First Assignment

Sergi Carol
Balbina Virgili

April 8, 2018

Introduction

The aim of this document is to explain our learning experience using a tool to analyze a data set. The chosen analytical tool is KNIME, which allow us to implement the behavior of some techniques and approaches previously studied. Our objective by the end of this project is to have successfully developed an analytical study of the chosen data, being able to solve all difficulties found during the process and, also, being able to objectively compare the different results obtained.

This document begins with an small explanation of our dataset, follows with an explanation of our workflow and, then, it continues with the different approaches that we have used to predict the data. Finally, this document ends with some conclusions of our experience.

1 Our Dataset

The dataset that we have used is called *Birth names in New York* [1]. This dataset contains information about the birth names of all the children in the city of New York over the years. The dataset has the following attributes:

- **Year of Birth**: The year in which the child was born.
- **Gender**: The gender of the child (MALE or FEMALE).
- **Ethnicity**: The ethnicity of the child (ASIAN AND PACIFIC ISLANDER, HISPANIC, WHITE NON HISPANIC or BLACK NON HISPANIC).
- **Child name**: The name of the child.
- **Count**: The number of children with the same name on the specified year.
- **Rank**: The position of the name on the specified year, ordered by the count.

2 KNIME

Our tool of use in order to work with our data set is *KNIME* [2]. *KNIME* allows to read, pre-process, process, and display data by the use of *nodes*. So, *KNIME* provides already implemented processes, known as *nodes*, which are ready to use by just configuring some parameters. A *workflow* consists of multiple nodes connected with each other in order to go from raw data to our objective. In Figure 1 we can see our solution *workflow*, which has been developed by connecting and setting parameters of specific nodes.

The main objective of this workflow using *KNIME* is to predict which *ethnicity* a person is from a given name and its gender.

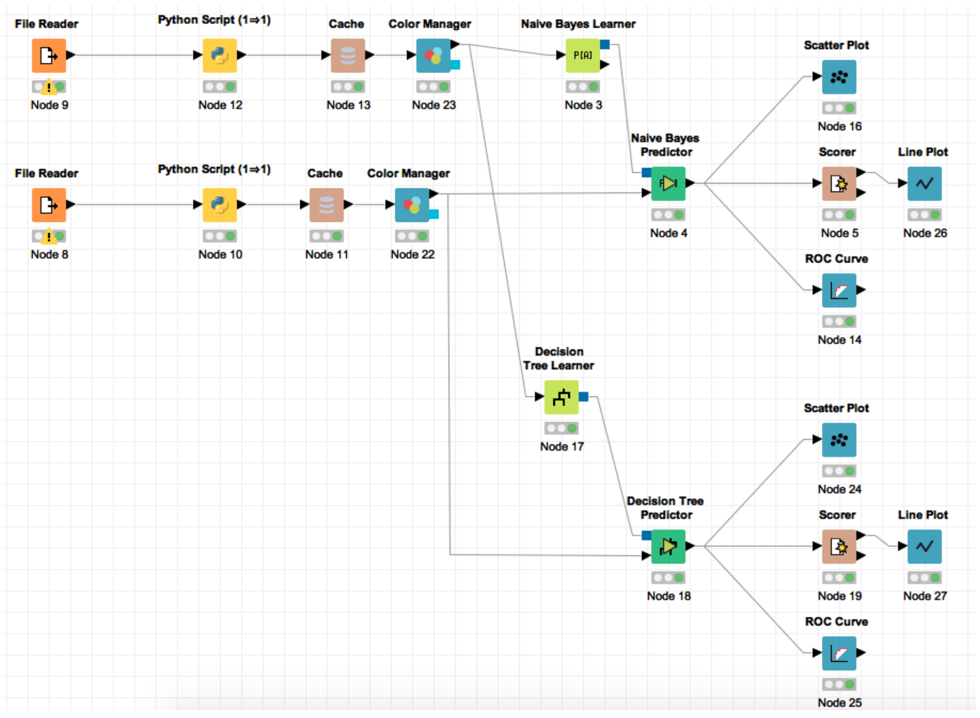


Figure 1: Workflow

Deeply information about most significant nodes used will be explained during this and the following sections of this document.

