an object is assigned to class or label based on its k-nearest neighbors. The object is assigned to a class which is most common among its K neighbors. It is called lazy learning Algorithm because all computation is deferred until classification.

**Implementation of K-NN:**

We have Implemented Naïve Bayes classifier in Python. Our Program performs 10-fold cross validation by choosing different training data and testing data in every iteration and outputs the average accuracy, precision, recall and F-Measure after the 10 iterations.

1. Choose the value of K, Where K is the number of nearest neighbors of unclassified label to compare and assign.
2. Read the data from the file and divide it into training and test datasets.
3. Take an unclassified data set from testing data and calculate the Euclidean distance from all the points in training data to this point.
4. Pick top K closest neighbors and check the labels of neighbors.
5. Assign the label which occurs in majority to our training dataset.
6. Repeat the same process for all the values in test data.
7. calculate the performance of model by comparing the assigned to original label of test data.
8. The same process is iterated for 10 iterations and average accuracy is calculated as average of accuracies in all iterations.

In K-NN Algorithm choosing K value is important because:

1. If K value is too low algorithm becomes sensitive to noise and it has very less information to classify the algorithm.
2. If K value is too large neighborhood may include points from other classes. i.e. Large amount of un-necessary information is given to algorithm.

**Results:**

For project3\_dataset1.txt:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| K Value | Average Accuracy | Average Precision | Average Recall | Average F-Measure |
| 3 | 96.64% | 98.06% | 92.26% | 0.950 |
| 5 | 96.34% | 96.67% | 92.83% | 0.946 |
| 10 | 97.19% | 98.90% | 93.06% | 0.96 |
| 15 | 97.19% | 98.91% | 93.06% | 0.958 |
| 20 | 95.97% | 98.858% | 89.6% | 0.93 |

For project3\_dataset2.txt

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| K Value | Average Accuracy | Average Precision | Average Recall | Average F-Measure |
| 3 | 64.77% | 53.482% | 48.69% | 0.506 |
| 5 | 63.69% | 51.76% | 43.14% | 0.46 |
| 10 | 69.29% | 63.09% | 42.99% | 0.508 |
| 15 | 68.53% | 62.73% | 40.23% | 0.488 |
| 20 | 68.65% | 65.65% | 41.01% | 0.502 |

**Result Analysis:**

1. K-NN provides very accuracy on dataset1 when compared to dataset2 because dataset1 contains only continuous data but dataset2 contains both continuous and Categorical data. This implies that K-NN does not perform well on categorical data or the data which contains both categorical and continuous data.
2. Results show that performance of algorithm is low when the k value is too low or too high. So k value should never be too high or too low.

**Pros:**

1. K-NN is easy to implement and robust to noise training data.
2. It works well on large datasets.

**Cons:**

1. Choosing the value K is difficult.
2. Normalization of data needs to be carried out to avoid domination of distance measures by any attributes.

**DECISION TREES**

A decision tree is a tool that uses tree-like graph or model of decisions and their consequences. It is a flowchart like structure. Each internal node represents a test on attribute and each branch represents outcome of the test, and each leaf node represents a class label. Paths from root to leaf represent classification rules. These models can also map non-linear relationships. Can solve any kind of problem.

proportion = count(class\_value)/count(rows)

Gini\_index = (1-sum(proportion\*proportion))\*(group\_size/total\_samples)

Gain = currgini - (probleft \* gini(leftClass)) - (probRight \* gini(rightClass))

**Implementation of Algorithm:**

This algorithm mainly involves four steps after pre-processing the data. These include computing Gini index, creating a split, building a tree, and finally making a prediction.

1. First, the given dataset is split into training and testing datasets using 10-fold validation.
2. Training dataset is used to train the model and to construct a decision tree and then classification results are calculated using the testing data.
3. Each column in the dataset represents an attribute. Records based on all values of a column are taken and gini index is computed.
4. The algorithm is performed on the column and the value which provides minimum GINI score is taken as it provides maximum gain and is considered as split point.
5. Now we iterate over each row, checking if attribute value is below or above split value and assign it to left or right group respectively.
6. The above steps are recursively called for both left and right groups of the node.
7. In this way we construct a decision tree. At any stage, if all the records belong to same class, then it is made a leaf node with a particular label for that class.
8. This algorithm is repeatedly called for each subset until it cannot be split further. We will be unable to continue splitting when all rows belong to one group as we will have no records to split on one side or another.
9. Finally, based on the number of iterations, accuracy from each iteration is taken and average accuracy is calculated for the entire dataset.

