```
import pprint, sys, math

class PPMC:

    ESCAPE_SEQUENCE = "$"
    TOTAL_CHARACTERS = 256

    context_length = 2
    seen_characters = set([])
    counts = {}
    context_dictionaries = []
    training_message = ""
    probabilities = []

    def __init__(self, training_message, context_length=2):
        self.context_length = context_length
        self.training_message = training_message
        self.context_dictionaries = [{} for i in range(self.context_length+1)]

    def get_relevant_values(self, dictionary, impossible_char_candidates):
        values = []
        for (k,v) in dictionary.items():
            if k not in impossible_char_candidates:
                values.append(v)
        return values

    def update_impossible_set(self, dictionary, impossible_char_candidates):
        for k in dictionary.keys():
            impossible_char_candidates.add(k)

    def output_log(self, relevant_message):
        char = relevant_message[-1]
        context = relevant_message[:-1]
        length = len(context)
        impossible_char_candidates = set([])
        while context:
            if context in self.context_dictionaries[length]:
                values = self.get_relevant_values(self.context_dictionaries[length][context],
                                                  impossible_char_candidates)
                if not values:
                    context = context[1:]
                    length = len(context)
                    continue
                if char in self.context_dictionaries[length][context]:
                    self.probabilities.append(self.context_dictionaries[length][context][char]*1.0/
                                                                        (len(values)+sum(values)))
                    print char+","+str(self.probabilities[-1])
                    return
                else:
                    self.probabilities.append(len(values)*1.0/(len(values)+sum(values)))
                    print self.ESCAPE_SEQUENCE+","+str(self.probabilities[-1])
                    self.update_impossible_set(self.context_dictionaries[length][context],
                                               impossible_char_candidates)
            context = context[1:]
            length = len(context)
        assert length==0
        values = self.get_relevant_values(self.counts, impossible_char_candidates)
        if not values:
            self.probabilities.append(1.0/(self.TOTAL_CHARACTERS-len(self.seen_characters)))
            print char+","+str(self.probabilities[-1])
            return
```

```
        if char in self.counts:
            self.probabilities.append(self.counts[char]*1.0/(len(values)+sum(values)))
            print char+","+str(self.probabilities[-1])
        else:
            self.probabilities.append(len(values)*1.0/(len(values)+sum(values)))
            print self.ESCAPE_SEQUENCE+","+str(self.probabilities[-1])
            self.probabilities.append(1.0/(self.TOTAL_CHARACTERS-len(self.seen_characters)))
            print char+","+str(self.probabilities[-1])

    def update_dictionaries(self, relevant_message):
        char = relevant_message[-1]
        context = relevant_message[:-1]
        length = len(context)
        while context:
            if context in self.context_dictionaries[length]:
                if char in self.context_dictionaries[length][context]:
                    self.context_dictionaries[length][context][char]+=1
                else:
                    self.context_dictionaries[length][context][char]=1
            else:
                self.context_dictionaries[length][context]={char:1}
            context = context[1:]
            length = len(context)
        assert length==0
        if char in self.counts:
           self.counts[char]+=1
        else:
            self.counts[char]=1

    def print_dictionaries(self):
        pp = pprint.PrettyPrinter(indent=2)
        pp.pprint(self.counts)
        pp.pprint(self.context_dictionaries)

    def calc_self_information(self):
        total_bits = 0.0
        for p in self.probabilities:
            total_bits += math.log(1.0/p, 2)
        return total_bits

    def build_model(self):
        assert len(self.training_message)>0
        self.probabilities.append(1.0/self.TOTAL_CHARACTERS)
        print self.training_message[0]+","+str(self.probabilities[-1])
        self.counts = {self.training_message[0]:1}
        self.seen_characters.add(self.training_message[0])
        # self.print_dictionaries()

        for i in range(1, len(self.training_message)):
            cur_char = self.training_message[i]
            cur_context_length = i if i < self.context_length else self.context_length
            relevant_message =  self.training_message[i-cur_context_length:i+1]
            self.output_log(relevant_message)
            self.update_dictionaries(relevant_message)
            self.seen_characters.add(self.training_message[i])
        print "Total bits "+str(self.calc_self_information())

training_msg = "accbaccacba"
inst = PPMC(training_msg, int(sys.argv[1]))
inst.build_model()
```