In our case, we begin with reading the data using the *File Reader* node, which reads the data set in *CSV* [3] format and outputs it into a *Python script* node, which is explained in the *Pre-processing* section. After the pre-processing, we cache the result of our pre-processing so we do not have to keep reading and pre-processing our data since this is a costly operation. We then input our data to a *Color Manager node* which colors each row by their ethnicity. We do this because it helps to identify each class later on.

An important thing to note is that we do not split our dataset into testing and training, but instead we use the data from the previous year to train out model, and the data from the current year to test it. It would also be possible to train the model with more years besides the last one, but we observed that names usually go by trends, meaning that using the data from 5 years ago would not produce a good model for our testing data.

After performing the initial data processing for both, the testing and training data, we create two different approaches in order to predict our model. These two approaches are explained in the following sections. Finally, we visualize the results of the data using *Scorers*, *ROC Curves*, *Scatter Plots* and *Line Plots*.

2.1 Pre-processing

In order to accomplish our task, we have needed to do some pre-processing with our input data. This is due to the names being grouped up and the only real attribute telling how many names where for one ethnicity was the *count* attribute. As such, our pre-processing takes the raw data from the file, parses it, and creates as many new rows as the value of count has for a given name.

Our first approach to create a pre-processing script was to implement a python program, outside *KNIM*, which would read the csv file containing the data, parse it, and replicate each column *count* times, and finally write the new columns on the file. The code looked like this.

```python
def parse_file(file):
    """
    Parses the file and duplicates the rows as many time as count
    """
    # Begin on the second line since the first is the header
    duplicated_rows = []
    for row in file[1:]:
        duplicated_rows += duplicate_row(row)
    write_rows(duplicated_rows)

def duplicate_row(row):
    """ Duplicates a row as many times as count value """
    count = int(row[4])
    rows = []
    for i in range(count):
        rows.append(row[:])
    return rows

def write_rows(rows_to_wirte):
    """ Writes the rows into the CSV file """
    writer = csv.writer(open('birth_names_2014.csv', 'a'))
    writer.writerows(rows_to_wirte)
```

The approach of pre-processing the data before hand had an important drawback, in case we wanted to switch the years that we wished to process, we would need to run the script beforehand, and as such, lose time. With this in mind we have decided to implement a new node in the *KNIME* workspace, by using the *Python* integration for KNIME [4]. This integration allows to write Python code as a node in KNIME in a way that take one (or more) inputs, and export one (or more) outputs.

In order to make use of the python integration, the extension must be installed, by going into *File -> Install Knime Extension*. Then, look for *Python* and install the extension. Once this is done, a few new nodes will appear in the nodes list.

When the node is opened, we have been able to see an small text editor with a console, a few variables ready to use, and the type of variables that are being used. The python variables themselves are stored using *Panda* data frames types, as such at least a bit of knowledge of Panda's data types is advised before using the Python integration.

```python
# Copy input to output
import pandas as pd
table = input_table.copy()

duplicate_rows = []

for index, row in table.iterrows():
        for i in xrange(row['Count']):
                table = pd.DataFrame(
                [row], columns= table.columns).append(table, ignore_index=True)

output_table = table
```

In this case the *KNIME* integration uses the *Panda* [5], which is a data analysis framework. The extension to work with Python gives the input data as a variable called *input_table*, and sets the output of the node to a variable called *output_table*. In our case we iterate through the rows of our input and append a new row with the same data as many times as the *count* variable.

Finally, we have also cached the result of the pre-processing in order to save time for the other iterations. We have also included a *Color Manager* in order to have a better visualization of the data.

2.2 Naive Bayes Predictor

Our first approach is to use a *Naive Bayes* predictor, which is a simple classification technique based on Bayes' Theorem. This method creates a probabilistic model that helps to make strong assumptions on how the data is generated, by assuming that all attributes of the examples are independent to each other given the context of the class. As a consequence of this assumption, the parameters of each attribute can be learned separately, what greatly simplifies learning, especially when the number of attributes is large. Although this 'naive bayes assumption' is clearly false in most real-world tasks, Naive Bayes often performs classification very well [6].

