Goldilocks Algorithm: Search for the Perfect IMEter

Nivedita Attada
B.S. Global Disease Biology, B.S. Cognitive Science
Graduation: June 2022

Ian Korf
Research Advisor
Professor, Department of Molecular and Cellular Biology

Abstract

Gene expression can be enhanced based on the location of introns. Based on if an intron is closer to the proximal end or distal end of a gene, we can tell how much it impacts gene expression; proximal introns contribute more to gene expression than distal introns. In order to find an algorithm that predicts the gene expression levels, I implemented two versions of the IMEter algorithm. Version 1 scored introns based on whether they were proximal or distal whereas version 3 scored introns based on assigning weight to each kmer in the intron based on its location. My results showed that version 3 resulted in a slightly better fit to the test set than version 1. Additionally, I noticed that using a geometric equation to weigh the kmers did not make much of a difference as opposed to a linear equation in version 3.

Introduction

Genes hold basic information on which traits are displayed and which traits are more dominant as opposed to others. Introns are smaller chunks of DNA that do not code for proteins but contribute to increasing gene expression. Studies have been done in various organisms to show how introns impact transgenes, which is new DNA that is introduced into the organism. For example, two separate studies done in mice and Drosophila melanogaster reported that introns increased the expression of transgenes in these organisms [1][2]. This discovery that introns can enhance gene expression is called intron mediated enhancement or IME [3]. This can have major implications for plants because they have the potential to increase favorable properties of plants used in agriculture. This can potentially protect against plant diseases, increase nutrient content, or increase yield which could also help for social causes such as hunger crises.

A few plants have also been studied in the context of IME such as maize, rice (Oryza sativa), and Arabidopsis thaliana. In one study, it was shown that intron presence and placement impacted gene expression depending on where it was located in maize cells [4]. The results of the studies in maize cells and mice showed that an intron located near the 5' end of the strand of mRNA increases the gene expression of genes [2][4]. In another study on rice, it was found that an intron influenced gene expression and the location of the intron determined whether the gene was expressed or not [5]. Intron location is important in how much the gene is expressed. Proximal introns contribute more to gene expression than distal introns [6].

Although we know that introns predict gene expression, we do not know how to calculate the distance between introns and how much that corresponds with gene expression. Kmer size and the threshold to determine what is proximal versus distal can be variable but it was found that optimal threshold and kmer size were 400 base pairs and 5 respectively [6]. The IMEter algorithm that I used as a model calculated the log-odds

score based on whether an intron was proximal or distal using data from Arabidopsis thaliana.

In this paper, I explore how I could score the introns of Arabidopsis thaliana to find an algorithm that uses the IMEter score to predict gene expression. I worked on implementing two methods of IMEter which are called version 1 and version 3. Version 1 mirrored the model algorithm and version 3 used a different method to score the introns. My goal was to find which of two IMEter algorithms would predict the level of intron mediated enhancement better. I used different values to weigh the kmers that make up the introns in each of my algorithms. I found that overall version 3 predicted intron mediated enhancement slightly better than version 1.

Methods

IMEter Trainer V1

IMEter trainer V1 is used to find the frequency of each kmer given. In this version of IMEter, each kmer is not given a weight and is scored based on whether an intron begins before or after the 400 base pair mark regardless of length.

```
def getAllKmers(size):
    allKmers = {}
    for kmer in itertools.product('ACGT', repeat=size):
        kmer = ''.join(kmer)
        allKmers[kmer] = 0
    return allKmers
```

Kmers are a combination of nucleotides of a given length. This function creates a dictionary and stores every possible combination of kmers given their size. For example, each nucleotide can be made up of A,C,G or T and if there is a size specified, this function would find every possible combination of 5 with these four letters.

```
def getFrequency(kmers):
    total = 0
    for k in kmers:
        total += kmers[k]
    frequency = {}
    for kmer in kmers:
        frequency[kmer] = kmers[kmer] / total
    return frequency
```

getFrequency is a function that takes in a dictionary of Kmers like the one created using getAllKmers and calculates how often each kmer occurs and produces a dictionary with each Kmer and its value being the frequency.

```
D = 5 #length of splice donor site
K = 5 #kmer size
A = 10 #length of splice acceptor site
proximal_count = getAllKmers(K) #counts of each kmer – proximal
distal_count = getAllKmers(K) #counts of each kmer – distal
```

