Identify the Gross Pay of an individual on Unigram Language Model by Corpus Cross Entropy: Project Report

**Team Members:**

Steven Valdez: Project Research and development

William Torres: Project Research and development, Project Report

Ben Koutsoumbaris: Project Research, Project Report, and Presentation

**Introduction**

This report’s purpose is to catalog and detail the project that solves the issues identified in the “Identify the Gross Pay of an individual on Unigram Language Model by Corpus Cross Entropy” assignment. This report will discuss the algorithms used. These descriptions include summaries of the algorithms used, training procedures, the trained parameters, the methods of dealing with blank data entries, and the testing metrics used, as well as a data instance representation in the form of a feature vector. After discussing the algorithms used, the testing experiment details, the results of the experiments, and the design pros and cons will be summarized and discussed. Lastly, the report will cover improvements that could be made to future builds of the algorithm.

The details of the project data specifications is as follows:

*“Given the data set* ***D = {< xi , yi >} N i=1*** *specified by the file* ***Human.csv****, where* ***N = 16281****, Each of the rows represent a data instance with its class label. That is, xi is a data instance,* ***x i = (x i 1 , x i 2 , ..., x i 14)*** *and* ***y******i*** *is a class label,* ***y i ∈ {> 50k, ≤ 50k}****. Moreover for each of* ***x i m ∈ x i*** *, there is a set of attributes, such that* ***x i m = {x i m,1 , ..., x i m,L },*** *where* ***L*** *is a natural number.”*

The group was tasked with creating a program project in a language of choice to complete the following:

* Get rid of rows which contain the ‘?’ character in the data set
* Assign arbitrary code to each of the attributes ***xm ∈ x***
* Split the data set into two sets at random, one being 70% of the data and the other 30%
* On the 70% set…
  + Determine the number of occurrences for each attribute
  + Using the MLE algorithm, determine the probability of an attribute for the class **‘> 50k’**
  + Using the MLE algorithm, determine the probability of an attribute for the class ‘<= **50k’**
* On the 30% set…
  + Pretend the class info for each of the data instances don’t exist
  + Compute the entropy of each instance using the Corpus Cross Entropy method
  + Assign an instance to a class label **∈ {> 50k, ≤ 50k}** of which has the lower entropy value.
  + Diagram

    Description automatically generated with low confidenceEvaluate the system using the following accuracy measurement:

Where ***ˆy i***is the assigned class label by the classifier and ***y i***is the true class label of a data instance ***x[i]*** in ***Dtest*** and ***T*** is the number of data instances in ***Dtest***.

Icon

Description automatically generated with medium confidence

For this project, C++ was the programming language of choice. This was primarily due to the group’s history with the language; every member had worked with C++ in the past and was relatively familiar with most of the common libraries, conventions, syntax, etc. An early consideration was to use the Python language instead, but the group found that the additional help they’d gain from using the Python libraries did not outweigh the experience they had with C++.

The group used a variety of libraries to perform different functions. Below are the C++ libraries that were used:

Text

Description automatically generated

*Fig. 1 The C++ libraries used*

**Description of Algorithms**

*Feature Vector*

The data instances/rows in our data are represented by a class called dataInstance. Our class has integers to represent each column since we are using numbers to represent all the data. We also have another class called unique. This class has 3 integers and 2 floats to represent the unique value, the number of times it appears for each class and its probability for each.

*Algorithm Descriptions*

This project used two distinctly different algorithms to perform the tasks introduced in the *Introduction* section of this document. For the 70% data set, the program uses the Max Likelihood Estimation, or MLE, algorithm to find the probabilities for each of the attributes in each of the classes described.

For the 30% data set, the program uses the Corpus Cross Entropy method to determine the entropy of each instance, using the following formula:

When given a corpus *C* of size *N* consisting of tokens *c1, …, cN*.

*Training Procedures*

First, we determine the unique values and the number of times it appears in our data. We do this by looping through each column and storing the unique values in our unique vectors for every column. If a value repeats, we increment our counter for that value depending on what 50k class that row has. So, for every unique value, we have 2 different counters.

