CSE182-L6

P-value computation
Dictionary matching
P-value

- **P-value(11):** probability that a specific value (11) or something more extreme is achieved by chance.
- **E-value(11):** The number of times we expect to see a specific value or something more extreme just by chance.
- **E.g.**, look at alignment scores obtained by chance
  - 1, 2, 8, 3, 5, 3, 6, 12, 4, 4, 1, 5, 3, 6, 7, 15
- **Compute a Distribution**
  - 1–2         xxx
  - 3–4         xxxx
  - 5–6         xxxx
  - 7–8         xx
  - 9–10
  - 11–12       x
  - 12–13
  - 14–15       x
- **P-value (11) = 2/15**
Distribution

• Given a collection of numbers (scores)
  – 1, 2, 8, 3, 5, 3, 6, 4, 4, 1, 5, 3, 6, 7,....

• Plot its distribution as follows:
  – X-axis = each number
  – Y-axis (count/frequency/probability) of
    seeing that number
  – More generally, the x-axis can be a range
    to accommodate real numbers
P-value computation

• Goal: Compute P-value(x)

• A simple empirical method:
  • Compute a distribution of scores against a random database.
  • Use an estimate of the area under the curve to get the probability.
  • OR, fit the distribution to one of the standard distributions.
Z-scores for alignment

• Initial assumption was that the scores followed a normal distribution.

• Z-score computation:
  – For any alignment, score $S$, shuffle one of the sequences many times, and recompute alignment. Get mean and standard deviation

\[ Z_s = \frac{S - \mu}{\sigma} \]

• Look up the Z-score in a table to get a P-value

\[ f(Z) = \frac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}} \]
Blast E-value

• Initial (and natural) assumption was that scores followed a Normal distribution
• 1990, Karlin and Altschul showed that ungapped local alignment scores follow an exponential distribution
• Practical consequence:
  – Longer tail.
  – Previously significant hits now not so significant
Altschul Karlin statistics

- For simplicity, assume that the database is a binary string, and so is the query.
  - Let match-score=1,
  - mismatch score=-∞,
  - indel=-∞ (No gaps allowed)

- What does it mean to get a score k?
Large database search

Database (n)

Query (m)

Database size $n=100M$, Query size $m=1000$.

$O(nm) = 10^{11}$ computations
Observations

• For a typical query, there are only a few ‘real hits’ in the database.
• Much of the database is random from the query’s perspective
• Consider a random DNA string of length n.
  – $P_r[A]=P_r[C]=P_r[G]=P_r[T]=0.25$
• Assume for the moment that the query is all A’s (length m).
• What is the probability that an exact match to the query can be found? (P-value)
• What is the expected number of hits (E-value)
Basic probability

- Probability that there is a match starting at a fixed position $i = 0.25^m$
- What is the probability that some position $i$ has a match.
- Dependencies confound probability estimates.
Basic Probability: Expectation

• Q: Toss a coin many times: If it is HEADS, and the previous time was HEADS too, you get a dollar.
• What is the money you expect to get after n tosses?
  – Let $X_i$ be the amount earned in the i-th toss

\[ E(X_i) = 1 \cdot p + (0) \cdot (1 - p) = p \]

• Total money you expect to earn

\[ E\left( \sum_i X_i \right) = \sum_i E(X_i) = np \]
Expected number of matches

- Expected number of matches can still be computed.

  \[ \text{Pr(Match at Position } i \text{)} = p_i = 0.25^m \]
  \[ E(X_i) = p_i = 0.25^m \]

- Let \( X_i = 1 \) if there is a match starting at position \( i \), \( X_i = 0 \) otherwise.

  \[ E(\sum_i X_i) = \sum_i E(X_i) = n \left( \frac{1}{4} \right)^m \]
Expected number of exact Matches is small!

