**Assignment 03 – Naive Bayes/Logistic Regression Classification**

***Chentai Yuan, Qianzhong Chen, Hao Ding***

***ECE448, Group 07***

***TA: Shuting Tao, Hanrong Zhang***

***April 24, 2023***

**Table of Contents**

**1. Description (Algorithm)2**

**2. Ultimate Tic-Tac-Toe 3**

**3. Offensive agent vs Your agent 4**

**4. Human vs. Your agent6**

**5. Extra Credit 7**

**6. Acknowledgments 9**

**Section Ⅰ: Description (Algorithm)**

For this exact cover problem, firstly, we find out all of the transformers of three types of blocks (domino, triomino, pentomino) by rotation and flipping. We keep a list to record all of the transformers for each shape. In total, there are 63 different pentominoes. Then, for each board, we only place single one transformer but try it for every place of the board. If no conflict occurs (no 0 is covered), we record the tilled board and the upper left coordination of the block transformer. So far, we have gained all of the elements we need to finish tilling task, if we overlap all of the boards we found before, we will find out that all of the 1s are covered. Now, we need to get rid of repetitions and pentominoes overlapping issues.

We took advantage of Algorithm X’s great performance in solving exact covering problem. For every board we found before, we changed its shape to a single row. For instance, the board was initially a (x, y) matrix, but now being transformed into (1, x\*y), every column of the row represents an entry of the board. By data processing, 1 means that this entry is tilled and 0 means no object take this position. Afterwards, we put all of the transformed rows together from top to below, forming a big matrix. Now, the exact covering problem has been simplified to selecting several rows, forming a new matrix to make sure every column has and only has one 1.

Algorithm X works in the following pattern: The algorithm selects a column in the matrix (referred to as the "pivot column") and tries to select one of the rows in that column to be included in the exact cover. If a row is selected, all other rows that contain elements in the same columns as the selected row are removed from consideration, and the process continues recursively with the reduced matrix until a valid exact cover is found. If at any point the algorithm reaches a dead end (i.e., it cannot find a valid row to select for the pivot column), it backtracks to the previous pivot column and tries a different row in that column. If there are no more rows to try in the previous pivot column, the algorithm backtracks further until it finds a pivot column where there are still rows available to try.

While implementing Algorithm X, some heuristics are used naturally.

Column Ordering: To improve performance, Algorithm X selects pivot columns in a specific order that the algorithm selects the pivot column with the fewest 1s first, and then recursively selects pivot columns in increasing order of the number of 1s in each column.

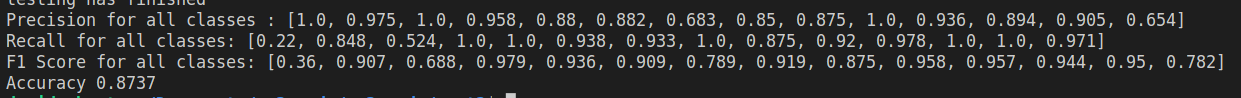
Branch and Bound: Branch and bound involves pruning parts of the search tree that are known to lead to invalid solutions, which can greatly reduce the amount of backtracking required.