*Trained Parameters*

Now that we have every unique value and the number of times it appears for both 50k classes, we can do our probability calculations. We do this by using the MLE algorithm. In other words, we take the number of times it appeared with a certain 50k class and divide it by the total number of times that certain 50k class appeared. We use this formula for both 50k classes so we end up with 2 probabilities for every unique value that is stored in the unique class. This will be used to calculate entropy which we use for predictions.

*Methods of Dealing with Blank Entries*

We skip data instances with ‘?’ in them. We do this by not storing them when we read the human.csv file into our program. For entries or values that appear in the 30% testing data but not in the 70% training data, we skip them in entropy calculation. In other words, we treat them as if they were 0. We are mainly adding in our entropy formula so this approach works.

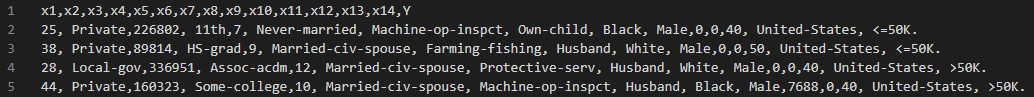
*Testing Metrics*

For our testing set. We calculate entropy for every row using the probabilities that we obtained from the 70% training data. This is done using the corpus cross entropy formula given above. We do this for both 50k classes. We then compare the entropy and choose the smallest out of the two by assigning it its corresponding class label. For example, if the entropy calculated using the >50k probabilities was smaller than the entropy for the <=50k probabilities, we then assign it the >50k class label using whatever arbitrary value was given to it (in this case, 2). In our program, we store this label in a separate array called entropyResult so it can be used to calculate accuracy.

**Experiments**

*Settings*

To test the program, the group was provided with a data set contained in the comma-separated values file Human.csv. This file contained data on various statistics concerning individuals.



*Fig. 2 An example of the data set rows. Note the topmost row was ignored in calculations.*

In our experiment, we calculate accuracy by going through our predictions and counting how many times it matches its actual label. We take this and divide it by the size of the 30% testing data. This is how we determine the efficiency of our system. We calculate it to around 6 decimal places.

*Results*

In our system, we get around 75% - 80% accuracy regularly. This is relatively high and gives us good evidence that our system is valid and effective. The below figures are examples of the results generated by the program:

*Text

Description automatically generated*

*Fig. 3 The 70%/30% data split*

*A screenshot of a computer

Description automatically generated with medium confidence*

*Fig. 4 Examples of generated probabilities for each attribute*

*Text

Description automatically generated with low confidence  
Fig. 5 The results of 70% and 30% split testing*

*Design Pros/Cons*

There were several pros and cons that came with the design choices that our group made. We were very proud that the program accomplished the core objective, so we’d list this as the first and most important pro. In other words, the program accomplished all that was being asked of it in the steps listed in the *Introduction* portion of this document, using the algorithms described in the *Algorithm Descriptions* section. We were thankful that the libraries we used were simple and familiar to us; their ease of use allowed us to quickly determine which functions from each we needed to use. One additional pro was that throughout our testing, we were able to compile and execute the code very speedily. The program never took over a few seconds to build.

On the other hand, some cons that limited the performance and versatility of the code were present. Most notably, there were portions of the project that were “hard-coded”. That is, they were implemented in such a way that they would only work with the provided data set. An example of this was the function that assigned arbitrary numbers to each value in the array, which had a case for each possible string that could be found in the data set. The code would likely not work if another data set besides the one provided was used. Another possible con has to do with the types of computers being used to develop the program. Since we were using computers with higher-end specs, a user on a lower-end computer may find that the time the program takes to run is longer than expected.

**Improvements**

There were some potential improvements discussed that could be made to the program. The biggest improvement idea was to allow for the program to work on any data set contained in the .csv filetype, by changing the code. This would allow for the program to perform the calculations with the algorithms on any data set, not just the specific one that it was hard coded to work with. We also had the thought of making slight improvements to the user interface (UI). We could investigate creating a custom UI to display and categorize the different calculations that the program performs. This would allow for a more pleasant viewing experience as compared to a terminal-based UI. Besides these two improvements, in the future, we could also add more probability calculation options, besides the ones that use MLE and Cross Entropy methods.