- Expected number of matches = $n \times 0.25^m$
  - If $n=10^7$, $m=10$,
    - Then, expected number of matches = 9.537
  - If $n=10^7$, $m=11$
    - expected number of hits = 2.38
  - $n=10^7$, $m=12$,
    - Expected number of hits = 0.5 < 1
- Bottom Line: An exact match to a substring of the query is unlikely just by chance.
Exponential distribution

- Random Database, Pr(1) = p
- What is the expected number of hits to a sequence of k 1’s

\[(n - k)p^k \approx ne^{k \ln p} = ne^{-k \ln \left(\frac{1}{p}\right)}\]

- Instead, consider a random binary Matrix. Expected # of diagonals of k 1s

\[\Lambda = (n - k)(m - k)p^k \approx nme^{k \ln p} = nme^{-k \ln \left(\frac{1}{p}\right)}\]
E-values for a simple score function

- If match-score=1, and all penalties are $-\infty$, then
- E-value =

$$\Lambda = (n - k)(m - k) p^k \approx nme^{k \ln p} = nme^{-k \ln \left(\frac{1}{p}\right)}$$

- E-value increases linearly with size of database
- E-value decreases exponentially with increasing scores
- What about P-value(k): Probability that there is at least one hit with score k?
• As you increase k, the number decreases exponentially.
• The number of diagonals of k runs can be approximated by a Poisson process

\[ \Pr[u] = \frac{e^{-\Lambda} \Lambda^u}{u!} \]

\[ \Pr[u > 0] = 1 - e^{-\Lambda} \]

• In ungapped alignments, we replace the coin tosses by column scores, but the behaviour does not change (Karlin & Altschul).
• As the score increases, the number of alignments that achieve the score decreases exponentially
E-values and P-values with more complex score functions

• How do we compute P-values and E-values with more complex score functions?
• Karlin and Altschul demonstrated that the similar functions (as in the simple case) apply.
Blast E-value

- Choose a score such that the expected score between a pair of residues < 0
- Expected number of alignments with a score $S$

\[
E = Kmne^{-\lambda S} = mn2^{-\left(\frac{\lambda S - \ln K}{\ln 2}\right)}
\]

\[
\Pr(S \geq x) = 1 - e^{-Kmne^{-\lambda x}}
\]

- The parameters $K, \lambda$ are computed empirically
- For small values, E-value and P-value are the same
WHY IS BLAST FAST?
True or False

• There are at least two people in NYC with the same amount (number) of hair.
• What if we exclude bald people?
• What is the pigeonhole principle?
Observation 2

• Suppose we are looking for a database string with greater than 90% identity to the query (length 100)
  • Partition the query into size 10 substrings. At least one much match the database string exactly
Why is this important?

• Suppose we are looking for sequences that are 80% identical to the query sequence of length 100.
• Assume that the mismatches are randomly distributed.
• What is the probability that there is no stretch of 10 bp, where the query and the subject match exactly?

\[ \left( 1 - \left( \frac{8}{10} \right)^{10} \right)^{90} = 0.000036 \]

• Rough calculations show that it is very low. Exact match of a short query substring to a truly similar subject is very high.
  – The above equation does not take dependencies into account
  – Reality is better because the matches are not randomly distributed
Just the Facts

• Consider the set of all substrings of the query string of fixed length $W$.
  – Prob. of exact match to a random database string is very low.
  – Prob. of exact match to a true homolog is very high.
  – Keyword Search (exact matches) is MUCH faster than sequence alignment
Speeding up via an exact match heuristics

• Consider a query string of length $m$
• A db string of length $n$
• Start by looking for exact matches of keywords of length $W$ between the query and database string.
  - Wherever, there is an exact match, perform a SW local alignment.
Why is BLAST fast?

- Assume that keyword searching does not consume any time and that alignment computation the expensive step.
- Query m=1000, random Db n=10^7, no TP
- SW = O(nm) = 1000*10^7 = 10^{10} computations
- BLAST, W=11
  - E(#11-mer hits) = 1000* (1/4)^11 * 10^7 = 2384
  - Number of computations = 2384*100*100 = 2.384*10^7
  - Ratio = 10^{10}/(2.384*10^7) = 420
- Further speed improvements are possible
Keyword (Dictionary) Matching

• How fast can we match keywords?
• Hash table/Db index? What is the size of the hash table, for m=11
• Suffix trees? What is the size of the suffix trees?
• Trie based search. We will do this in class.
The last step in Blast

• We have discussed
  – Alignments
  – Db filtering using keywords
  – Scoring matrices
  – E-values and P-values

• The last step: Database filtering requires us to scan a large sequence fast for matching keywords
Dictionary Matching

- Q: Given k words ($s_i$ has length $l_i$), and a database of size n, find all matches to these words in the database string.
- How fast can this be done?
Dict. Matching & string matching