Naive Bayes needs a collection of labeled training examples to estimate the parameters of the generative model and the classification of new examples is performed with Bayes' rule by selecting the class that is most likely to have generated the example. That is why, as we had already explained on the previous section, we have defined the data from the year before (2013) as training examples to predict new values of the year 2014.

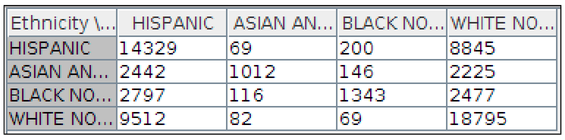
To implement this using KNIME, we have used two different nodes; Naive Bayes Learner and Naive Bayes Predictor. The options used for each of them are the following ones:

- Naive Bayes Learner
  - Classification Column: *Ethnicity*
  - Default probability: *0,001*
  - Maximum number of unique nominal values per attribute: *2030*
- Naive Bayes Predictor
  - Change prediction column name: *Prediction (etnicy)*
  - Append columns with normalized class distribution

The results obtained by the Naive Bayer Predictor implemented are showed below. To show the results obtained, a scorer node, a scatter plot node and a ROC curve node have been used.

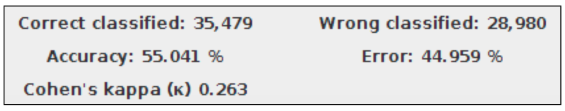
- **Scorer node**

  The scorer node provides the *confusion matrix*, as well as the percentage of accuracy and error of the performed prediction. The confusion matrix is a table that is used to describe the performance of a classification model on a set of test data for which the true values are known. And the accuracy can be defined as the number of hits divided by the total number of predictions.



Ethnicity \...	HISPANIC	ASIAN AN...	BLACK NO...	WHITE NO...
HISPANIC	14329	69	200	8845
ASIAN AN...	2442	1012	146	2225
BLACK NO...	2797	116	1343	2477
WHITE NO...	9512	82	69	18795

Figure 2: Confusion Matrix for Naive Bayes



Correct classified: 35,479	Wrong classified: 28,980
Accuracy: 55.041 %	Error: 44.959 %
Cohen's kappa (κ) 0.263	

Figure 3: Execution information

  The results retrieved show that the accuracy of the implemented predictor is 55.041%, which is not a very high percentage but is accepted for predicting data with a classifier. In general, the confusion matrix show that there are cases in which each ethnic is bad predicted for any of the other ethnics. Also, the confusion matrix reveals that the best predicted ethnics are *Hispanic* and *White Non Hispanic*, and, at the same time, it confuses each other much more than the other ethnics. Unfortunately, *Asian and Pacific Islander* and *Black Non Hispanic* have not been well predicted in most cases.

  Furthermore, the scatter node calculates the Cohen's kappa. The Cohen's kappa is an statistic coefficient which measures inter-rater agreement occurring by chance for items. Thus "perfect agreement" would be indicated by K = 1, and no agreement (other than that expected by chance) means that K= 0.[7]

  With the implemented scenario, the Cohen's kappa coefficient is 0,263, which is considered as poor or fair, depending on the references. In most cases, a balance between Cohen's Kappa coefficient and accuracy is needed. So, it is better to achieve a lower value of Cohen's Kappa coefficient but better accuracy, and not reversal.

- **Scatter plot node**

  The scatter plot node provides the graphical implementation of a scatter plot, which is a graph of two variables represented along two axes.

