Foundations of AI - Lab 2 - WriteUp

Wikipedia Language Classification

Gautam Gadipudi - gg7148@rit.edu

Code Description:

Open the README.md in a markdown viewer.

Feature Description:

- At least 2 words contain substring “ij”
    - True if the example contains at least 2 words with “ij” substring
    - False otherwise
- At least one word is “de” or “het”
    - True if the example has at least word as “de” or “het
    - False otherwise
- At least one word is “en”
    - True if the example has at least one word as “en”
    - False otherwise
- At least one word is “and”
    - True if the example has at least one word as “and”
    - False otherwise
- Is one of Dutch common words
    - True if the example has one word from common Dutch words
    - False otherwise
- Is one of English common words
    - True if the example has one word from common English words
    - False otherwise
- Average word length
    - True if average word length is greater than equal to 4.25
    - False otherwise
- Two letter(char) word freq.
    - True if percentage of two character words is greater than or equal to 0.2
    - False otherwise
- Words have common bigrams eg. “th”, “er”, “on” etc.
    - True if any word has a common bigram
    - False otherwise
- Words have common trigrams eg. “the”, “and”, “tha”, “ent” etc.
    - True if any words has a common trigram
    - False otherwise

Decision tree learning:

1. Steps in training:

Steps to build the decision tree:

- First extract features from the input and lets call them rows.
- Get the best possible split for the input rows
    - This is done by calculating the information gained by choosing each of the available columns as the root node.
    - We choose the node with the most information gain.
- Then we split the data into two (i.e true_rows and false_rows) w.r.t. the node.
    - This is done using a partition function that returns true_rows and false_rows tuple.
- Then we recursively build the tree on the true_rows and false_rows until there is no information gain, at which point we return the leaf node which predicts from the existing rows whether to return ‘nl’ or ‘en’ depending on their number of occurrences.

2. Calculations in training:

- *Gini impurity*: It is a measure of how well a node segregates the given input(rows). Lesser the gini value, better the node is in segregating rows.

$$Gini\ impurity = 1 - \sum_{x \in [en,\ nl]} P(x)^2$$

$$where,$$

$$P(x)\ is\ the\ probability\ of\ \ occurence\ of\ \ x\ in\ all\ rows.$$

- *Information gain*: It is a measure of how much information is gained if we split using a particular feature.

$$Info.\ gained = gini(current\_node) - (P(False) * gini(false\_rows) + P(True) * gini(true\_rows))$$

$$where,$$

$$current\_node\ is\ the\ node\ for\ which\ we\ are\ calculating\ the\ info.\ gain$$

$$true\_rows\ and\ false\_rows\ are\ a\ result\ of\ \ splitting\ using\ \ the\ node.$$

Decision tree visualization:

col4 >= True? {'nl': 173, 'en': 119}
    True
    col3 >= True? {'nl': 173, 'en': 3}
        True
        {'en': 1}
        False

```
col8 >= True? {'nl': 173, 'en': 2}
        True
        col5 >= True? {'nl': 158, 'en': 1}
                True
                col1 >= True? {'nl': 62, 'en': 1}
                        True
                         {'nl': 38}
                        False
                        col6 >= True? {'nl': 24, 'en': 1}
                                True
                                 {'nl': 16}
                                False
                                col2 >= True? {'nl': 8, 'en': 1}
                                        True
                                         {'nl': 4}
                                        False
                                        col9 >= True? {'nl': 4, 'en': 1}
                                                True
                                                 {'nl': 3, 'en': 1}
                                                False
                                                 {'nl': 1}
                False
                 {'nl': 96}
        False
        col5 >= True? {'nl': 15, 'en': 1}
                True
                col1 >= True? {'nl': 6, 'en': 1}
                        True
                         {'nl': 4}
                        False
                                True
                                False
                                col6 >= True? {'en': 1, 'nl': 1}
                                        True
                                         {'nl': 1}
                                        False
                                         {'en': 1}
                False
                 {'nl': 9}
```

False
{'en': 116}

Adaboost ensemble learning:

In contrast to a single tree in decision tree learning, we use a forest of trees of unit height to make a prediction. These trees are called stumps.

1. Steps in training:

- Extract the features and let's call them rows.
- At first, all the rows are given a weight of 1/n, where n is the number of examples in the training set.
- Get the best possible split for the input rows.
    - This is done by calculating the information gained by choosing each of the available columns as the root node.
    - We choose the node with the most information gain.
- Choose this as the stump and calculate error and update the weights using the calculations provided in the next section.
- Do this for k iterations, here k is equal to the number of features.
- For each iteration, also calculate z value, also called “amount of say” when classifying. (Calculation shown in the next section.)
- After all k iterations, return k stumps and their z values as hypotheses.

2. Calculations in training:

Use the same exact algorithm below:

function ADABOOST($examples, L, K$) **returns** a weighted-majority hypothesis
  inputs: $examples$, set of N labeled examples $(x_1, y_1), \ldots, (x_N, y_N)$
    L, a learning algorithm
    K, the number of hypotheses in the ensemble
  local variables: **w**, a vector of N example weights, initially $1/N$
    h, a vector of K hypotheses
    z, a vector of K hypothesis weights

  for $k = 1$ **to** K **do**
    $\mathbf{h}[k] \leftarrow L(examples, \mathbf{w})$
    $error \leftarrow 0$
    for $j = 1$ **to** N **do**
      if $\mathbf{h}[k](x_j) \neq y_j$ **then** $error \leftarrow error + \mathbf{w}[j]$
    for $j = 1$ **to** N **do**
      if $\mathbf{h}[k](x_j) = y_j$ **then** $\mathbf{w}[j] \leftarrow \mathbf{w}[j] \cdot error/(1 - error)$
    $\mathbf{w} \leftarrow$ NORMALIZE(**w**)
    $\mathbf{z}[k] \leftarrow \log(1 - error)/error$
  return WEIGHTED-MAJORITY(**h**, **z**)

Adaboost hypotheses:

Stump: col4 >= True?, Z: -0.30368241379822924
Stump: col4 >= True?, Z: -0.22055175720677825

Stump: col4 >= True?, Z: 0.0683700496701496
Stump: col4 >= True?, Z: -0.1491187548190892
Stump: col4 >= True?, Z: 0.007778785055869354
Stump: col4 >= True?, Z: -0.05576816193284429
Stump: col4 >= True?, Z: 0.08597314195688026
Stump: col4 >= True?, Z: 0.14170493885359886
Stump: col4 >= True?, Z: 0.0722627979072632
Stump: col4 >= True?, Z: 0.03292709807453965

Decision tree prediction:

For each test example/to be predicted example,

- Extract features
- Traverse these along the decision tree
- Print the value of the leaf node

Adaboost prediction:

For each test example/to be predicted example,

- Extract features
- Get the prediction from each stump.
- Print the weighted majority of the prediction w.r.t the z values for stumps.