• How fast can you do it, if you only had one word of length m?
  – Trivial algorithm $O(nm)$ time
  – Pre-processing $O(m)$, Search $O(n)$ time.
• Dictionary matching
  – Trivial algorithm $(l_1+l_2+l_3...)n$
  – Using a keyword tree, $l_p n$ ($l_p$ is the length of the longest pattern)
  – Aho-Corasick: $O(n)$ after preprocessing $O(l_1+l_2..)$
• We will consider the most general case
Direct Algorithm

Observations:
• When we mismatch, we (should) know something about where the next match will be.
• When there is a mismatch, we (should) know something about other patterns in the dictionary as well.
The Trie Automaton

- Construct an automaton $A$ from the dictionary
  - $A[v,x]$ describes the transition from node $v$ to a node $w$ upon reading $x$.
  - $A[u,'T'] = v$, and $A[u,'S'] = w$
  - Special root node $r$
  - Some nodes are terminal, and labeled with the index of the dictionary word.
An $O(l_p n)$ algorithm for keyword matching

\begin{align*}
l &= 1 \\
c &= 1 \\
v &= \text{root} \\
\text{repeat} \\
&\quad \text{if } ((w = A(v, T[c])) \neq \phi) \\
&\quad \quad v = w \\
&\quad \quad c = c + 1 \\
&\quad \quad \text{if } (v \text{ has label } i) \\
&\quad \quad \quad \text{print “Pattern } i \text{ matches starting at position } l” \\
&\quad \quad \text{else} \\
&\quad \quad \quad c = l + 1 \\
&\quad \quad \quad l = c \\
&\quad \quad \quad v = \text{root} \\
&\quad \text{end} \\
&\text{until } (c > n) /* n \text{ is the database size} */
\end{align*}

- Start with the first position in the db, and the root node.
- If successful transition
  - Increment current pointer
  - Move to a new node
  - If terminal node “success”
- Else
  - Retract ‘current’ pointer
  - Increment ‘start’ pointer
  - Move to root & repeat
Illustration:

\[
\begin{align*}
&l = 1 \\
&c = 1 \\
&v = \text{root} \\
\text{repeat} & \\
&\quad \text{if } (((w = A(v, T[c])) \neq \phi)) \\
&\quad \quad v = w \\
&\quad \quad c = c + 1 \\
&\quad \quad \text{if } (v \text{ has label } i) \\
&\quad \quad \quad \text{print “Pattern } i \text{ matches starting at position } l'' \\
&\quad \quad \text{else} \\
&\quad \quad \quad c = l + 1 \\
&\quad \quad \quad l = c \\
&\quad \quad \quad v = \text{root} \\
\text{end} & \\
\text{until } (c > n) \quad /* n \text{ is the database size} */
\end{align*}
\]
Idea for improving the time

- Suppose we have partially matched pattern i (indicated by l, and c), but fail subsequently. If some other pattern j is to match
  - Then prefix(pattern j) = suffix [first c-l characters of pattern(i)]
An O(n) alg. For keyword matching

\[ l = 1 \]
\[ c = 1 \]
\[ v = \text{root} \]

repeat
  if \((w = A(v, T[c]) \neq \phi)\)
    \[ v = w \]
    \[ c = c + 1 \]
    if \(v\) has label \(i\)
      print \(\text{"Pattern } i \text{ matches starting at position } l\)"
  else if \(v==\text{root}\)
    \[ c = c + 1 \]
    \[ l = c \]
  else
    \[ l = c - lp[v] \]
    \[ v = F[v] \]
  end
until \(c > n\)

- Start with the first position in the db, and the root node.
- If successful transition
  - Increment current pointer
  - Move to a new node
  - If terminal node “success”
- Else (if at root)
  - Increment ‘current’ pointer
  - Move ‘start’ pointer
  - Move to root
- Else
  - Move ‘start’ pointer forward
  - Move to failure node
Failure function

• Every node $v$ corresponds to a string $s_v$ that is a prefix of some pattern.
• Define $F[v]$ to be the node $u$ such that $s_u$ is the longest suffix of $s_v$
• If we fail to match at $v$, we should jump to $F[v]$, and commence matching from there
• Let $lp[v] = |s_u|$
Illustration

