UHASSELT
KNOWLEDGE IN ACTION

Maastricht University

Faculty of Sciences
School for Information Technology
Master of Statistics and Data Science

Master's thesis

Assessing performance of heuristic-based biological alignment algorithms

Violet Ankunda

Thesis presented in fulfillment of the requirements for the degree of Master of Statistics and Data Science, specialization Bioinformatics

SUPERVISOR :

Prof. dr. Dirk VALKENBORG

SUPERVISOR :

Dhr. Dirk VAN HYFTE

2020
2021

Transnational University Limburg is a unique collaboration of two universities in two countries: the University of Hasselt and Maastricht University.

UHASSELT
KNOWLEDGE IN ACTION

www.uhasselt.be
Universiteit Hasselt
Campus Hasselt:
Martelarenlaan 42 | 3500 Hasselt
Campus Diepenbeek:
Agoralaan Gebouw D | 3590 Diepenbeek

Maastricht University

Faculty of Sciences
School for Information Technology
Master of Statistics and Data Science

Master's thesis

Assessing performance of heuristic-based biological alignment algorithms

Violet Ankunda

Thesis presented in fulfillment of the requirements for the degree of Master of Statistics and Data Science, specialization Bioinformatics

SUPERVISOR :
Prof. dr. Dirk VALKENBORG

SUPERVISOR :
Dhr. Dirk VAN HYFTE

Acknowledgments

I thank the Almighty God for all the guidance and protection He has given me and seeing me through my education.
To my family especially my mother, Mrs Kyoheirwe Jenny, my brothers, Turibamwe Gordon and Tusiime Graham, my sisters, Abenawe Fortunate and Ahabwe Viola, thank you so much for all the support, love , care and encouragement.
I am deeply grateful to my supervisors, Prof. dr. Dirk Valkenborg and Dr. Dirk Van Hyfte for their dedication, support and encouragement throughout this master thesis period.
My heart felt gratitude also goes to the Biostrand team; Sébastien Lemal, Yegor Korovin, Christophe Van Neste, and Georgios Triantopoulos for sacrificing their time, supporting me and working tirelessly to make this master thesis successful. Thank you for being a good team to work with.
I equally appreciate my lecturers at Hasselt University and VLIR-UOS for this scholarship in order to purse my Master's degree.
I would also like to thank Dr John Mulindwa Kitayimbwa for his encouragement and guidance. Thank you for always believing in me.
Finally, to my friends; Moses Kivumbi (Uganda), Alma Muropa (Zimbabwe), Lembris Njotto (Tanzania), Josline Otieno (Kenya), Caroline Namanya (Uganda) and all others, thank you so much for all the support and encouragement.

Abstract

Sequence alignment is the process of comparing different sequences by searching for a series of individual characters. The most common Bioinformatics tool for sequence alignment is BLAST. Current state-of-the-art biological sequence alignment algorithms such as BLAST relies on heuristics and dynamical programming based on probabilistic models: this impacts their performance and scalability as they are prone to error propagation [1]. Moreover, these algorithms perform analysis within a so-called query window defined as the most similar region to that of the query sequence, with a risk of missing homologies outside that window which may possibly remain relevant.
The main purpose of this Master Thesis project is to evaluate the performance of BLAST, retrieve homologous sequences of given queries from a set of well known protein sequences, as well as evaluating how different metrics can be used for performance.
To study the variation in BLAST performance with respect to the default ones, BLAST was run against the PDB data while modifying one parameter at a time, ROC and precision-recall curves were plotted for various results of varied parameters. To assess the overall performance, area under the curve was calculated for each of the graphs.
The results indicated a marginal difference between the performance of BLAST using default parameters and modifying the parameters.
Key Words: *Sequence Alignment*, *BLAST*, *ROC*, *Precision-recall curve*

Contents

List of Tables

List of Figures

1 INTRODUCTION

1.1 Overview

Sequence alignment is the process of comparing different sequences by searching for a series of individual characters or character patterns that have the same arrangement in both sequences [2]. Sequence alignments are either local or global. Local alignment aims to find the best match between two sequences while global alignment aims to find the best match over the whole length of the sequences. Similarity searching is effective because proteins that share statistically significant sequence similarity can be inferred to be homologous, and homologous proteins share similar structures and, often, similar functions [3].
The discovery of sequence homology or similarity to a known protein or family of proteins often provides the first clues about, the function of a newly sequenced gene. As the DNA and amino acid sequence databases continue to grow in size they become increasingly useful in the analysis of newly sequenced genes and proteins because of the greater chance of finding such homologies [4]. The commonest Bioinformatics tool for evaluating the similarity between biological sequences is Basic Local Alignment Search Tool (BLAST) [4].

1.1.1 Basic Local Alignment Search Tool (BLAST)

BLAST is an algorithm used for comparison of amino acid sequences of different proteins or the nucleotides sequences of nucleic acid. BLAST was invented in 1990 and has since then become the defacto standard in search and alignment tools. Through a BLAST search, one can compare a query sequence with a database of sequences, and thereby identify library sequences that share resemblance with the query sequence above a certain threshold [2].
BLAST searches through a large database of known sequences to find sequences that are similar to the query sequence. This tool can be used via a web interface or as a stand-alone tool to compare a user's query to a database of sequences [5].
BLAST offers different types of searches and the choice of which one to use depends on the objective and the type of data used. Table 1 shows the different BLAST programs with the different types of data to be used:

Table 1: *Different BLAST programs*

Program	Query Sequence Type	Target Sequence Type
BLASTN	Nucleotide	Nucleotide
BLASTP	Protein	Protein
BLASTX	Nucleotide (translated)	Protein
TBLASTN	Protein	Nucleotide (translated)
TBLASTX	Nucleotide (translated)	Nucleotide (translated)
Source:	`http://www.ncbi.nlm.nih.gov/blast`	

The most common searches are BLASTN and BLASTP, but this projects focuses on BLASTP.
Besides sequence similarity searching, BLAST can be used to locate domains and identify folds

