

# Solving Coin toss probability problems with NLP

## Abstract:

Humans have always tried to create machines that can think and behave like them. Processing the natural language is one of the most important thing such a machine would be able to do. Solving basic math word problems using NLP is one of the most interesting areas in the NLP research. Many people have previously attempted to solve the math word problems with NLP. But all of the algorithms are built for solving the arithmetic math word problems. This project demonstrates an algorithm to solve a class of probability problems, the coin toss problems.

## Introduction:

Solving a probability problem requires the machine to know all the possible outcomes of the problem and the outcomes that satisfy the problem statement. In the coin toss problems, all the possible states are given by the number of coins tossed in the problem. The question can be about the count of heads or tails or both. The problem can be solved either supervised and unsupervised method. Since there are no labelled data for coin toss problems the algorithm is designed using unsupervised method.

Problem
If two coins are tossed, what is the probability of seeing one head and one tail?
Solution: Totals possible outcomes=4 Outcomes which contains one head and tail=2 Probability =.5

Consider the problem in table 1. To be able to solve the problem we need the following parameters.

### Parameters:

- Number of coins tossed
- Number of heads in the outcome
- Number of tails in the outcome

From the above problem we know that No of coins tossed is 2. No of tails is 1 and No of heads in 1. We will learn this parameters from the question by knowing the

parts of speech tags of each word in the question and assign each number to its relevant parameter.

## **Previous work:**

Such a problem was first attempted by Bobrow as part of his PhD dissertation in 1964. The system described in (Bobrow, 1964) could solve an algebra problem given in a subset of natural language. A common trait of most systems that use this approach is to work with a subset of the natural language, most commonly via a Controlled Natural Language. This work attempts to extract information from the ambiguous English sentences themselves. The latest in this line of work for mathematical algebraic word problems, without using empirical methods, is in (Bakman, 2007) which proposed a method that improved upon (Dellarosa, 1986). Dellarosa used “schemas” to solve addition/subtraction problems in 1986 with some improvements over (Fletcher, 1985) for classifying entities like “dolls” are “toys”. Schemas are templates for problem solving. (Bakman, 2007) extends schemas to handle extraneous information and can solve multi-step problems unlike its predecessors. In a review of such knowledge-based systems by (Mukherjee and Garain, 2009), it was remarked that there are no datasets to compare performances. The recent empirical methods that are cited below provided datasets that can now be used for evaluation. For example, (Kushman et al., 2014) proposed a word problem solver which uses statistical analysis to solve word problems. They handled a more complex class of word problems which involve solving a set of simultaneous linear equations. More recently, (Hosseini et al., 2014) attempted the same addition/subtraction word problem framework where they concentrated on classifying the verbs in a given problem into semantic classes. They were able to show remarkable improvement over (Kushman et al., 2014). They used a state representation to depict temporal ordering. However, the system could not reason about existing scenarios like questions that involved comparisons. There is some development in this direction by the mathematical modelling conglomerate WolframAlpha (2015).

## Project:

### Preprocessing:

This section explains the preprocessing done in the project for a given probability question. I used python nltk (Natural Language ToolKit) for preprocessing the data.

#### The preprocessing steps:

- Converting to Lower case
- Tokenization
- Lemmetization

The problem
If two coins are tossed ,What is the probability of 2 heads?

The problem after converting to lowercase
if two coins are tossed ,what is the probability of 2 heads?

The problem after tokenization
['if', 'two', u'coins', 'are', 'tossed', ',', 'what', 'is', 'the', 'probability', 'of', '2', u'heads', '?']

The problem after lemmetization
['if', 'two', u'coin', 'are', 'tossed', ',', 'what', 'is', 'the', 'probability', 'of', '2', u'head', '?']

#### Parts of speech tagging

The tokenized problem is passed to nltk.pos\_tag() for assigning parts of speech tags to the tokens.

The problem after tokenization
['if', 'two', u'coins', 'are', 'tossed', ',', 'what', 'is', 'the', 'probability', 'of', '2', u'heads', '?']