- What is $F(n_{10})$?
- What is $F(n_5)$?
- $F(n_3)$?
- $L_p(n_{10})$?
Illustration

\[
l = 1 \\
c = 1 \\
v = \text{root} \\
\text{repeat} \\
\quad \text{if } ((w = A(v, T[c]) \neq \emptyset) \\
\quad \quad v = w \\
\quad \quad c = c + 1 \\
\quad \quad \text{if } (v \text{ has label } i) \\
\quad \quad \quad \text{print "Pattern } i \text{ matches starting at position } l" \\
\quad \text{else if } (v == \text{root}) \\
\quad \quad c = c + 1 \\
\quad \quad l = c \\
\quad \text{else} \\
\quad \quad l = c - lp[v] \\
\quad \quad v = F[v] \\
\text{end} \\
\text{until } (c > n)
\]
Illustration

\[ l = 1 \]
\[ c = 1 \]
\[ v = \text{root} \]
repeat
  \[ (w = A(v, T[c]) \neq \phi) \]
  \[ v = w \]
  \[ c = c + 1 \]
  \[ \text{if} \ (v \text{ has label } i) \]
    \[ \text{print "Pattern } i \text{ matches starting at position } l" \]
  \[ \text{else if} \ (v = \text{root}) \]
    \[ c = c + 1 \]
    \[ l = c \]
  \[ \text{else} \]
    \[ l = c - lp[v] \]
    \[ v = F[v] \]
end
until \( (c > n) \)
Illustration

\[ l = 1 \]
\[ c = 1 \]
\[ v = \text{root} \]

repeat
  if \((w = A(v, T[c]) \neq \phi)\)
    \[ v = w \]
    \[ c = c + 1 \]
    if \((v\ \text{has label } i)\)
      print "Pattern i matches starting at position \(l\)"
  else if \((v==\text{root})\)
    \[ c = c + 1 \]
    \[ l = c \]
  else
    \[ l = c - lp[v] \]
    \[ v = F[v] \]
  end
until \((c > n)\)
Illustration

\[ l = 1 \]
\[ c = 1 \]
\[ v = \text{root} \]

repeat
\[ (w = A(v, T[c]) \neq \phi) \]
\[ v = w \]
\[ c = c + 1 \]
\[ \text{if } (v \text{ has label } i) \]
\[ \text{print } \text{“Pattern } i \text{ matches starting at position } F^t \text{”} \]
\[ \text{else if } (v == \text{root}) \]
\[ c = c + 1 \]
\[ l = c \]
\[ \text{else} \]
\[ l = c - lp[v] \]
\[ v = F[v] \]
\[ \text{end} \]
until \((c > n)\)
Illustration

\[ \text{P O T A S T P O T A T O} \]

\[ l = 3 \quad c = 7 \]

\[ l = 1 \]
\[ c = 1 \]
\[ v = \text{root} \]
\[ \text{repeat} \]
\[ \text{if } ((w = A(v, T[c]) \neq \phi)) \]
\[ \quad v = w \]
\[ \quad c = c + 1 \]
\[ \quad \text{if } (v \text{ has label } i) \]
\[ \quad \text{print “Pattern } i \text{ matches starting at position } l” \]
\[ \text{else if } (v = \text{root}) \]
\[ \quad c = c + 1 \]
\[ \quad l = c \]
\[ \text{else} \]
\[ \quad l = c - lp[v] \]
\[ \quad v = F[v] \]
\[ \text{end} \]
\[ \text{until } (c > n) \]
Illustration

\[
l = 7 \quad c = 7
\]

\[
\begin{align*}
&v = \text{root} \\
&\text{repeat} \\
&\quad \text{if} \ ((w = A(v, T[c]) \neq \emptyset) \\
&\quad \quad v = w \\
&\quad \quad c = c + 1 \\
&\quad \quad \text{if} \ (v \text{ has label } i) \\
&\quad \quad \quad \text{print} \ \text{"Pattern } i \text{ matches starting at position } l" \\
&\quad \quad \text{else if} \ (v = \text{root}) \\
&\quad \quad \quad c = c + 1 \\
&\quad \quad \quad l = c \\
&\quad \quad \text{else} \\
&\quad \quad \quad l = c - lp[v] \\
&\quad \quad \quad v = F[v] \\
&\quad \text{end} \\
&\text{until} \ (c > n)
\end{align*}
\]
Illustration