How BLAST works

BLAST is based on a heuristic algorithm [5]. A heuristic algorithm is an algorithm that is built to solve a problem faster and in a more efficient way.
Using the heuristic search, BLAST assumes that if two sequences are similar, there is a short sequence that they share in common. BLAST finds all possible short sequences in the query sequence and locates them in the large database. This process is called seeding. For example, for BLASTP, the default word size is 3, W=3. If a query sequence has ABCDE, the searched words are ABC, BCD, CDE. After synthesizing words for a given sequence of interest, neighborhood words are also assembled. Once both words and neighborhood words are organized, they are compared with the database sequences in order to find matches [2]. A scoring matrix for example BLOSUM62, BLOSUM45 is used to score each of the matched short sequences. Therefore, each short sequence will have an alignment score basing on which scoring matrix used. These short sequences are then extended in both directions by BLAST if the score is above a predefined threshold and if it can not extend further, it joins the residues with gaps. However this extension can either improve the alignment score or reduce the score. Once the sequence is extended as further as possible, the result is a High Scoring Pair (HSP). The HSP is scored and lastly the significance of the HSP score is evaluated. If the score is less than the predefined threshold, the extension of the alignment is stopped.

Since such algorithms like BLAST assume for two sequences to be similar, there is a short sequence common in both sequences, it performs analysis within a query window. It uses the short similar sequence in both sequences. Because of this, there is a risk of missing homologies outside that window which may be relevant. For example, given two protein sequence alignments below, basing on the word size defined, BLAST would find a short word similar in both sequences.

```
                P  Y  D  N  G  F  S  F  L  K  S  E  L  A  G
Alignment 1:                |  |  |  |  |  |
                A  G  L  S  G  F  S  F  L  K  A  M  F  D  R

                P  Y  D  N  G  F  S  F  L  K  S  E  L  A  G
Alignment 2:    |     |     |     |     |     |     |     |
                P  A  D  L  G  K  S  M  L  N  S  R  L  K  G
```

Sequences in alignment 1 share a word in common, '**GFSFLK**' and has 6 out of 15 identical amino acids. On the other hand, sequences in alignment 2 do not share a word in common, however has 8 out 15 identical amino acids. Because BLAST first locates the similar region, it may miss a case in alignment 2 and report the case in alignment 1.

1.2 Research Questions

The main objective of this project is to evaluate the performance of BLAST, to retrieve homologous sequences from a set of well-known proteins with respect to different parameters.
In addition to that, it is to evaluate how different metrics can be used for the performance measurement.
The results to the above objectives will provide a compilation of performance assessment to understand the strengths and weaknesses of BLAST.
This project also aims to formalize the methodology using state-of-the-art statistical analysis.
Lastly, is to use this framework to compare BLAST with other heuristic- based biological alignment algorithms.
In order to compare competing algorithms, we are investigating highly curated data sets for our benchmarking study. The benchmark data that is chosen comes from Structural Classification of Proteins-extended and Protein Data Bank and represents protein structures that are closely related. Therefore, in this master thesis, we conduct a sensitivity study that investigates how the results in terms of false positive and false negative will change upon different parameter settings for the BLAST algorithm. Such a study is important because for a comparison among competing algorithms it is of needed that BLAST is operating on the particular data set in an optimal way to avoid a flawed comparison.

1.3 Data Description

In this subsection, the data used for this project is described. To run the BLAST software, two sets of data are needed, that is to say, the query sequence to search for and the database to search against. The database to search against is also called the target sequence.

1.3.1 Protein Data Bank (PDB)

In this project, the PDB data was used as the target sequence. The PDB is an open source database for the three-dimensional structural data of large biological molecules, such as proteins and nucleic acids. The data is submitted by Biologists and Biochemists from all over the world who use X-ray crystallography or NMR spectroscopy to capture it. Proteins in the PDB database are uniquely identified by protein ids.

In the early days of the PDB, data were distributed via magnetic tape and later by CD-ROM. Now there is an ftp site, `https://www.wwpdb.org/ftp/pdb-ftp-sites` that contains the data in three formats: PDB, mmCIF-PDBx and PDBML-XML [7].

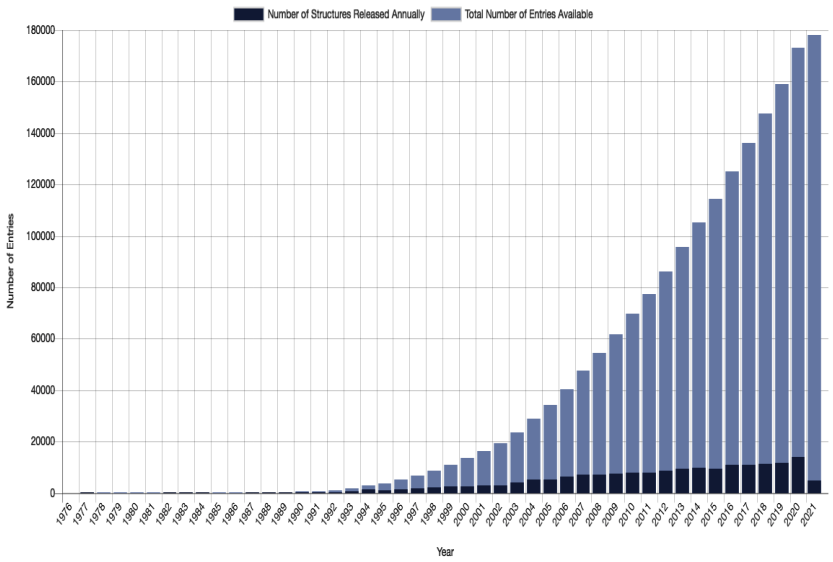


Figure 1: *Growth of the number of structures in PDB data (1976 - 2021). Adapted from* `http://www.rcsb.org/`

Figure 1 displays the growth in the number of structures in the PDB which continues to increase each year. At present (May, 2021), there are 177,910 structures in the PDB archives.