The parts of speech tags for the words
[('if', 'IN'), ('two', 'CD'), ('coins', 'NNS'), ('are', 'VBP'), ('tossed', 'VBN'), (',', ','), ('what', 'WP'), ('is', 'VBZ'), ('the', 'DT'), ('probability', 'NN'), ('of', 'IN'), ('2', 'CD'), ('heads', 'NNS'), ('?', '.'), ('.', '.')] ]

The CD tag for a token represents the token is a number.

Token	Parts of speech tag
2	CD
Two	CD

## Algorithm:

1. Find all the CD tag tokens in the problem
2. Convert the number in the string to the integers
3. Store all the numbers in the question in an array called number array.
4. Find the position of coins, head and tail in the question.
5. Sort the coins, head and tail by the position they occur in the question and store them in an array called position array
6. Assign the numbers in number array to the parameters in the position array in the same order they occur.
7. Find all the possible cases from the number of coins parameter.
8. For each case find whether the case satisfies the number of heads and tails parameter
9. Divide the number of cases that satisfies the parameters by the total number of parameters and return the result.

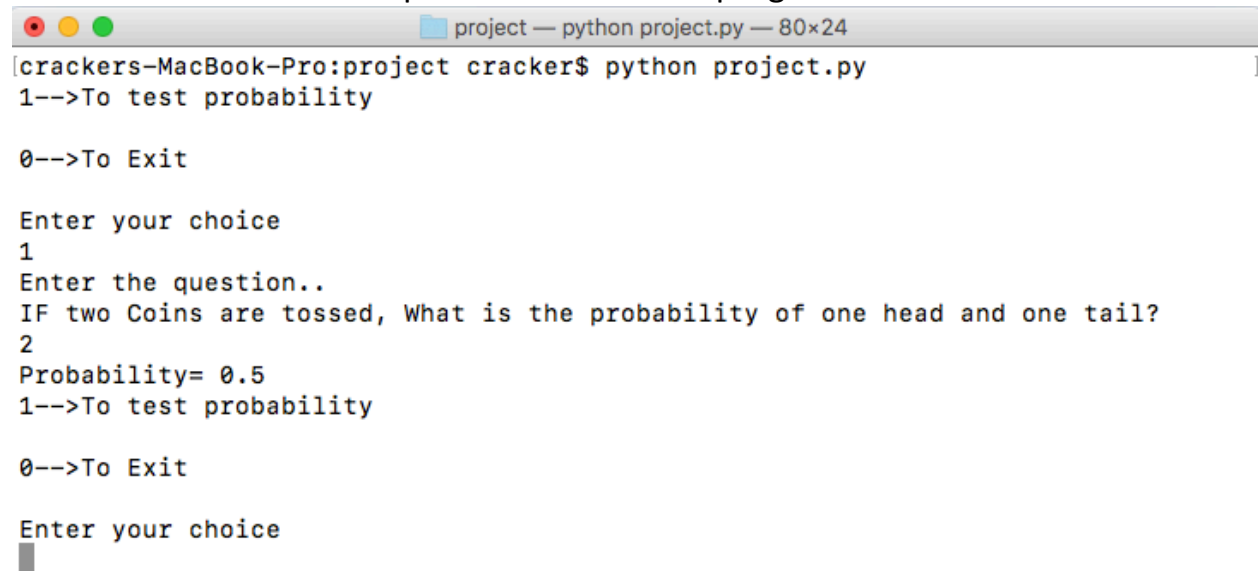
## Example:

The problem
If two coins are tossed what is the probability of seeing one head and one tail.

Step 1: Finding the CD tagged tokens
[Two, one, one]
Step 2 and Step 3: Convert the numbers to integers and store them in an array
[2,1,1]
Step 4: Find the position of coins, head and tail
[3,13,16]
Step 5: Sort them and store them in an array
[coin, head, tail]
Step 6: Assigning the numbers to the values in the position array
[coins=2, head=1,tail=1]
Step 7: Find all the possible cases
<p>Given coins=2</p> <p>Possible cases are [(h, h), (h, t), (t, h),(t, t)]</p> <p>Total possible cases=4</p>
Step 8: Finding the cases satisfying the problem
<p>Given parameters one head and one tail.</p> <p>Cases satisfying the problem</p> <p>(h, h) =False</p> <p>(h, t) =True</p> <p>(t, h) =True</p> <p>(t, t) =False</p> <p>Total cases that satisfy the problem=2</p>
Step 9: Return result
<p>Total cases=4</p> <p>Cases that satisfy the problem=2</p> <p>Probability=2/4</p> <p>=-.5</p>

## Implementation:

The screenshots of the implementation of the program.