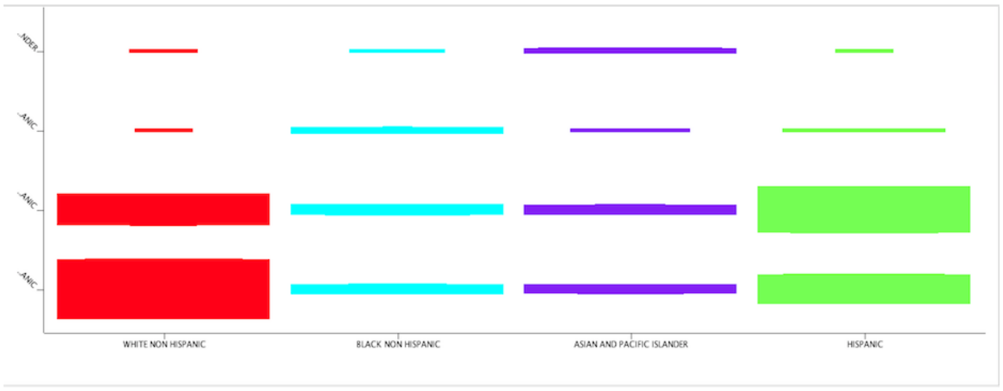


Figure 4: Scatter plot for Naive Bayes

  The scatter plot retrieved just print an approximation of the final result obtained. We just see a volume of the numeric results and the labels of the y-axis are not well shown.

- **Line Plot node**

  The line plot node provides the graphical implementation of a line plot, which is a graph that represents the frequency of data occurring a long a categorical line.

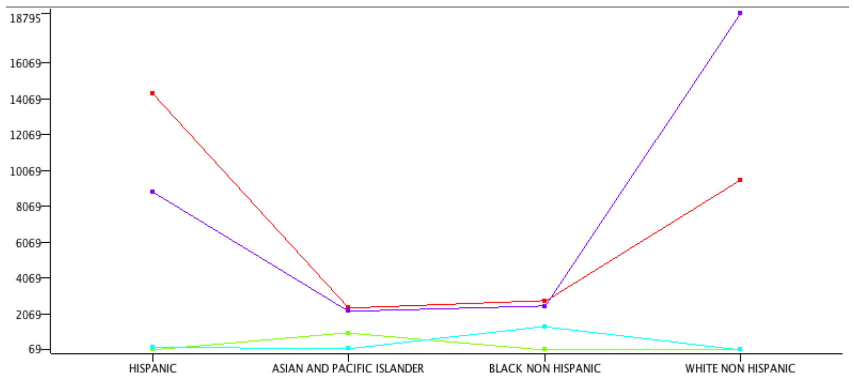


Figure 5: Line plot for Naive Bayes

  The line plot retrieved let us show the confusion matrix with a graph. This way, it is easier to understand the true predicted values against the bad predicted ones for each possible ethnics, and the proportions of quantity between each value are also easy to compare.

- **ROC curve node**

  The ROC curve node provides the graphical implementation of a receiver operating characteristic (ROC) curve [8], which is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. In other words, the ROC curve is created by plotting the true positive rate against the false positive rate at various threshold settings.