1.3.2 Structural Classification of Proteins - extended (SCOPe)

Nearly all proteins have structural similarities with other proteins and, in many of these cases, share a common evolutionary origin [10]. The Structural Classification of Proteins (SCOP) database aimed to provide a detailed and comprehensive description of the structural and evolutionary relationships between all proteins of known structure [9]. SCOPe data is a database that extends the SCOP data. It classifies proteins based on similarities of their structures and amino acid sequences.
The proteins in this database are classified hierarchically with a number of levels. The fundamental unit of classification is a domain in the experimentally determined protein structure [10].

The classification of the proteins is as follows:
Species: proteins that have a distinct protein sequence that is naturally occurring or artificially created variants [10].

Protein: grouping together similar sequences of essentially the same functions that either originate from different biological species or represent different isoforms within the same species [10].
Family: categorizing proteins with similar sequences but different functions.
Super-family: bridging together protein families with common functional and structural features inferred to be from a common evolutionary ancestor [10].
Fold: superfamilies that have similar structures are grouped together.
Class: Different folds are grouped into different classes basing on their secondary structure content and organisation.
The hierarchical classification of the proteins is shown in Figure 2.

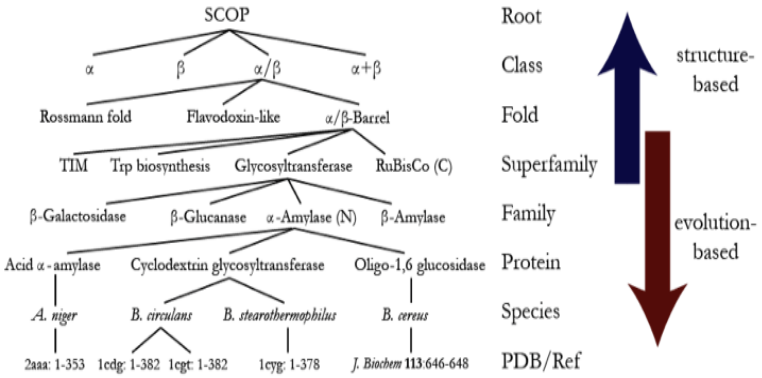


Figure 2: *SCOP hierarchical classification of proteins.*
Adapted from `https://scop.berkeley.edu/help/ver=2.07`

Crystal structure of selenomethionine substituted isoflavanone 4'-O-methyltransferase is an example of an annotated sequence with protein id, '1ZHF' and below is how it is classified according to SCOPe.
Class, a: All alpha proteins
Fold, a.4: DNA/RNA-binding 3-helical bundle
Superfamily, a.4.5: "Winged helix" DNA-binding domain
Family, a.4.5.0: automated matches

The SCOPe data, version 2.07 which was used for this project can be accessed freely via the link `https://scop.berkeley.edu/downloads/ver=2.07`.

2 METHODOLOGY

To be able to assess the performance of BLAST, BLAST results with modified parameters were analysed. However, before the parameters of BLAST were modified, a query set was prepared. The flow chart in Figure 3 provides an overview of the methodology.

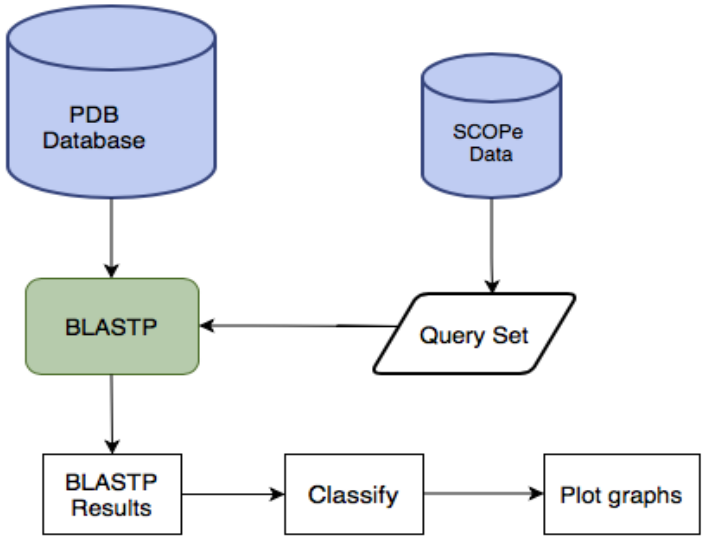


Figure 3: *Overview of the methodology*

2.1 Query Set Preparation

A query set was generated from the SCOPe data. The SCOPe data used had 276,231 rows with 87,225 unique protein ids. To get rid of sequences with more than one chain and protein tags, proteins sequences/protein ids with only one full chain were selected. This generated a subset with 2,158 unique protein ids. 100 sequence ids of different SCOPe families were randomly selected from the subset generated. A set of these sequences formed the query set that was used for the analysis.

2.2 Parameters Modified

Like any other algorithm, BLAST too has parameters which were modified to assess the variation in performance. BLAST was run using the query set generated in Section 2.1 and the PDB database as the target sequence while adjusting the parameters.
The parameters modified were:

2.2.1 Expect Value

Expect value (e-value) is the number of matches you expect to be found by chance. The lower the e-value, the better the alignment. E-value is defined as:

$$E = Kmne^{-\lambda S}$$

where, K and λ are parameters, m is the query sequence length, n is the target sequence (database) length and S is the score. This implies that increasing either of sequences in-

creases e-value, nevertheless e-value decreases exponentially as score increases. The default expect threshold for BLASTP is 10. This implies that alignments that score an e-value greater than 10 are not returned. Increasing the default threshold increases the number of alignments generated.
The rule of thumb for different values of e-value is given below:

- e-value < 10e-100 : Identical sequences.

- 10e-100 < e-value < 10e-50: Almost identical sequences.

- 10e-50 < e-value < 10e-10: Closely related sequences.

