**[ Algorightms & Functions ]**

**1. Naive Bayes**

Simple preprocessing is performed before implementing the classifier. For categorical variable data, each category is converted into a number. A function is created for this, which converts categories into numbers for categorical columns and outputs them as a list according to the column size. Declares which columns are categorical in the data as a list and turns the categories into numbers.

**Pseudo Code**

def NaiveBayes(self, train\_df):

reference\_dict = dict()

for col in train\_df.columns[:-1]:

temp\_res = []

for label in self.\_\_unique\_label:

if col in self.\_\_categorical\_column:

unique\_value = train\_df[col].unique()

temp\_res1 = []

for u in unique\_value:

temp\_res1.append(category\_prob(train\_df, col, u, self.\_\_target, label, l=1))

temp\_res.append(temp\_res1)

else:

temp\_res.append(estimate\_mean\_std(train\_df, col, self.\_\_target, label))

reference\_dict[col] = temp\_res

self.\_\_reference\_dict = reference\_dict

prior\_probs = []

for label in self.\_\_unique\_label:

prior\_prob = estimate\_target\_prob(train\_df, self.\_\_target, label)

prior\_probs.append(prior\_prob)

self.\_\_prior\_probs = prior\_probs

**def estimate\_target\_prob(df, target, label):**

It is a function that obtains the estimated value of P. For each label of the output variable, the estimated value of P is obtained.

**def category\_prob(df, column, category, target, label, l=1):**

It is a function that estimates the probability value of Jinyoung, whose independent variable is categorical. Calculate the values for the categories that each independent variable may have and the labels of the output variables.

**class NaiveBaysClassifier:**

**def \_\_init\_\_(self,unique\_label,target,categorical\_column=[]):**

t is a class initialization function. When creating a class, the label of the output variable, the label of the output variable, and the categorical variable are told, and the label of the output variable can be automatically determined using the learning data, but the label of the output variable is included because it may not exist in the learning data.

**class NaiveBaysClassifier:**

**def train(self, train\_df):**

Functions responsible for training classifiers. The probability of the output label is estimated using the learning data and the conditional probability for each independent variable is estimated. Since these estimates do not change in the prediction phase, the estimates of the conditional probability distribution are placed in the \_\_reference\_dict field to shorten the test time so that they can be referenced in the prediction phase.

**class NaiveBaysClassifier:**

**def predict(self, new\_data):**

It is a function that receives a new independent variable and outputs a prediction label. First, the list with the number of labels of the output variable as the length is initialized to zero, and the conditional probability value is added by adding a log function. The probability of the output variable is also added by adding a log function. The index with the maximum value in the final list object\_value is returned to the prediction label.

**2. TF-IDF**

A method of weighting the importance of each word within DTM by using the frequency of words and the frequency of reverse documents (which take a specific expression for the frequency of documents). First, create a DTM, then assign a TF-IDF weight. TF-IDF can be used primarily to determine the similarity of documents, to determine the importance of search results in search systems, and to determine the importance of specific words within documents. TF-IDF determines that words that appear frequently in all documents are of low importance, and words that appear frequently in certain documents are of high importance. A low TF-IDF value means a low importance, and a high TF-IDF value means a high importance. In other words, in the case of terms such as the and a, the value of the TF-IDF of the term is naturally lower than that of other words because it is often found in all documents.

**Pseudo Code**

def TF-IDF(file):

tokens = tokenize(file)

for tk in tokens:

score = TF-IDF(tk)

wordVector.append((tk, score))

categoriesVectors = loadCategoryVactors()

for ca in categoriesVectors:

similarity = cosineSimilarity(ca, wordVector)

profile.append((ca.name, similarity))

return profile

**3. KNN**

In the classification phase, k is a user-defined constant, and an unlabeled vector is classified by assigning the label which is most frequent among the k training samples nearest to that query point. A commonly used distance metric for continuous variables is Euclidean distance. For discrete variables, such as for text classification, another metric can be used, such as the overlap metric. In the context of gene expression microarray data, for example, k-NN has been employed with correlation coefficients, such as Pearson and Spearman, as a metric.Often, the classification accuracy of k-NN can be improved significantly if the distance metric is learned with specialized algorithms such as Large Margin Nearest Neighbor or Neighbourhood components analysis.

**Pseudo Code**

def KNN(train, test, label):

for y in train:

calculateDistance(train[y], test)

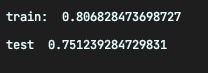
n = getSubset(train)

// n contains k training samples which are the k nearest neighbors of the test sample x

calculateCategory(label[y])

**[ Dataset Usage & Result ]**

1. http://archive.ics.uci.edu/ml/datasets/Adult



Screen Shot 2022-10-27 at 5.11.51.png

Screen Shot 2022-10-27 at 5.11.14.png

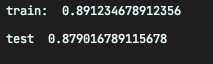
2. https://archive.ics.uci.edu/ml/datasets/glass+identification

Screen Shot 2022-10-26 at 20.41.45.png

Screen Shot 2022-10-27 at 3.25.38.png

Screen Shot 2022-10-27 at 3.25.06.png

3. News\_Category\_Dataset\_v3\_balanced.json



Screen Shot 2022-10-27 at 3.24.44.png

Screen Shot 2022-10-27 at 5.11.32.png

**[ Conclusion ]**

The three results according to the dataset are sequentially showing the results using Naive Bayes, TF-IDF, and K-NN algorithms, respectively. The first dataset, with independent variables categorical and two classes of output variables, showed accuracy in the 70-80% range for all algorithms. The second dataset, where the independent variable is continuous and the output variable is multiple classes, showed a significantly lower accuracy than 50% in the Naive Bayes algorithm, and nearly 80% in the other two algorithms. Finally, a dataset with fewer independent and output variables showed very low accuracy in the TF-IDF algorithm, but high accuracy figures above 90% in the other two algorithms.

Based on the above results, it seems necessary to select an appropriate algorithm according to the number of independent and output variables, and although the third dataset is significantly lower than the other two algorithms with more than 100,000 rows in about 20,000 rows, I think that the amount of accuracy is not the biggest factor. In the case of TF-IDF, there is a amount of data, but the accuracy is around 70% compared to other algorithms, so I think there was a lack of overall in some other areas. The best-performing algorithm is the K-NN algorithm, with all three data sets showing more than 85% results. The Naybe Bayes algorithm also showed good performance, but I suspect that the form of the independent variable may have been a problem, given that it showed close to 50% results on the second dataset.