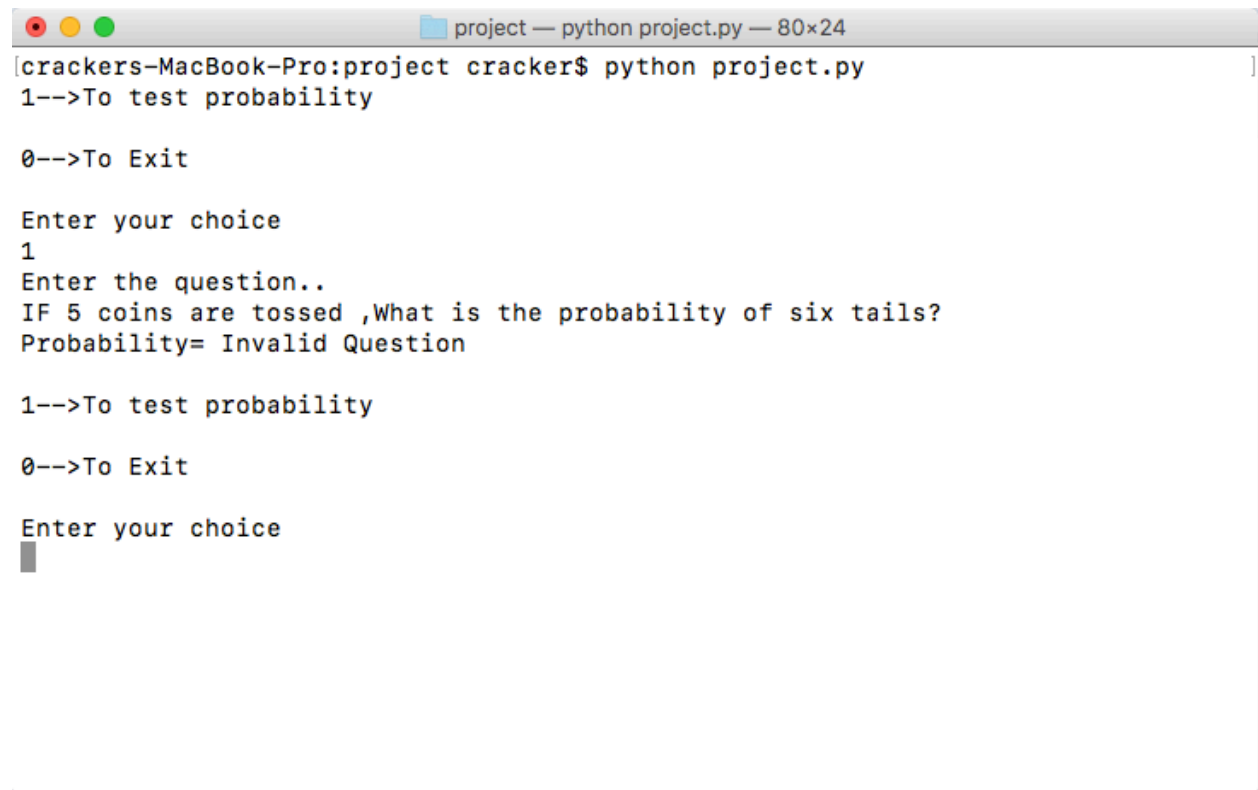
```
project — python project.py — 80x24
[crackers-MacBook-Pro:project cracker$ python project.py
1-->To test probability

0-->To Exit

Enter your choice
1
Enter the question..
IF two Coins are tossed, What is the probability of one head and one tail?
2
Probability= 0.5
1-->To test probability

0-->To Exit

Enter your choice
█
```



```
project — python project.py — 80x24
[crackers-MacBook-Pro:project cracker$ python project.py
1-->To test probability

0-->To Exit

Enter your choice
1
Enter the question..
IF 5 coins are tossed ,What is the probability of six tails?
Probability= Invalid Question

1-->To test probability

0-->To Exit

Enter your choice
█
```

```
project — python project.py — 80x24
crackers-MacBook-Pro:project cracker$ python project.py
1-->To test probability

0-->To Exit

Enter your choice
1
Enter the question..
What is the probability of three heads and one tail, if four coins are tossed?
1
Probability= 0.25
1-->To test probability

0-->To Exit

Enter your choice
█
```

```
project — python project.py — 80x24
crackers-MacBook-Pro:project cracker$ python project.py
1-->To test probability

0-->To Exit

Enter your choice
1
Enter the question..
If nine coins are tossed,what is the probability of 2 heads and five tails?
2
Probability= 0.48046875
1-->To test probability

0-->To Exit

Enter your choice
█
```