- 10e-10 < e-value < 1: Could be a true homologue.

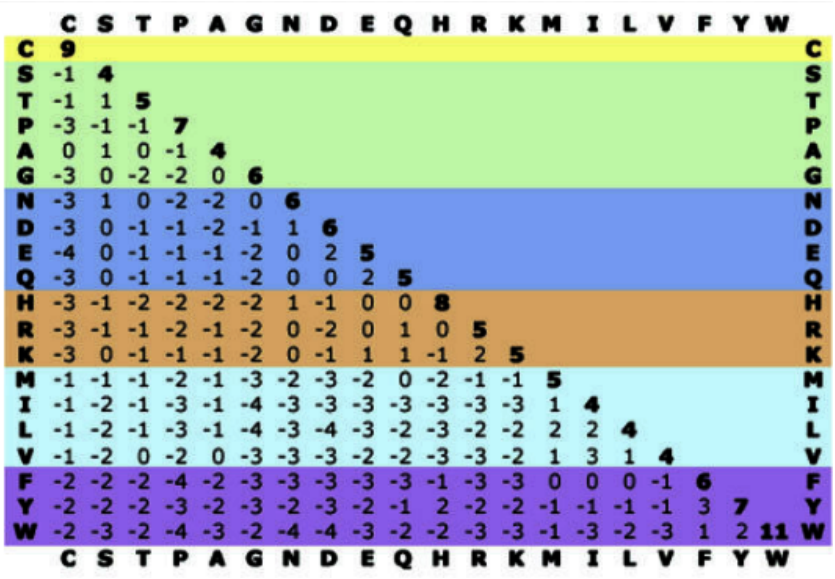
- e-value > 1: Proteins are most likely not related

2.2.2 K-mers (Word Size)

K-mers refers to all possible sub sequences of length k in a query sequence. For example, given a sequence ABCD, all the possible 2-mers are : AB, BC, CD. BLAST assumes if two sequences are similar, then they share a word in common. BLAST first breaks the query sequences into short sequences that are of length k, it then searches and locates all the k-mers in the large database. Only the short sequences found in the target are used to build the alignment and assigned a score. Also neighborhood words for each k-mers is generated with a score above the predefined threshold. BLAST then extends these short sequences in both directions. The default word size for BLASTP is W = 3 (3-mers). Increasing the word size reduces the number of seeds and the number of hits because it requires longer continuous matches. Reducing the word size produces more hits since it requires short sub sequences to be matched.

2.2.3 Substitution Matrices

Scoring matrices or substitution matrix is a collection of scores for aligning nucleotides or amino acids with one another [17]. The scoring matrix can affect the algorithm's ability to identify distantly related sequences. BLASTP assigns a similarity score to each pair of the aligned sequences [11]. The commonest substitution matrices used are: Point Accepted Mutation (PAM) [15] and Blocks Substitution Matrices (BLOSUM) [16]. Different PAM or BLOSUM like: PAM250, PAM30, PAM70 BLOSUM45, BLOSUM80 matrices can be used depending on the target sequences whether most similar sequences or distant sequences. The number attached to the matrix is the evolutionary distance. Low values of PAM target aligments that are higly similar while high PAM values target weaker alignments. On the other hand, BLOSUM high values target highly similar alignments while low BLOSUM values are bettter for distantly related sequences. The default substitution matrix for



	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W	
C	9																				C
S	-1	4																			S
T	-1	1	5																		T
P	-3	-1	-1	7																	P
A	0	1	0	-1	4																A
G	-3	0	-2	-2	0	6															G
N	-3	1	0	-2	-2	0	6														N
D	-3	0	-1	-1	-2	-1	1	6													D
E	-4	0	-1	-1	-1	-2	0	2	5												E
Q	-3	0	-1	-1	-1	-2	0	0	2	5											Q
H	-3	-1	-2	-2	-2	-2	1	-1	0	0	8										H
R	-3	-1	-1	-2	-1	-2	0	-2	0	1	0	5									R
K	-3	0	-1	-1	-1	-2	0	-1	1	1	-1	2	5								K
M	-1	-1	-1	-2	-1	-3	-2	-3	-2	0	-2	-1	-1	5							M
I	-1	-2	-1	-3	-1	-4	-3	-3	-3	-3	-3	-3	-3	1	4						I
L	-1	-2	-1	-3	-1	-4	-3	-4	-3	-2	-3	-2	-2	2	2	4					L
V	-1	-2	0	-2	0	-3	-3	-3	-2	-2	-3	-3	-2	1	3	1	4				V
F	-2	-2	-2	-4	-2	-3	-3	-3	-3	-3	-1	-3	-3	0	0	0	-1	6			F
Y	-2	-2	-2	-3	-2	-3	-2	-3	-2	-1	2	-2	-2	-1	-1	-1	-1	3	7		Y
W	-2	-3	-2	-4	-3	-2	-4	-4	-3	-2	-2	-3	-3	-1	-3	-2	-3	1	2	11	W
	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W	

Figure 4: BLOSUM62 scoring matrix

BLASTP is BLOSUM62 which target alignments with 20 – 30% identity [12]. Figure 4 shows the BLOSUM62 scoring matrix. The matrix is symmetric and therefore only the lower entries are shown.

2.2.4 Gap Penalties

Once BLAST locates the short similar sequences in the target sequence, it extends it in both directions. However, if it can not be extended, BLAST joins it with a gap in order to generate a proper alignment. The penalty for the creation of a gap should be large enough that gaps are introduced only where needed, and the penalty for extending a gap should take into account the likelihood that insertions and deletions occur over several residues at a time [18]. The default setting for gap penalties in BLASTP is 11 for opening a new gap and 1 for extending an already existing gap.