\[ \text{P} \quad \text{O} \quad \text{T} \quad \text{A} \quad \text{S} \quad \text{T} \quad \text{P} \quad \text{O} \quad \text{T} \quad \text{A} \quad \text{T} \quad \text{O} \]

\[ l = 7 \quad c = 8 \]

\begin{verbatim}
l = 1
c = 1
v = root
repeat
  if \((w = A(v,T[c]) \neq \phi)\)
    v = w
    c = c + 1
  if \((v \text{ has label } i)\)
    print "Pattern } matches starting at position }"
  else if \((v == \text{root})\)
    c = c + 1
    l = c
  else
    l = c - lp[v]
    v = F[v]
until \((c > n)\)
\end{verbatim}
Illustration

\[ \text{P O T A S T P O T A T O} \]

\[ l = 7 \quad c = 7 \]

\[ v = \text{root} \]

\[ \text{repeat} \]

\[ \text{if } (w = A(v,T[c]) \neq \emptyset) \]

\[ v = w \]

\[ c = c + 1 \]

\[ \text{if } (v \text{ has label } i) \]

\[ \text{print "Pattern } i \text{ matches starting at position } l" \]

\[ \text{else if } (v == \text{root}) \]

\[ c = c + 1 \]

\[ l = c \]

\[ \text{else} \]

\[ l = c - lp[v] \]

\[ v = F[v] \]

\[ \text{end} \]

\[ \text{until } (c > n) \]
Time analysis

• In each step, either c is incremented, or l is incremented.
• Neither pointer is ever decremented \((lp[v] < c-l)\).
• l and c do not exceed n
• Total time \(\leq 2n\)

```
l = 1
c = 1
v = root
repeat
  if ((w = A(v, T[c]) \neq \phi))
    v = w
    c = c + 1
  if (v has label i)
    print “Pattern i matches starting at position l”
  else if (v==root)
    c = c + 1
    l = c
  else
    l = c - \{p[v]
    v = F[v]
  end
until (c > n)
```
Blast: Putting it all together

- Input: Query of length $m$, database of size $n$
- Select word-size, scoring matrix, gap penalties, E-value cutoff
- [Blast](#)
Blast Steps

1. Generate an automaton of all query keywords.
2. Scan database using a “Dictionary Matching” algorithm (O(n) time). Identify all hits.
3. Extend each hit using a variant of “local alignment” algorithm. Use the scoring matrix and gap penalties.
4. For each alignment with score S, compute E-value, and the P-value. Sort according to increasing E-value until the cut-off is reached.
5. Output results.
BLAST output

Distance tree of results

Sequences producing significant alignments:

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Description</th>
<th>Score (Bits)</th>
<th>E Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>sp</td>
<td>P58308</td>
<td>OX2R_MOUSE</td>
<td>Orexin receptor type 2 (0x2r) (Hypocretin r</td>
</tr>
<tr>
<td>sp</td>
<td>P56719</td>
<td>OX2R_RAT</td>
<td>Orexin receptor type 2 (0x2r) (Hypocretin rec</td>
</tr>
<tr>
<td>sp</td>
<td>Q9TUP7</td>
<td>OX2R_CANFA</td>
<td>Orexin receptor type 2 (0x2r) (Hypocretin r</td>
</tr>
<tr>
<td>sp</td>
<td>O43614</td>
<td>OX2R_HUMAN</td>
<td>Orexin receptor type 2 (0x2r) (Hypocretin r</td>
</tr>
<tr>
<td>sp</td>
<td>Q6IPH4</td>
<td>OR6B2_HUMAN</td>
<td>Olfactory receptor 6B2 (Olfactory receptor</td>
</tr>
<tr>
<td>sp</td>
<td>P09703</td>
<td>US27_HCMVA</td>
<td>G-protein coupled receptor homolog US27 (HH</td>
</tr>
<tr>
<td>sp</td>
<td>Q4QXU5</td>
<td>MRG52_MACMU</td>
<td>Mas-related G-protein coupled receptor mem</td>
</tr>
<tr>
<td>sp</td>
<td>Q76916</td>
<td>PPAR2_RAT</td>
<td>Free fatty acid receptor 2 (G-protein couple</td>
</tr>
<tr>
<td>sp</td>
<td>Q8NH53</td>
<td>O52N1_HUMAN</td>
<td>Olfactory receptor 52N1</td>
</tr>
<tr>
<td>sp</td>
<td>Q8VG90</td>
<td>OL502_MOUSE</td>
<td>Olfactory receptor 502 (Olfactory receptor</td>
</tr>
<tr>
<td>sp</td>
<td>Q9EP86</td>
<td>NPFF1_RAT</td>
<td>Neuropeptide FF receptor 1 (G-protein cou...</td>
</tr>
</tbody>
</table>
Distant hits