## Code:

The algorithm is implemented in python.

```
from nltk.tokenize import sent_tokenize
from nltk.tokenize import word_tokenize
import nltk
from nltk.stem.porter import PorterStemmer
from nltk.stem import WordNetLemmatizer
import math
def str2int(s):
    numbers={'one':1,'two':2,'three':3,'four':4,'five':5,'six':6,'seven':7,'eight':8,'nine':9,'zero':0}
    if(s.isdigit()):
        return int(s)
    elif(s==''):
        return -1
    else:
        return numbers[s]
class coins(object):
    def __init__(self,postags,words):
        self.numbers=[]
        self.get_numbers(postags)
        self.keyword=' '
        self.keywords=1
        self.head=' '
        self.tail=' '
        self.coinindex=0
        self.headindex=0
        self.tailindex=0
        self.keywordindex=0
        self.h=0
        self.t=0
        if('head' in words and 'tail' in words):
            self.head='head'
            self.tail='tail'
            self.headindex=words.index('head')
            self.tailindex=words.index('tail')
            self.coinindex=words.index('coin')
            self.keywords=2
        else:
            if('head' in words):
                self.keyword='head'
                self.keywordindex=words.index('head')
                self.coinindex=words.index('coin')
                self.keywords=1
```



```

        elif('tail' in words):
            self.keyword='tail'
            self.keywordindex=words.index('tail')
            self.coinindex=words.index('coin')
            self.keywords=1

        self.post=True
        if(self.keywords==2):

            if(self.coinindex>(max(self.headindex,self.tailindex))):
                print '1'
                self.post=True
                if(self.headindex>self.tailindex):
                    self.h=self.numbers[1]
                    self.t=self.numbers[0]

                else:
                    self.h=self.numbers[0]
                    self.t=self.numbers[1]

            else:
                print '2'
                self.post=False
                if(self.headindex>self.tailindex):
                    self.h=self.numbers[2]
                    self.t=self.numbers[1]

                else:
                    self.h=self.numbers[1]
                    self.t=self.numbers[2]

        elif(self.keywords==1):
            if(self.coinindex>self.keywordindex):
                self.post=True

            else:
                self.post=False

    def get_numbers(self,postags):
        self.cds=[' ',' ',' ']
        self.index=-1;
        self.preindex=-1;
        for word in range(len(postags)):
            if(postags[word][1]=='CD'):
                if(word==(self.preindex+1)):
                    if(self.index==-1):
                        self.index=0
                        self.cds[self.index]
+=postags[word][0]

                else:
                    self.index+=1