2.2.5 Integration of Low Complexity Regions

Low-complexity regions (LCRs) in protein sequences are regions containing little diversity in their amino acid composition [19]. A major problem with low-complexity sequences arises in sequence homology searches. Because of the repetitive nature of these sequences, one might detect many high-scoring similarities that are biologically meaningless [20]. To be able improve sequence similarity searching, BLASTP uses SEG [21] to filter out regions of low complexity in the query sequence. While SEG is quite effective in determining low-complexity segments, it is sensitive to the choice of parameters. SEG uses four parameters,

including the window size and the low (trigger) complexity [20].

2.3 Classification of BLAST Results

For each query sequence, BLAST was run against the PDB database while modifying the parameters and the results from BLAST search were classified accordingly. BLAST returns a number of sequences that it finds similar from the PDB database to the query sequence. BLAST results were classified as either True Positives (TP), False Positives (FP), True Negatives (TN) or False Negatives (FN) with respect to family, super-family, fold and class as per the classifications of the SCOPe data. Figure 5 displays how each sequence returned by BLAST for every query sequence was classified.

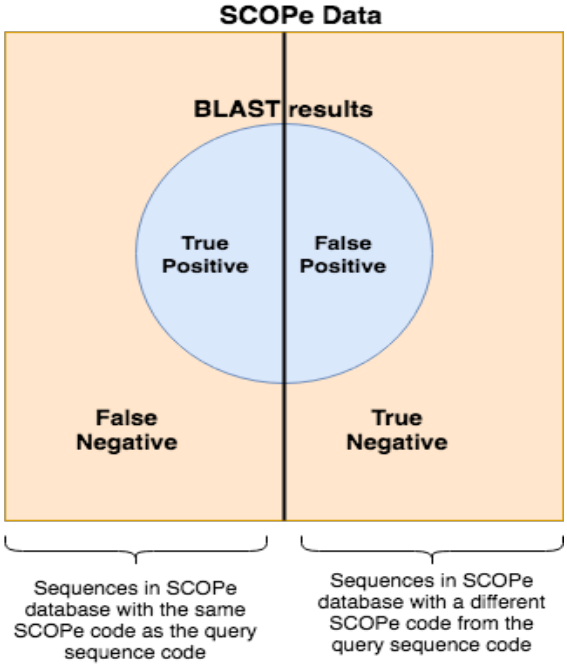


Figure 5: *Classification of BLAST results*

Given these four outcomes, a 2 × 2 confusion matrix can be generated and this is shown in Table 2 for each of the query sequence:

Table 2: *A 2 × 2 confusion matrix*

		Predicted		Total
		Positive	Negative	
Actual	Positive	TP	FN	P
	Negative	FP	TN	N

2.3.1 Positives

For each query sequence, all sequences returned by BLAST as similar to the query sequence were classified as positives which were further categorised as either True Positives or False Positives.

True Positive (TP)

A sequence returned by BLAST for a certain query was considered a true positive under a given SCOPe classification if it's SCOPe code was the same as the the SCOPe code of the query sequence regarding the same classification group. For example, given a query sequence with protein id '2GFS', the corresponding SCOPe family code is 'd.144.1.7', 'd.144.1' is the corresponding SCOPe super-family code, 'd.144' is the corresponding SCOPe fold code and 'd' is the class of the protein. A sequence returned by BLAST was considered a true positive of the query sequence '2GFS' if it's SCOPe family code is 'd.144.1.7' with regards to family annotation, 'd.144.1' with respect to the super-family annotation, 'd.144' with respect to the fold annotation and 'd' as per the class group. However, if a sequence is a TP in the family group, it is certainly a TP regarding super-family annotation, fold annotation and the class annotation.

False Positive (FP)

A sequence returned by BLAST for a certain query sequence was considered a false positive if it's SCOPe classification code is not equal to the SCOPe classification code of the query sequence under the same classification annotation. With regards to the SCOPe family, a sequence was classified as FP if it's SCOPe family code is not the same as the query SCOPe family code. Under the super-family annotation, this sequence was considered a false positive if the SCOPe super-family code is not equal to the SCOPe super-family of the query sequence. As for the fold annotation, a sequence was regarded as FP if it's SCOPe fold code is not the same as the query SCOPe fold code. Concerning the class classification, it was regarded as FP if it's SCOPe class code is different from the query SCOPe class code.
For example, with sequence '2GFS' as a query sequence, BLAST returns a number of sequences from the PDB database that it finds similar to '2GFS'. Suppose one of sequences returned by BLAST is '1IH0' with SCOPe family code 'a.39.1.5', this sequence would be considered as a FP, in all the SCOPe classifications for query sequence '2GFS'. However considering a sequence, '4ANS' with SCOPe family code 'd.144.1.0' would be classified as a FP under the family annotation but a TP under the super-family, fold and class annotations. If a sequence is a FP under the class annotation, it is certainly a FP in all the other classifications.
Sequence alignments returned by BLAST that are not annotated, that is to say, the sequence

does not exist in the SCOPe database were not considered for further analysis.

2.3.2 Negatives

Negatives considered were all the sequences in the SCOPe database that were not returned by BLAST for each given query sequence. Negatives were also further categorised as True Negatives or False Negatives with regards to all the four SCOPe classifications.

True Negative (TN)

TN were generated as sequences not returned by BLAST and have a different SCOPe code from the query sequence SCOPe code with respect to family, super-family, fold and class for each query sequence.

False Negative (FN)

A sequence in SCOPe but not returned by BLAST for a given query sequence was considered FN if the sequence's SCOPe code is the same as the SCOPe code of the query sequences following the family, superfamily, fold and class annotations.

2.4 Receiver Operating Characteristic (ROC)

ROC graph is a 2-dimensional graph for visualising and assessing the performance of an algorithm or classifier. It is generated by plotting the True Positive Rate (TPR) on the y-axis and the False positive rate (FPR) on the x-axis for all possible thresholds. An ROC graph depicts relative trade-offs between benefits (true positives) and costs (false positives) using different thresholds [13].
For each threshold used, a confusion matrix as shown in Table 2 is generated.