Here, we have all the variables that we use in the upcoming sections. D, K, and A are length of splice site, kmer size, and length of splice acceptor site. These are essential to scoring the introns.

```
f = gzip.open(args.trainer, 'rt')
while True:
    line = f.readline()
    if line == '': break

    data = line.split()
    begin = data[1]
    strand = data[3]
    seq = data[-1]
    if strand != '+':
        continue

    for i in range(D,len(seq)-K-A+1):
        kmer = seq[i:i+K]
        if int(begin) < 400:
            if kmer in proximal_count:
                proximal_count[kmer] += 1
        else:
            if kmer in distal_count:
                distal_count[kmer] += 1
f.close()
```

This part of the program takes in a file which contains the data to be analyzed which would be each intron along with some additional data. It goes through each intron looking at a small portion of the size specified (size of the kmer previously stated) and keeps track of the number of each kmer present in the intron. In version 1, we use a cutoff of 400 base pairs, which means every kmer in the intron that begins before 400 will be considered as proximal and every intron that starts after will be distal. This is important because this is what helps us predict the expression levels. The for-loop displayed above adds one count based on whether the intron is proximal or distal.

```
proximal_freq = getFrequency(proximal_count)
distal_freq = getFrequency(distal_count)

for kmer in proximal_freq:
    print(kmer, math.log2(proximal_freq[kmer]/distal_freq[kmer]))
```

After we find the counts, we use the function previously created to find the frequency of each kmer using a log odds calculation.

IMEter V3 Linear Trainer

This program uses a linear function to weight each kmer. The kmers before and near the offset are weighted more than the kmers at the distal end.

```
def linear(num, start, slope):
    if num < start:
        p_weight = 1.0
        d_weight = 0.0
    elif (num - start) * slope >= 1:
        p_weight = 0.0
        d_weight = 1.0
    else:
        d_weight = (num - start) * slope
        p_weight = 1 - d_weight
    return p_weight, d_weight
```

In the version 3 of the program, we weight each kmer rather than assigning it a value of 1 because the kmers that are more proximal are much more significant and lose significance in regards to gene expression as they become more distal. In the linear trainer we use a slope to weigh the kmers in a linear fashion. For example, if the slope was 1/100, we would weigh each kmer based on how far on the intron based on this linear scale.

IMEter V3 Geometric Trainer

This program uses an exponential function to weight each kmer. The kmers at the proximal end of the intron are weighed much heavier than the kmers at the end. We tested different decay rates and offsets to find the optimal weights of the kmers.

```
def geometric(num, start, decay_rate):
    if num < start:
        p_weight = 1.0
        d_weight = 0.0
    else:
        p_weight = (1.0 - decay_rate) ** (num - start)
        d_weight = 1 - p_weight
    return p_weight, d_weight
```

In order to compare the linear trainer to a different model, we also programme a geometric trainer which weighs the kmers based on how proximal or distal they are using a different equation. Instead of calculating them linearly, we use exponential weights because we expected proximal weights to be significantly greater than distal weights.

IMEter Decoder

This program takes in a file that is produced from the trainers above and scores introns. At the end, we can see that the IMEter scores produced from this program are comparable to gene expression. The overall goal is to find the best prediction of gene expression.

```
def read_fasta(filename):

    name = None
    seqs = []

    fp = get_filepointer(filename)

    while True:
        line = fp.readline()
        if line == '': break
        line = line.rstrip()
        if line.startswith('>'):
            if len(seqs) > 0:
                seq = ''.join(seqs)
                yield(name, seq)
                name = line[1:]
                seqs = []
            else:
                name = line[1:]
        else:
            seqs.append(line)
    yield(name, ''.join(seqs))
    fp.close()
```

This section of code was borrowed from previously existing code and takes in a fasta file and allows us to read the data contained in it.

```
def imeter(seq, K, model, D=5, A=10):
    s = 0
    for i in range(D,len(seq)-K-A+1):
        kmer = seq[i:i+K]
        s += model[kmer]
    return s

model = {}
K = None
fp = open(args.model)
for line in fp.readlines():
    kmer, score = line.split()
    K = len(kmer)
    model[kmer] = float(score)



for name, seq in read_fasta(args.introns):
    s = imeter(seq, K, model)
    print(name, s)
```

For the decoder, we finally score the introns using all the Kmer scores we produced using the trainers. This score allows us to make predictions on the expression level of a given intron based on how proximal or distal it is. Higher IME scores correspond with higher expression levels.