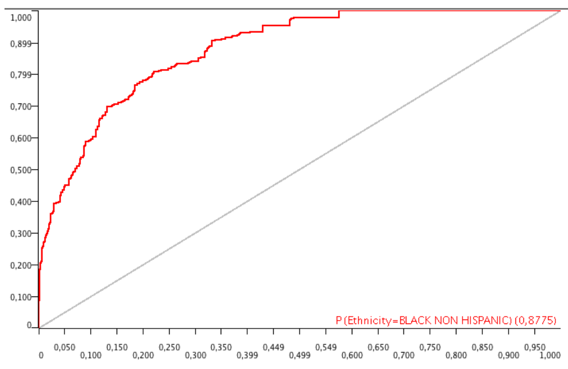


Figure 6: ROC curve of black non hispanic

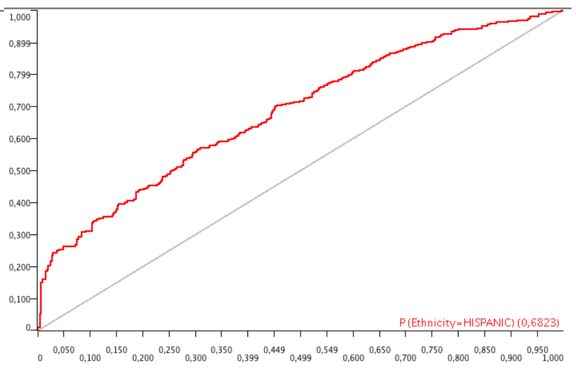


Figure 7: ROC curve of hispanic

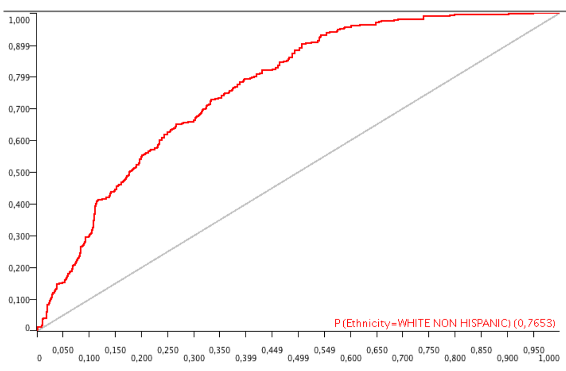


Figure 8: ROC curve of white non hispanic

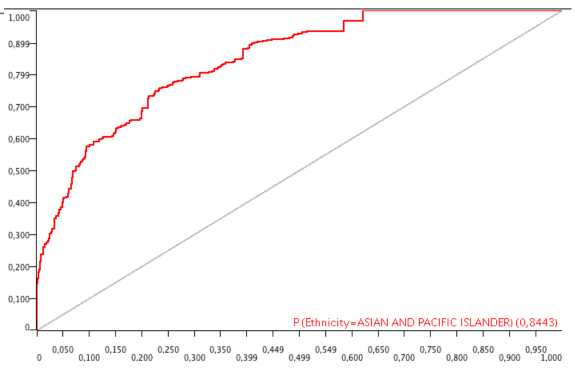


Figure 9: ROC curve of asian and pacific islander

The results retrieved show the ROC curve of each predicted ethnic. The important thing to take into account is the area under the curve (AUC). AUC represents the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative example. So, it measures the classifiers skill in ranking a set of patterns according to the degree to which they belong to the positive class, but without actually assigning patterns to classes.

We can see that the ethnic *Black Non Hispanic* has the biggest AUC, while *Hispanic* has the lowest one. Comparing it with the results obtained with the *confusion matrix*, we can confirm that the overall accuracy depends on the ability of the classifier to rank patterns, but also on its ability to select a threshold in the ranking used to assign patterns to the positive class if above the threshold and to the negative class if below. That is one of the reasons why, even *Hispanic* has been one of the best predicted ethnics of our implementation, it has the lowest AUC. And all the way around with *Asian and Pacific Islander*.

2.3 Decision Tree

In order to try to improve our error of the implemented *Naive Bayes Predictor*, our second approach has been using a *Decision Tree Predictor*, which uses Decision Tree algorithms. A decision tree algorithm is based on a recursive approach in which it begins by picking a test attribute so that the split reduces the *joint heterogeneity*. The predictor follows a decomposition of the input space, where all the splits are binary.

As a result, the decision tree algorithm generates a flow chart like a tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node consists in a class label.[9]

A predictor that uses a Decision Tree algorithm, also needs a collection of labeled training examples to estimate the parameters of the generative model. So we have also defined the data from the year before (2013) as training examples to predict new values of the year 2014.

To implement this using KNIME, we have used two different nodes; Decision Tree Learner and Decision Tree Predictor. The options used for each of them are the following ones:

- Decision Tree Learner
  - Classification column: *Ethnicity*
  - Quality measure: *Gini index*
  - Pruning method: *No pruning*
  - Reduced error pruning
  - Min. number of records per node: *2*

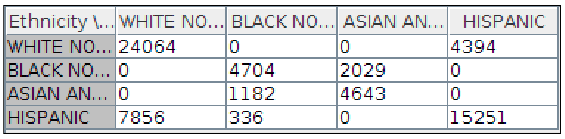
- Number of records to store for view: *10.000*
- Average split point
- Number of threads: *4*
- Skip nominal column without domain information

- Decision Tree Predictor
  - Maximum number of stored patterns for HiLite-ing: *65.000*
  - Change prediction column name: *Prediction (Ethnicity)*
  - Append columns with normalized class distribution

The results obtained by the Decision Tree Predictor implemented are showed below. We have also visualized it with a scorer node, a scatter plot node and a ROC curve node. As all this methods have already been briefly explained on section *Naive Bayes Predictor*, this time we are just exposing the results obtained for each one.