**Results**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DataSet | Avg-Accuracy | Avg-Precision | Avg-Recall | F-Measure |
| project3\_dataset1.txt | 91.920 | 89.174 | 89.026 | 0.888 |
| project3\_dataset2.txt | 59.306 | 41.105 | 45.054 | 0.4231 |

**Pros:**

1. Are simple to understand and interpret.
2. Can handle both numerical and categorical data. Can also handle multi-output problems.
3. It implicitly performs feature selection.
4. Nonlinear relationships between parameters do not affect tree performance.

**Cons:**

1. It can result in overfitting.
2. Can be unstable because small variations in the data might result in a completely different tree being generated.

3. Can create biased trees if some values dominate.

**RANDOM FORESTS**

Random forests are ensemble learning method for classification, regression and other tasks. It operates by constructing a multitude of decision trees at training time and outputs the class that is the mode of the classes in case of classification. An ensemble method is a technique that combines predictions from multiple algorithms to make accurate predictions than any individual model.

**Bagging**: It is a machine-learning ensemble technique designed to improve the stability and accuracy of algorithms used in classification. It also reduces variance and helps to avoid overfitting.

**Implementation of Algorithm:**

1. The entire dataset is split into training and testing sets using 10-fold validation.
2. Training set is used to train the model to construct a decision tree.
3. Choose T – number of trees in each node.
4. We use **bagging** where we choose a training set N times with replacement from the training set.
5. For each node, we randomly select m features (m < M, where M is total number of features) and calculate the best split.
6. The we take the majority voting among all the trees to decide a label for a record.
7. Finally, the accuracy from each iteration is taken to find the average accuracy for the whole dataset.

**Results**:

For Dataset1: project3\_dataset1.txt

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number of Trees | Avg Accuracy | Avg Precision | Avg Recall | Avg F-Measure |
| 5 | 94.548 | 94.267 | 90.527 | 0.9224 |
| 10 | 95.256 | 98.434 | 88.945 | 0.9335 |
| 15 | 93.151 | 96.323 | 84.946 | 0.901 |

For Dataset2: project3\_dataset2.txt

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number of Trees | Avg Accuracy | Avg Precision | Avg Recall | Avg F-Measure |
| 5 | 65.592 | 49.365 | 39.643 | 0.4211 |
| 10 | 69.495 | 57.708 | 40.121 | 0.4609 |
| 15 | 67.76 | 58.012 | 35.895 | 0.3627 |

**Pros**:

1. Reduction in overfitting: by averaging several trees, there is a significantly lower risk of overfitting.
2. Less variance: By using multiple trees, you reduce the chance of stumbling across a classifier that doesn’t perform well because of the relationship between the train and test data.

**Cons**:

1. More complex compared to decision trees.
2. Hard to visualize the model.
3. More computationally expensive.

**Performance Measure:**

We used **10-fold cross validation technique** to evaluate the performance of the models. It is a technique used to evaluate predictive models by partitioning original sample into training set to train model and test set to evaluate the model. This technique also avoids overfitting.

In general, in K-Fold cross validation we split our data into k different subsets. we use k-1 subsets to train our data and leave last subset as test data. We then average the model against each of the folds and finalize our model.

The performance metrics used for the above models include: Accuracy, Precision, Recall and F1-score.

**Accuracy=(TP+TN)/(TP+FP+TN+FN)**

**Precision=TP/(TP+FP)**

**Recall=TP/(TP+FN)**

**F1 Score=2\*(Recall\*Precision)/(Recall+Precision)**

**KAGGLE COMPETITION**

We started off with different classification algorithms like Support Vector Machine, KNN and Logistic Regression which are evaluated to 0.84, 0.84, 0.96 respectively. But ended up with Random Forest for few reasons which are

* 1. They can’t be overfitted.
  2. It is a bagging algorithm which aims to reduce the complexity of models that overfit the training data.

Using Random Forest Algorithm. We managed to achieve scores which are consistent to 1.0

**Best Parameters Observed:**

Criterion: gini

n\_estimators: 50

n\_jobs: 10

**Further Improvements**

To further improve the model, we implemented multiple classifiers like SVM, Random Forest, Logistic Regression and AdaBoost and took Hard Majority Voting using Voting Classifier which consistently performed well with an accuracy score above 0.92. To calculate the accuracy, we did split the training dataset into training and testing parts where model is trained on training data and predict the features of test data and predicted class is then compared to the actual test class.

Highest Accuracy using Training Dataset = Fluctuating between 0.91 - 0.93

Best Parameters Observed:

**1. SVM**

C = 0.001

Gamma = 0.001

Kernel = rbf

**2. Random Forest**

n\_estimators = 50

n\_jobs = 10

**3. Log Regression**

Cross Validation = 5

penalty = l2

**4. AdaBoost**

n-estimators = 100

random\_state = 5