### README

Heng zhe Duan(hd79), Yu Cheng(yc489)

Usage:

make bayes

./bayes <Training\_File> <Validation\_File>

The program follows the Naive Bayes algorithms given. It stores all the probability in natural log base to avoid underflow. It uses a struct entry to extract the information from the given data, and stores frequencies of words in a map, and probability values for each word in a vector.

### Bayes.cpp

#include <iostream>

#include <vector>

#include <cmath>

#include <sstream>

#include <fstream>

#include <map>

#include <string>

using namespace std;

// A sample

typedef struct entry {

int classification;

vector<pair<int, int> > frequency;

entry(string s) {

stringstream ss(s);

ss >> classification;

int a;

int b;

char c;

while (ss >> a >> c >> b) {

frequency.push\_back(make\_pair(a, b));

}

}

entry() { }

void print() {

cout << "classification = " << classification << endl;

for(int i = 0; i < frequency.size(); ++i) {

cout << "Feature: " << frequency[i].first << " has value: " << frequency[i].second << endl;

}

}

} entry;

// Penalties for false pos and false neg

const double c10 = 10;

const double c00 = 0;

const double c01 = 1;

const double c11 = 0;

map<int, int> pos\_word\_map;

map<int, int> neg\_word\_map;

int pos\_word\_count = 0;

int neg\_word\_count = 0;

int max\_feature = -1;

int pos\_ins\_count = 0;

int neg\_ins\_count = 0;

vector<double> pos\_prob;

vector<double> neg\_prob;

double py, pny;

int predictc(entry e) {

double pos = log(py);

double neg = log(pny);

for (int i = 0; i <e.frequency.size(); ++i) {

pos += pos\_prob[e.frequency[i].first]\*e.frequency[i].second;

neg += neg\_prob[e.frequency[i].first]\*e.frequency[i].second;

}

if (log(c10-c11) + pos >= log(c01-c00) + neg) return 1;

else return -1;

}

int predictb(entry e) {

double pos = log(py);

double neg = log(pny);

for (int i = 0; i <e.frequency.size(); ++i) {

pos += pos\_prob[e.frequency[i].first]\*e.frequency[i].second;

neg += neg\_prob[e.frequency[i].first]\*e.frequency[i].second;

}

if (pos > neg) return 1;

else if (pos < neg) return -1;

else return pos\_ins\_count >= neg\_ins\_count;

}

int main(int argc, char\* argv[]) {

if (argc != 3) {

cout << "Usage: " << argv[0] << " <TRAINING\_FILE> <VALIDATION\_FILE>" << endl;

}

ifstream train(argv[1]);

ifstream valid(argv[2]);

while (!train.eof() && train.good()) {

string s;

getline(train, s);

if (s.length() > 0) {

entry e(s);

if (e.classification == 1) {

++pos\_ins\_count;

for (int i = 0; i < e.frequency.size(); ++i) {

pos\_word\_map[e.frequency[i].first] += e.frequency[i].second;

pos\_word\_count += e.frequency[i].second;

if (e.frequency[i].first > max\_feature) max\_feature = e.frequency[i].first;

}

}

else {

++neg\_ins\_count;

for (int i = 0; i < e.frequency.size(); ++i) {

neg\_word\_map[e.frequency[i].first] += e.frequency[i].second;

neg\_word\_count += e.frequency[i].second;

if (e.frequency[i].first > max\_feature) max\_feature = e.frequency[i].first;

}

}

}

}

cout << "Max feature = " << max\_feature << endl;

// Calculate probability

for (int i = 0; i <= max\_feature; ++i) {

pos\_prob.push\_back(log((double)(pos\_word\_map[i]+1)/(double)(pos\_word\_count + max\_feature)));

neg\_prob.push\_back(log((double)(neg\_word\_map[i]+1)/(double)(neg\_word\_count + max\_feature)));

}

cout << "total pos words = " << pos\_word\_count << endl;

cout << "total neg words = " << neg\_word\_count << endl;

cout << "total pos count = " << pos\_ins\_count << endl;

cout << "total neg count = " << neg\_ins\_count << endl;

py = (double)(pos\_ins\_count) / ((double)(pos\_ins\_count+neg\_ins\_count));

pny = (double)(neg\_ins\_count) / ((double)(pos\_ins\_count+neg\_ins\_count));

cout << "pos y prob = " << py << endl;

cout << "neg y prob = " << pny << endl;

// Do prediction

int correct = 0;

int falsepos = 0;

int falseneg = 0;

int correctc = 0;

int falseposc = 0;

int falsenegc = 0;

while (!valid.eof() && valid.good()) {

string s;

getline(valid, s);

if (s.length() > 0) {

entry e(s);

if (predictb(e) == e.classification) {

++correct;

}

else if (e.classification == 1) {

++falseneg;

}

else {

++falsepos;

}

if (predictc(e) == e.classification) {

++correctc;

}

else if (e.classification == 1) {

++falsenegc;

}

else {

++falseposc;

}

}

}

cout << "part b:" << endl;

cout << "correct = " << correct << endl;

cout << "falsepos = " << falsepos << endl;

cout << "falseneg = " << falseneg << endl;

cout << "accuracy = " << (double)correct/((double)correct+falsepos+falseneg) << endl;

cout << "part c:" << endl;

cout << "correct = " << correctc << endl;

cout << "falsepos = " << falseposc << endl;

cout << "falseneg = " << falsenegc << endl;

cout << "accuracy = " << (double)correctc/((double)correctc+falseposc+falsenegc) << endl;

return 0;

}