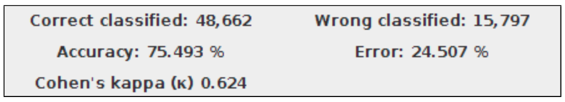
- **Scorer node**

  As we already know, the scorer node provides the *confusion matrix*, as well as the percentage of accuracy and error of the performed prediction and the Cohen's kappa.



Ethnicity \...	WHITE NO...	BLACK NO...	ASIAN AN...	HISPANIC
WHITE NO...	24064	0	0	4394
BLACK NO...	0	4704	2029	0
ASIAN AN...	0	1182	4643	0
HISPANIC	7856	336	0	15251

Figure 10: Confusion Matrix for Naive Bayes



Correct classified: 48,662	Wrong classified: 15,797
Accuracy: 75.493 %	Error: 24.507 %
Cohen's kappa (κ) 0.624	

Figure 11: Execution information

The results retrieved show that the accuracy of the implemented predictor is 75.493%, so it has increased almost a 20% in relation to the result obtained with Naive Bayes Predictor. The confusion matrix reveals that the best predicted ethnic is *White Non Hispanic*. We can see that the implemented predictor just confuses ethnics by pares. So, *White Non Hispanic* has sometimes been predicted as *Hispanic* and *Black Non Hispanic* as *Asian and Pacific Islander* and, reversal. We can also see that *Hispanic* has not been well predicted as *Black Non Hispanic* few times but other cases, the miss-predictions have been occurred just for one other ethnic. Moreover, the Cohen's kappa coefficient is 0,624, which is considered as a good rate.

- **Scatter plot node**

  The scatter plot implemented is showed below.

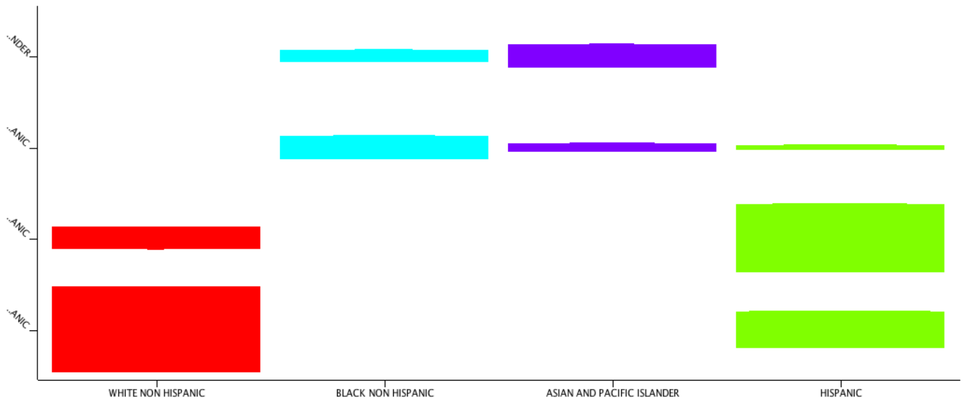


Figure 12: Scatter plot for Decision Tree

As it happened with the results obtained of the Naive Bayes predictor, the scatter plot retrieved just print an approximation of the final result obtained. So, we are able to just see a volume of the numeric results and the labels of the y-axis are not well shown.

- **Line Plot node**

  The line plot implemented is showed below.