TPR which is also known as the recall or sensitivity is the proportion of the positives that are correctly classified as positives. This is the percentage of all sequences whose SCOPe code is equal to the query sequence SCOPe code that BLAST returns. In other words, of all the sequences that have the same SCOPe code as the query sequence, what percentage is returned by BLAST.
TPR can be calculated as:

$$TPR = \frac{TP}{P} = \frac{TP}{(TP + FN)}$$

FPR is the proportion of the negatives that are incorrectly classified as the positives. This is the percentage of all sequences whose SCOPe code is not equal to the query sequence SCOPe code but are returned by BLAST. In other words, of all sequences that have a different SCOPe code from the query sequence, what percentage is returned by BLAST.

These sequences are classified incorrectly by BLAST.
FPR can be calculated as:

$$FPR = \frac{FP}{N} = \frac{FP}{(FP + TN)}$$

FPR can also be calculated as 1 - Specificity, where specificity is the proportion of negatives that are correctly classified.

2.5 Precision-Recall Curve

Precision is the proportion of true positives of the total true positives and false positives. Precision describes how good an algorithm is at classifying the positive sequences.
Precision can be calculated as:

$$Precision = \frac{TP}{(TP + FP)}$$

Recall is the proportion of the true positives of the total true positives and negatives. Recall is the same as sensitivity as described in Section 2.4

Precision recall curve is a 2 dimensional plot of precision on the y-axis and recall on the x-axis for different thresholds.

2.6 Area Under Curve (AUC)

The AUC measures the entire area under the ROC or Precision-Recall curve. It summarizes the overall performance of an algorithm over all possible thresholds. An ideal ROC curve will hug the top left corner, so the larger the AUC, the better the algorithm [14]. A perfect algorithm has AUC = 1

3 RESULTS

In this section, a comparison of the results while adjusting the parameters is presented. The technique used for adjusting the parameters was the One Variable At Time (OVAT). OVAT is varying one variable while keeping other parameters at default. For a given classification level, and a given class of parameters with BLAST, the total TP was calculated as the sum of all the TP from each query sequence. Total FP was the total sum of all the FP from each query sequence. For each query sequence, the FN were added up to get the total FN and also the TN were summed to get the total TN.
For example, suppose under the family classification level with default parameters, each sequence in the query set obtains BLAST results which can be classified as TP, FP, TN and FN. The total TP under this case is the sum of all the TP in each query sequence, FP is the sum of all FP generated in all the query sequence, total FN is the total of all FN from all the query sequences and finally the total TN is the summation of all the TN from all the query sequences.

3.1 Summary Statistics

The length of the sequences in the query set is shown in Figure 6 (left) and it shows that most of the sequences have between 100 and 400 amino acids. 1FTQ is the longest sequence with 842 amino acids while 1QG9 has the least amino acids, 21. The distribution of the number of sequences returned by BLAST for each query sequence using default parameters is shown in a histogram in Figure 6 (right) which suggests different sets of sequences where the most of the query sequences had less than 250 sequences returned by BLAST and a few extreme sequences. Figure 7 shows the number of sequences returned by BLAST for each sequence in the query set using BLAST default parameters. Two sequences; 2GFS (2251) and 4HVI (2253) had the highest number of sequences returned.

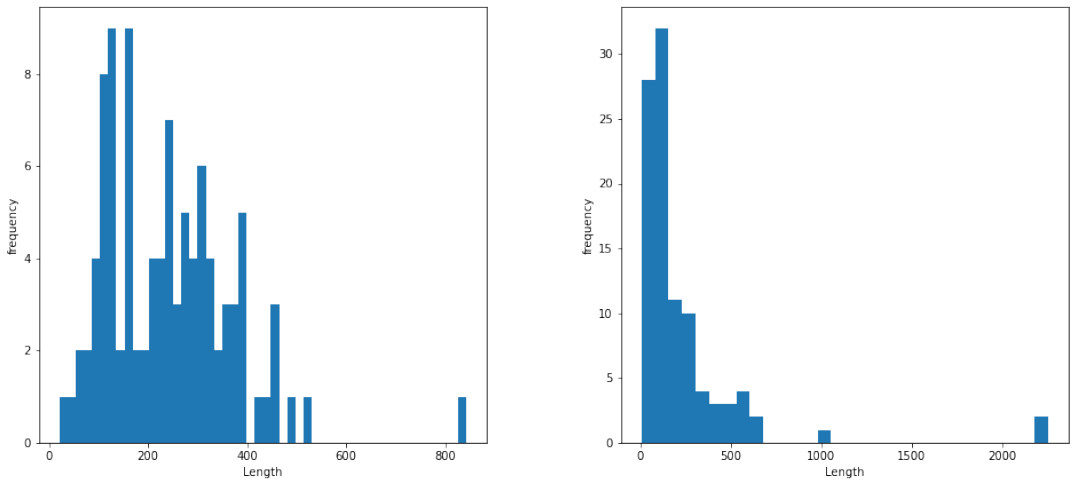


Figure 6: *Histogram of the length of the sequences (left) and the number of the sequences returned by BLAST(right)*