R^2 Analysis

This program takes the IMEter scores I calculated in the previous programs and compares them to the expression values of the introns that we already know in order to check how similar or different they are. Since my goal is to find the version of IMEter that best predicts intron expression levels, this is the value I analyze to find the algorithm that best predicts which introns enhance gene expression.

```
f = open(args.introns)
exp = []
ime = []
while True:
    line = f.readline()
    if line == '': break

    data = line.split()
    exp.append(float(data[2][1:]))
    ime.append(float(data[-1]))
```

I made an R^2 analysis program so that we are able to calculate the R^2 and find how accurate the program is to known values so that we can find the best model for predicting gene expression in other introns. The section of code above creates two lists holding the expression level and IME scores for the calculation of the R^2 value in the next section.

```
X = np.array(exp).reshape(-1,1)
y = ime
reg = LinearRegression().fit(X, y)
r2 = str(reg.score(X,y))
txt = 'Offset: ' + str(args.offset) + ' Slope: ' + args.slope + ' R^2: '+r2 + '\n'
f = open('Results','a')
f.write(txt)
f.close()
```

Above, I used machine learning to calculate the R^2 given IME scores and expression levels which we produced using the trainers and decoder. I used this later to find the best R^2 which we expected to show us which model would predict gene expression best.

Runner

The runner puts together all of the programs and instead of having to manually change each parameter, it allows me to run just this one program and test each different combination of parameters for offset and slope. This was used for IMEter version 3 since offset and slope needed to change in each trial.

The runner was implemented to take on the tedious task of running all of my programs for various test cases and compiling them into a file without having to do it myself. I ran this program and left my program for a while and ended up with the results.

```
offset = []
slope = []
rate = []
for i in range(1001):
    if i % 100 == 0 and i != 0:
        offset.append(i)
        slope.append(1/i)
        rate.append('1/'+str(i))
```

The offset, slope, and rate were three variables that we were testing and changed in every test case so this for loop changes the test case after every time the program is run so we can get every combination of test cases.

```
for off in offset:
    for i in range(len(slope)):
        trainer = f'python3 geometric_trainer.py at_ime_master.txt.gz {off} {slope[i]} > mod'
        os.system(trainer)
        decoder = f'python3 decoder.py mod db_IME_Rose_WT_introns.fa > introns'
        os.system(decoder)
        r2 = f'python3 r2_analysis.py introns {off} {rate[i]}'
        os.system(r2)
```

This second for loop is where the program communicates with the command line to run the exact programs I need in their specific order. First, it starts by running the trainer depending on whether I was using the V1 trainer, the linear trainer, or the geometric trainer. Next, it runs the decoder and saves the output into a file and finally, it runs the R^2 analysis program which yields the results that we can use for our comparison later on. I originally started by manually running each program but realized that I could do all the iterations for a specific trainer using this method.

Results

Version	Offset	Slope	R^2 Value
IMEter Version 1	-	-	0.658
IMEter Version 3: Linear	200	1/200	0.699
IMEter Version 3: Geometric	200	1/100	0.696

In version 1, we classified each intron as proximal or distal and weighed each kmer the same. However, this brings into question if each kmer in the intron should be weighed equally. In version 3, instead of weighing each kmer with the same value, I tested two different weight scales: linear and geometric. The linear scale used the slope of a line to determine how much weight to assign as the kmers become more distal. It assigns a weight of 1 to proximal kmers and we designate an offset which is the cutoff between proximal and distal kmers. After the offset, each kmer after gets weighted according to the linear scale until it reaches a score of 0 after which each kmer has no

value. This is because we know that kmers towards the distal end do not contribute to gene expression. The geometric works in the same way except it uses an exponential equation. Therefore, kmers will never truly reach a score of 0 but rather an asymptote close to 0. I tested different offsets from 100 base pairs to 1000 base pairs and different slopes from 1/100 to 1/1000 and then found the optimal R^2 to be at an offset of 200 and slope of 1/200 for linear and offset of 200 and slope of 1/100 for the geometric model.