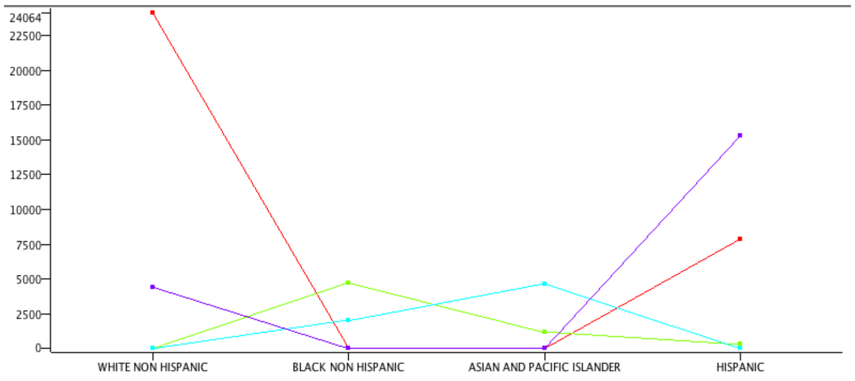


Figure 13: Scatter plot for Decision Tree

As it happened with the results obtained of the Naive Bayes predictor, the line plot retrieved let us show the confusion matrix with a graph. This way, it is easier to understand the true predicted values against the bad predicted ones for each possible ethnics, and the proportions of quantity between each value are also easy to compare.

- **ROC curve node**

  The results retrieved with the implemented ROC curve node are showed below.

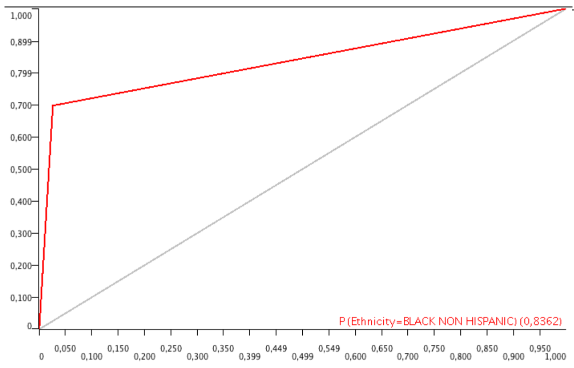


Figure 14: ROC curve of black non hispanic

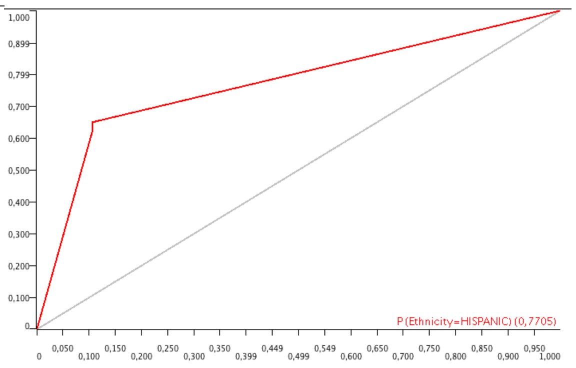


Figure 15: ROC curve of hispanic

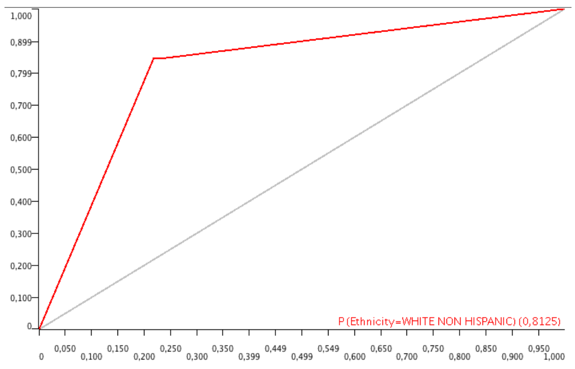


Figure 16: ROC curve of white non hispanic

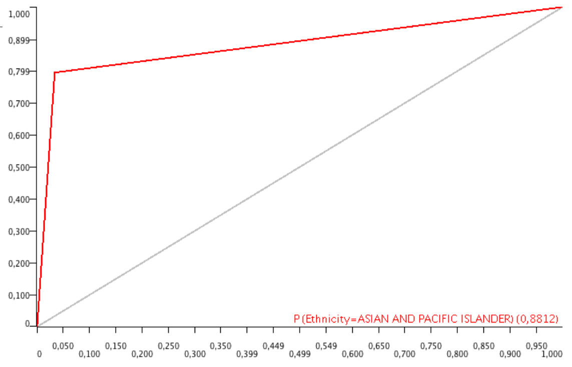


Figure 17: ROC curve of asian and pacific islander

We can see that the AUC for each ethnic has a completely different shape than the one obtained for the *Naive Bayes Predictor*. And, for each of them, there is a clear point where the bad predictions start to take place more often. In this case, the ethnic with lower AUC is still *Hispanic* but it is considered as a good range.