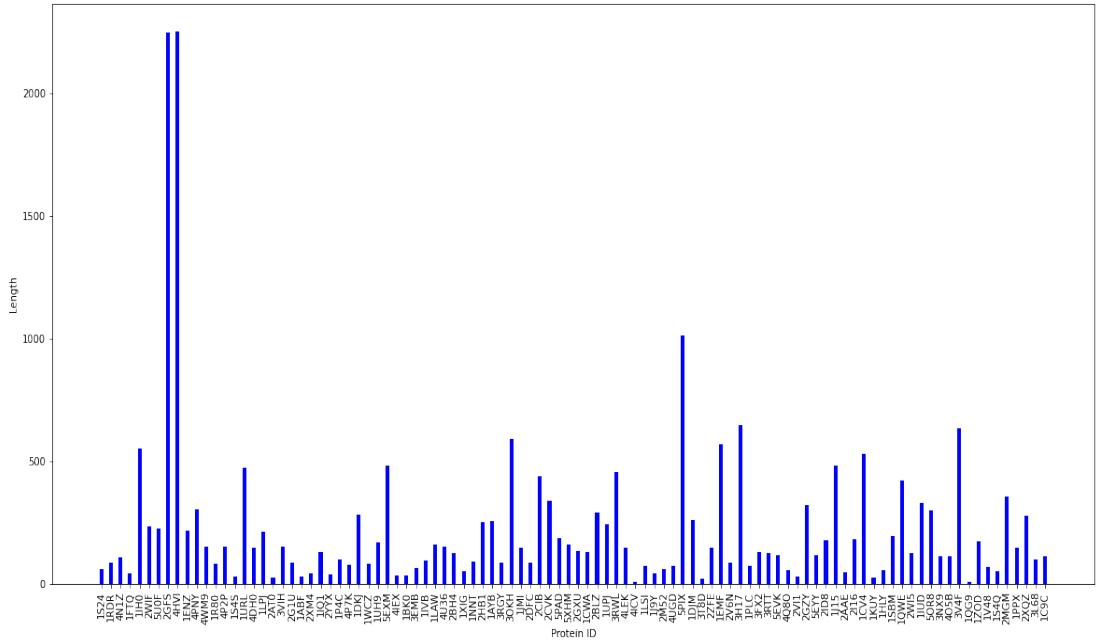


Figure 7: *Bar plot of the query sequences with number of sequences returned by BLAST*

3.2 Family

The analysis at the family level is presented in this subsection.

3.2.1 E-Value

The e-value was adjusted to different values, e-value = 5, e-value = 10, e-value = 1,000, e-value = 10,000 and e-value =1,000,000. E-value = 5, 10, 1,000, 10,000 display the same ROC curve as shown in Figure 8. This is because one is a subset of the other that is to say, for e-value = 5, BLAST returns sequence with e-value $\leq$ 5, e-value = 10, BLAST returns sequences with e-value $\leq$ 10. This implies sequences returned by BLAST when e-value = 5 are the same sequences returned by BLAST with a few more with e-value $>$ 5 but less than 10. Therefore, increasing e-value increases the number of hits which gains more TP but at the same time generate more FP. Precision-recall curve on the other hand shows all e-values have the same precision-recall curve.

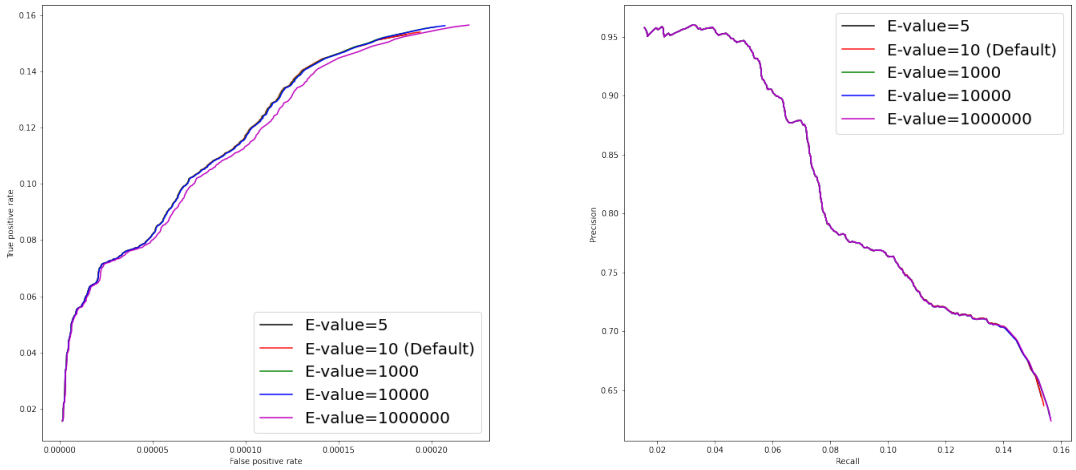


Figure 8: *ROC curve (left) and Precision-recall curve (right) for different e-values at the family level.*

Table 3 shows the different metrics for varying e-value and it shows that increasing the e-value increases the AUC for both the ROC and the precision curve since increasing e-value increases the hits too.

Table 3: *Different metrics for varying e-value*

E-value	FPR	TPR	Recall	Precision	AUC(ROC)	AUC (PR)
5	0.000186	0.153100	0.153100	0.644746	0.0000205	0.1126765
10	0.000194	0.154007	0.154007	0.636957	0.0000217	0.1132590
1,000	0.000202	0.155852	0.155852	0.629817	0.0000230	0.1144353
10,000	0.000207	0.156291	0.156291	0.625947	0.0000237	0.1147117
1,000,000	0.000220	0.156502	0.156502	0.623853	0.0000253	0.1148576

3.2.2 K-mers (Word Size)

BLAST allows for adjusting the word size with 2, 3, 4, 5, 6 and 7. Starting with short sequences of size either 2, 3, 4, 5, 6 or 7 yields quite the same results with changing thresholds. All word sizes increase the the true positive rate, false positive rate as they reduce precision as shown in Figure 9 with a slight reduction of true positives with W=3 after threshold = 0.000185.