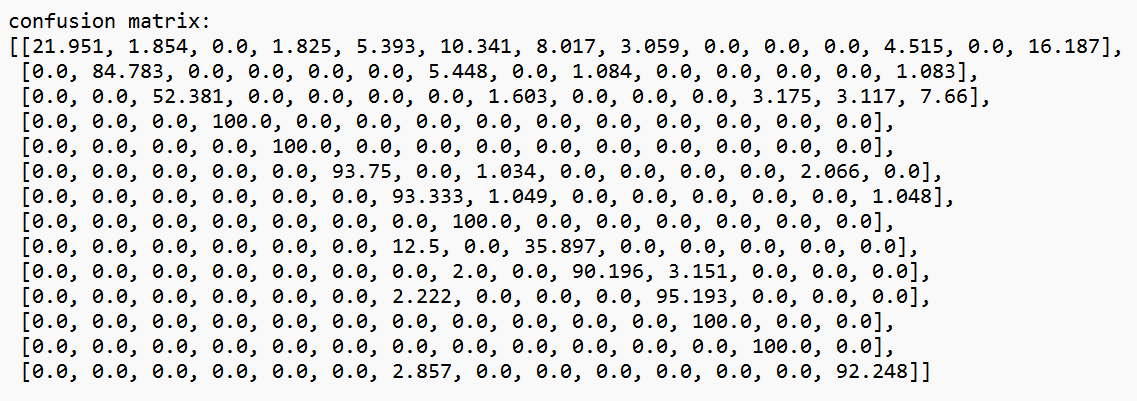
**Section Ⅱ: Text Classification**

1. Top 20 frequently occurred words of each class

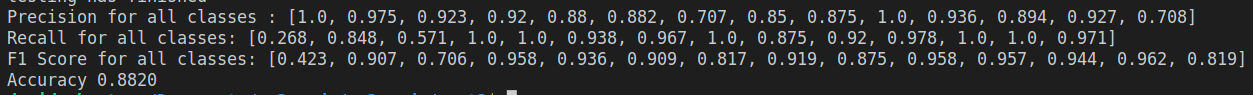


1. Prior Case

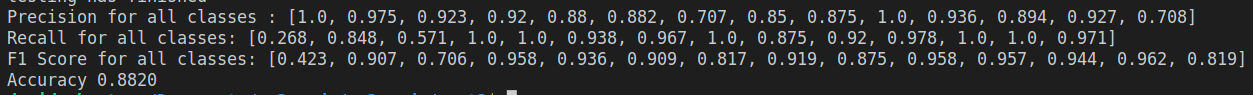




1. ML Case:

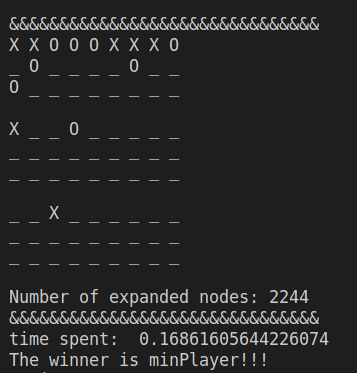


1. Uniform Distribution Case



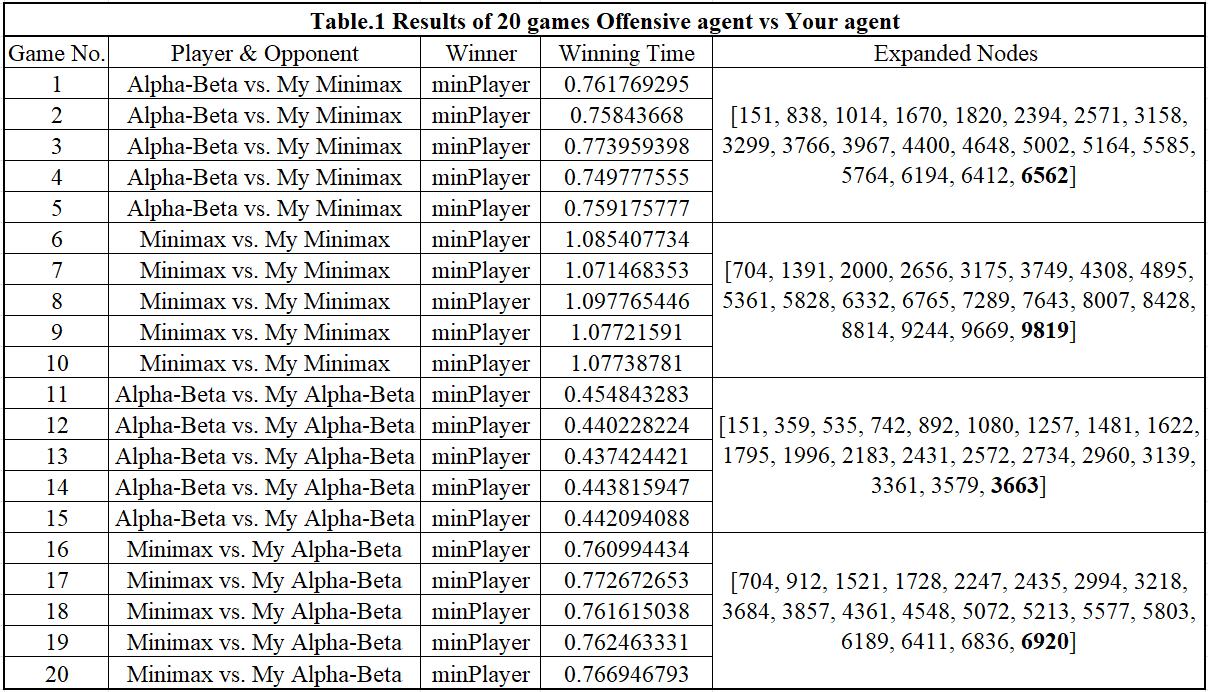
Discussion:

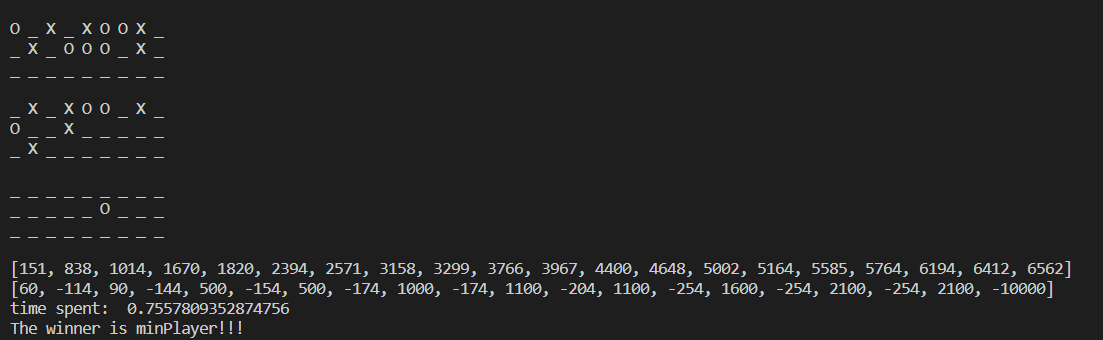
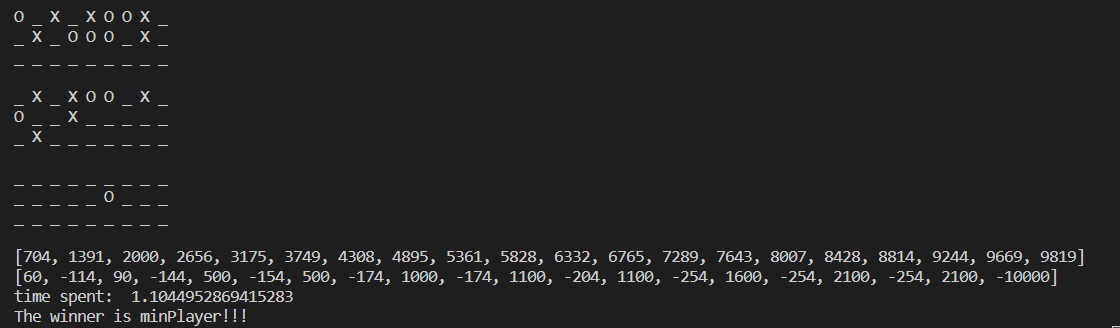
Not using class prior probability or Changing to uniform distribution increased the classification accuracy, meaning that including prior probability in naïve Bayes not always benefits the classification results. It may be attributed to the divergence in class distribution between training set and test set. Meanwhile, it makes great sense that ML case and uniform distribution case have exactly the same results, as posterior probability of ML case for each token should be 14 times of posterior probability of uniform distribution case, which dose not affect argmax function and final classification results.