3 Conclusions

After finishing the implementation of our first scenario using *KNIME*, we can say that it is a powerful tool because it has allowed us to implement the different utilities that we needed to develop the analytical studio of our chosen data. *KNIME* provides a wide range of already developed nodes, which are ready to use after just adjusting some properties. And, in case that you don't find the node that you need, it gives the opportunity to implement your own one.

As we had already explained, we used *Python Script* node to develop the pre-processing needed for our dataset. Thanks to it, we have been able to have all process with the other analytical phases and do not have a pre-processing separately.

Thanks to our developed work, we have learned the following basic points for each predictor.

- Naive Bayes
  - Fast to train and classify
  - Not sensitive to irrelevant features
  - The model assumes independence of features
- Decision Trees
  - Very flexible and effective
  - Useful with classification problems and regression problems
  - Tendency towards over-fitting the training data

With the results obtained with the different experiments done, it has become quite clear that the use of a *Decision Tree Predictor* is a better approach than using the *Naive Bayes* one for our chosen data set. To affirm it, a comparison of the results obtained from all methods explained has been done.

To do it, first of all, we have rejected the *scatter plot* because it provided an ambiguous information about the results obtained. On the other hand, with the *accuracy* calculated of both predictors, we can confirm that *Decision Tree Predictor* got right results 20% more times than *Naive Bayes Predictor*. *Line plot* has helped us to show and compare the results retrieved by the *confusion matrix* in a simpler and even more understandable way. Furthermore, we can also see that the AUC retrieved from *ROC curve* for each ethnic is also higher with *Decision Tree Predictor* than *Naive Bayes Predictor*. So, it also confirms that *Decision Tree Predictor* has more possible thresholds that retrieve more well predicted values.

For all the facts and reasons exposed above, we can confirm that *Decision Tree Predictor* is the best approach to predict the ethnic of a child, when his name and gender is given.

To conclude, we can also see that for both predictors implemented, there are some names that are not well-predicted and are confused much for just another ethnic. This might be because some of the names from one ethnic are also used in the other.

References

[1] *Most Popular Baby Names by Sex and Mother's Ethnic Group, New York City.* 2018. URL: https://catalog.data.gov/dataset/most-popular-baby-names-by-sex-and-mothers-ethnic-group-new-york-city-8c742.

[2] *KNIME - Open for Innovation.* URL: https://www.knime.com/.

[3] Y. Shafranovich. *Common Format and MIME Type for Comma-Separated Values (CSV) Files.* RFC 4180. RFC Editor, 2005, pp. 1–6. URL: https://tools.ietf.org/html/rfc4180.

[4] *Python Extensions.* URL: https://www.knime.com/book/python-extensions.

[5] *Python Data Analysis Library¶.* URL: https://pandas.pydata.org/.

[6] Andrew McCallum, Kamal Nigam, et al. "A comparison of event models for naive bayes text classification". In: *AAAI-98 workshop on learning for text categorization.* Vol. 752. 1. Citeseer. 1998, pp. 41–48.

[7] Alan B Cantor. "Sample-size calculations for Cohen's kappa." In: *Psychological Methods* 1.2 (1996), p. 150.

[8] A Mohd Khuzi et al. "Identification of masses in digital mammogram using gray level co-occurrence matrices". In: *Biomedical imaging and intervention journal* 5.3 (2009).

[9] DL Gupta, AK Malviya, and Satyendra Singh. "Performance analysis of classification tree learning algorithms". In: *International Journal of computer applications* 55.6 (2012).