```

```

self.cds[self.index]
=self.cds[self.index]+postags[word][0]
self.previndex=self.previndex+1
for s in self.cds:
    self.numbers.append(str2int(s))
def print_variables(self):
    print self.numbers
    print self.keyword
    print self.coinindex
    print self.keywordindex
    print self.post
    print self.head
    print self.headindex
    print self.tail
    print self.tailindex
    print self.keywords
    print self.h
    print self.t
def find_probability(self):
    self.cases=[]
    self.appcases=0
    if(self.keywords==1):
        if(not self.post):
            for i in
range(int(math.pow(2,self.numbers[0]))):
                self.case=[]
                self.bits='{0:09b}'.format(i)
                for c in self.bits[9-
self.numbers[0]:]:
                    if(c=='1'):
                        self.case.append(1)
                    else:
                        self.case.append(0)
                self.cases.append(self.case)
            else:
                for i in
range(int(math.pow(2,self.numbers[1]))):
                    self.case=[]
                    self.bits='{0:09b}'.format(i)
                    for c in self.bits[9-
self.numbers[1]:]:
                        if(c=='1'):
                            self.case.append(1)
                        else:

```

```

                                self.case.append(0)
                                self.cases.append(self.case)
self.appcases=0.0
if(not self.post):
    if(self.numbers[1]>self.numbers[0]):
        return "Invalid Question\n"
    else:
        for case in self.cases:
            if(self.keyword=='head'):

if(sum(case)==self.numbers[1]):

self.appcases+=1

                                if(self.keyword=='tail'):
                                    if((self.numbers[0]-
sum(case))==self.numbers[1]):

self.appcases+=1

                                return
self.appcases/len(self.cases)
    else:
        if(self.numbers[0]>self.numbers[1]):
            return "Invalid Question\n"
        else:
            for case in self.cases:
                if(self.keyword=='head'):

if(sum(case)==self.numbers[0]):

self.appcases+=1

                                if(self.keyword=='tail'):
                                    if((self.numbers[1]-
sum(case))==self.numbers[0]):

self.appcases+=1

                                return self.appcases/len(self.cases)
    else:
        if(not self.post):
            for i in
range(int(math.pow(2,self.numbers[0]))):
                self.case=[]
                self.bits='{0:09b}'.format(i)
                for c in self.bits[9-
self.numbers[0]:]:

                                if(c=='1'):

```

```

        self.case.append(1)
    else:
        self.case.append(0)
    self.cases.append(self.case)

    else:
        for i in
range(int(math.pow(2,self.numbers[2]))):
            self.case=[]
            self.bits='{0:09b}'.format(i)
            for c in self.bits[9-
self.numbers[2]:]:
                if(c=='1'):
                    self.case.append(1)
                else:
                    self.case.append(0)
            self.cases.append(self.case)
        self.appcases=0.0
        if(self.post):
            if((self.h+self.t)>self.numbers[2]):
                return "Invalid Question\n"
            else:
                for case in self.cases:
                    if(sum(case)>=self.h and
(self.numbers[2]-sum(case))>=self.t):
                        self.appcases+=1
                return
        self.appcases/len(self.cases)
    else:
        if((self.h+self.t)>self.numbers[0]):
            return "Invalid Question\n"
        else:
            for case in self.cases:
                if(sum(case)>=self.h and
(self.numbers[0]-sum(case))>=self.t):
                    self.appcases+=1
            return self.appcases/len(self.cases)

def main():
    ps=PorterStemmer()
    wl=WordNetLemmatizer()
    print "1-->To test probability\n"
    print "0-->To Exit\n"
    choice=input("Enter your choice\n")
    while(choice!=0):

```

```

question=raw_input("Enter the question..\n").lower()
sent_words=word_tokenize(question)
pos_tags=nltk.pos_tag(sent_words)
for i in range(len(sent_words)):
    sent_words[i]=wl.lemmatize(sent_words[i])

coin=coins(pos_tags,sent_words)
#coin.print_variables()
print "Probability=" ,coin.find_probability()
print "1-->To test probability\n"
print "0-->To Exit\n"
choice=input("Enter your choice\n")
if __name__=="__main__":
    main()

```

## Future work:

Currently the algorithm can only handle the parameters which has values between 0 and 9. In the future this value can be increased to include all the possible integers. Currently the algorithm can only handle the coin problem in the future it can be updated to include all classes of probability problems like dices, balls in a bucket etc.

## Conclusion:

The papers read during the project gave a good understanding of the word currently done in the field on NLP on the word problems. The project has very limited capability and in the future hope to increase the capability of the algorithm to include different classes of probability problems.