**sp|P79242|FPRL1_PANTA**  
FMLP-related receptor I (FMLP-R-I)  
Length=348

Score = 62.0 bits (149), Expect = 5e-09, Method: Composition-based stats.  
Identities = 77/327 (23%), Positives = 136/327 (41%), Gaps = 62/327 (18%)

**Query 63**  
IVFVVALIGNVLVCAVWKNKHMRVTNYPFVNLNSLADVNASLTCLPATLV-VDITEVTF 121  
+ FV+ ++GN LV + V RTTV +NL+AD T T LP +V + +E W

**Sbjct 32**  
VFVVLGVQGLNGV-1WQAFGRMFTRTTVCYLNLALADSFPTAT-LPFLIVSMAMGEKWP 89

**Query 122**  
FGQSLCKVIPYLQTVSVSVLTLSCIALDRHWAICHPLMFKS--TAKRANNISIVVIW 179  
PG LCK+I + +++ SV + IALDR + HP+ ++ T A IV WI

**Sbjct 90**  
FGFLCLDKLIHVVDINLFGSVLIGFTIALDRCICVLHPWAQNHRVTSNLKIVGPWIL 149

**Query 180**  
SCIIMIIPQAIYVECMSMLPGANKTLTPTVCDGHWGGEVPKM--------YHICFFLV 230  
++ +P ++ + + + N T T WGG ++ I P++

**Sbjct 150**  
ALVLTLPVFLPTTTVY-1PMGHTYCTPFPASWGGTFEERLKVAITLTTARGIIRFV1 205

**Query 231**  
TYMAPLCMLILAYQIFPRKLCRQIPTGSSVQRKQKPQPSPQPRSGQGIQSARKISAVA 290  
+ P+ ++ + Y I K+ + + + S

**Sbjct 206**  
GFSLPMSIVAICYGLIAAKIHKKGMKSS------------ 324

**Query 291**  
AEIKQIRARRRKTAMLMLVLLVLPAYCILPSLNVKLKRVF--GMFTHTEDRETVAWFTF 348  
+ R+L V+ P IC+ P ++ L V+ M ++ ++ +

**Sbjct 235**  
----------RPLRLVTLAVASPFFICWPPLQVLALLGTWLVKEMLFYGG-YKIIDLVNP 283

**Query 349**  
SHWLVYANSAANPPIYNLPSGKPREF 375  
+ L + N NP++Y P + PRE

**Sbjct 284**  
TSSLAFFNCCNLNPMLVFVQDFPERL 310

**sp|Q29003|5HT1E_PIG**  
5-hydroxytryptamine 1E receptor (5-HT1E) (Serotonin recep 1E) (5-HT1E)  
Length=149

Score = 62.0 bits (149), Expect = 5e-09, Method: Composition-based stats.  
Identities = 33/99 (33%), Positives = 59/99 (59%), Gaps = 2/99 (2%)

**Query 90**  
NYIPIYVLNLADVLTCLPATLVVDITEWFCQQSLCKVIPYLTQTSVSVSLTSLSCIA 149  
NY I +L++ D+LV + +P +++ + +W G +C+V + + +S+L L IA

**Sbjct 5**  
NYLICSLAVTDLVLAVLVMPLSIMYVMDSWRLGYFICEVWLSVDHTCTCSILHLCVIA 64

**Query 150**  
LDRHWAICHPLMF--KSTAKRARNISIVVIWISCIIMIP 186  
LDR+AI + + + K TAKRA I+ +W +S I +P

**Sbjct 65**  
LDRYWAIATNCAEIYARKTAKRALGMLTVWTTISIFISMP 103
End of Lecture 6