After running the IMEter algorithm with the R^2 analysis program for version 1 and version 3, I was able to see that the accuracy of predicting intron mediated enhancement improved, on average, by 0.04 in version 3. However, the difference between the linear and geometric model is not as significant. I also noticed that while testing the linear and geometric scoring, that the slope and offset of the optimal R^2 value were extremely similar which shows us that it does not make a big difference whether we weight each kmer using a linear weight or a geometric weight.

Discussion

In this study, I tried to find which implementation of IMEter had the higher R^2 value. The R^2 value would tell me which weight model best fits what occurs in biology. After testing version 1 and version 3 of IMEter, I noticed that version 3 overall had a higher R^2 value than version 1. This tells me that the location of the introns can depend on length and location and gene expression is better represented by a scale rather than just classifying it as proximal or distal.

From my results, I can conclude that version 3 fits real world data of Arabidopsis thaliana slightly better than version 1. I think this is because in version 1, kmers were all weighed the same. If an intron started one base pair before the cut off, it would be classified as proximal and if an intron started in the proximal region and was extremely long, every kmer would weigh the same. These can make the prediction of gene expression inaccurate because some of these introns may not actually increase gene expression as much. This is why we are able to see that version 3 is more accurate. It allows us to weigh proximal kmers more heavily than distal, allowing for a better representation of how much the intron contributes to gene expression.

I also noticed that the difference in R^2 value between the linear and geometric models of version 3 had a really small difference and is negligible. This showed me that we cannot conclude that an exponential curve is different from a linear slope when representing gene expression in A. thaliana. Trying different functions, like a parabolic function, might represent the gene expression data better and could be one possible way to expand on these findings.

Next steps could include testing version 1 and both models of version 3 on more data. Since I only have one data set to compare against, it could be that this data set of A. thaliana introns was not the best fit for version 1 or 3 but these IMEter algorithms may be

a better fit for other biological data. This would also lead to a question of whether there is a best fit IMEter for introns or whether it would differ from organism to organism. Additionally, another way to take this one step further would be to try different equations to weigh the kmers until the R^2 improves using trial and error. Recognizing patterns and what improves and what does not would allow us to combine what does work to find an IMEter algorithm that represents the data that we have.

References

1. Duncker, B. P., Davies, P. L., & Walker, V. K. (1997). Introns boost transgene expression in Drosophila melanogaster. Molecular & general genetics : MGG, 254(3), 291–296. https://doi.org/10.1007/s004380050418
2. Palmiter, R. D., Sandgren, E. P., Avarbock, M. R., Allen, D. D., & Brinster, R. L. (1991). Heterologous introns can enhance expression of transgenes in mice. Proceedings of the National Academy of Sciences of the United States of America, 88(2), 478–482. https://doi.org/10.1073/pnas.88.2.478
3. Laxa M. (2017). Intron-Mediated Enhancement: A Tool for Heterologous Gene Expression in Plants?. Frontiers in plant science, 7, 1977. https://doi.org/10.3389/fpls.2016.01977
4. Callis, J., Fromm, M., & Walbot, V. (1987). Introns increase gene expression in cultured maize cells. Genes & development, 1(10), 1183–1200. https://doi.org/10.1101/gad.1.10.1183
5. Jeon, J. S., Lee, S., Jung, K. H., Jun, S. H., Kim, C., & An, G. (2000). Tissue-preferential expression of a rice alpha-tubulin gene, OsTubA1, mediated by the first intron. Plant physiology, 123(3), 1005–1014. https://doi.org/10.1104/pp.123.3.1005
6. Rose, A. B., Elfersi, T., Parra, G., & Korf, I. (2008). Promoter-proximal introns in Arabidopsis thaliana are enriched in dispersed signals that elevate gene expression. The Plant cell, 20(3), 543–551. https://doi.org/10.1105/tpc.107.057190
7. Gallegos, J. E., & Rose, A. B. (2015). The enduring mystery of intron-mediated enhancement. Plant science : an international journal of experimental plant biology, 237, 8–15. https://doi.org/10.1016/j.plantsci.2015.04.017
8. Parra, G., Bradnam, K., Rose, A. B., & Korf, I. (2011). Comparative and functional analysis of intron-mediated enhancement signals reveals conserved features among plants. Nucleic acids research, 39(13), 5328–5337. https://doi.org/10.1093/nar/gkr043