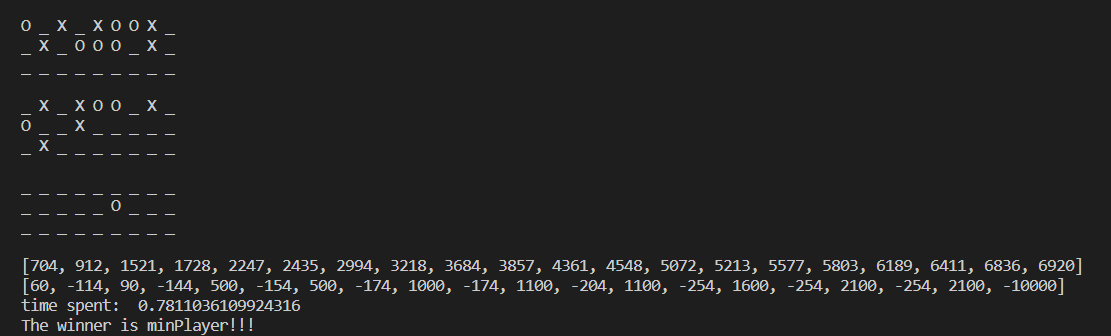
1. offensive(alpha-beta) vs defensive(alpha-beta), minPlayer go first:

**Section Ⅲ: Offensive agent vs Your agent**

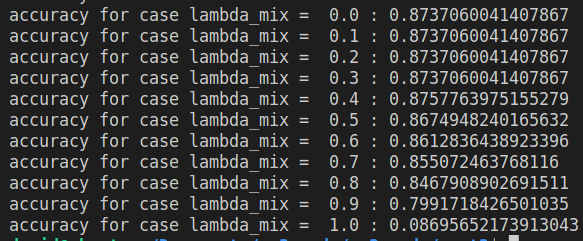
In my evaluation function, I added an evaluation rule 4 as calculating the distance bonus between the pieces on the board. The base points of distance will be 10, while the points will get smaller as the agent decides to put its piece (score += (10 - max(max\_distances))). Finally, a smaller distance will lead to larger bonus points. That evaluation rule will incline the agent to put their piece near to the exciting piece.

The final results for 20 games are shown in Table.1 below. The winner will always be my agent.

1. Offensive agent (Alpha-Beta) vs. Defensive agent (my Minimax):
2. Offensive agent (Minimax) vs. Defensive agent (my Minimax):
3. A picture containing graphical user interface

   Description automatically generatedOffensive agent (Alpha-Beta) vs. Defensive agent (my Alpha-Beta):
4. Offensive agent (Minimax) vs. Defensive agent (my Alpha-Beta):

**Section Ⅳ: Bigram Classification**

****

Maximum accuracy occurs at lambda\_mix = 0.4, with an accuracy of 0.8757, pure bigram classification accuracy is 0.0870.

*Q1. Does relaxation always help?*

Relaxation of naïve Bayes assumption does not always help. When we use a lambda\_mix of 1, which means that we only consider classifying using bigrams, the accuracy is very bad. It may happen because that bigram as a combination has way more degree of uniqueness, our training set cannot cover the breadth of bigrams and a lot of bigrams that our program has not learned occurs at test set, causing the trouble that it cannot classify the texts well.

*Q2. Is N-gram with a large N a good idea?*

No. As discussed before, with the increase of N, the complexity as well as number of linguistic data increase exponentially. As the lambda\_mix requires well-tuning, the much more time spent in training the model cannot trade-off the tiny improvement in performance.

**ACKNOWLEGEMENTS**

Statement of Contribution:

Qianzhong Chen developed all three parts and extra credit part individually and his code is submitted as .py. He is responsible for Section#2 and Section#4 of the report.

Chentai Yuan implemented the Assignment of uttt.py and wrote the report paper for section#3 and section#5. And he also helps to debug.

Hao Ding implemented the human agent and helped with debugging. He wrote the report paper for section#4.

**REFERENCE**

1. " Assignment 3: Planning, Games - ECE448 Spring 2023 Assignment#2 Manual."
2. Wikipedia on Logistic Regression: “https://en.wikipedia.org/wiki/ Logistic\_regression”