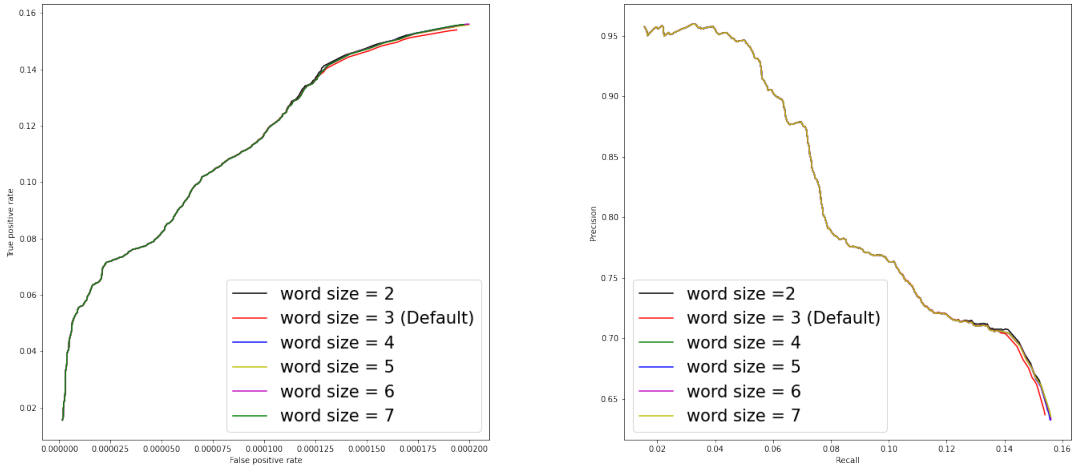


Figure 9: *ROC curve (left) and Precision-recall curve (right) for different word size values at the family level.*

Table 4 shows the different metrics of the performance of BLAST using different word sizes at optimum threshold. All metrics for the different word sizes are approximately the same with very small differences. AUC of the ROC curve and other metrics keep increasing until word size of length, 7 where it reduces again. This implies that using word size = 7 yields poor performance.

Table 4: *Different metrics for word size for maximum threshold*

Word size	FPR	TPR	Recall	Precision	AUC(ROC)	AUC (PR)
2	0.000192	0.154971	0.154971	0.640576	0.0000215	0.1139959
3	0.000194	0.154007	0.154007	0.636957	0.0000217	0.1132590
4	0.000199	0.156020	0.156020	0.634256	0.0000225	0.1146248
5	0.000200	0.155806	0.155806	0.632584	0.0000226	0.1144736
6	0.000200	0.156086	0.156086	0.633000	0.0000226	0.1146603
7	0.000197	0.155905	0.155905	0.635657	0.0000223	0.1145480

3.2.3 Substitution Matrices

The substitution matrices used were BLOSUM45 and BLOSUM90 besides BLOSUM62. BLOSUM45 targets distantly related proteins and BLOSUM90 targets highly similar proteins. Figure 10 shows that modifying the substitution matrix results into quite similar results. They all perform equally the same. This may be as a result that BLOSUM45 is for distantly related proteins and therefore closely related proteins are not reported. Considering BLOSUM90, it is for more related proteins and therefore distantly related proteins are not reported. Some alignments are not returned by BLAST hence losing a number of TP in both cases and this leads to quite similar metrics.

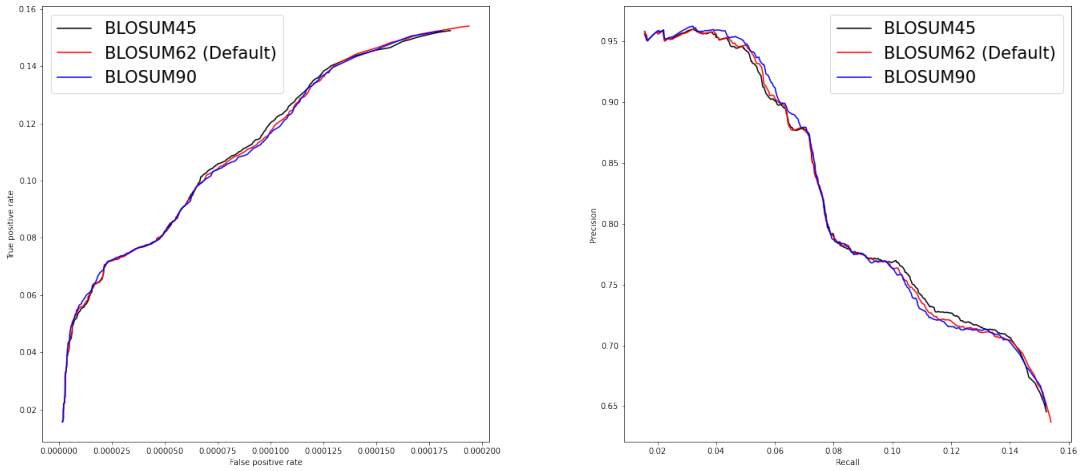


Figure 10: *ROC curve (left) and Precision-recall curve (right) for different scoring matrices.*

Table 5 shows the different metrics at optimal threshold and it shows that BLOSUM62 performs slightly better than BLOSUM45 and BLOSUM90 since it's AUC = 0.0000217 is

the greatest of them all and the AUC of the precision-recall curve.

Table 5: *Different metrics for scoring matrices at family level*

Scoring matrix	FPR	TPR	Recall	Precision	AUC(ROC)	AUC (PR)
BLOSUM45	0.000185	0.152440	0.152440	0.645282	0.0000204	0.1122344
BLOSUM62	0.000194	0.154007	0.154007	0.636957	0.0000217	0.1132590
BLOSUM90	0.000181	0.152175	0.152175	0.649684	0.0000197	0.1121721

3.2.4 Gap Penalties

There are quite a number of options to open a new gap and extend an already existing gap while using BLAST. BLAST allows the following gap penalties: (6, 2), (7, 2), (8, 2), (9, 1), (9, 2), (10, 1), (10, 2), (11, 1), (11, 2), (12, 1), (13, 1) and (32767, 32767) for (gap opening, gap extension). Figure 11 suggests that these gap penalties modify the results of BLAST quite the same except for one gap penalty with gap opening = 32767 and gap extension = 32767. This deviates much further from all the other gap penalties which can be seen in both the ROC curve and the precision-recall curve in Figure 11. Since the gap penalty for creating and extending a gap is very high, this results into very few significant alignments hence a very a low positive rate. Gap penalties (6, 2), (7, 2), (8, 2), (9, 1), (10, 1) increase the true positive rate much faster than the default.

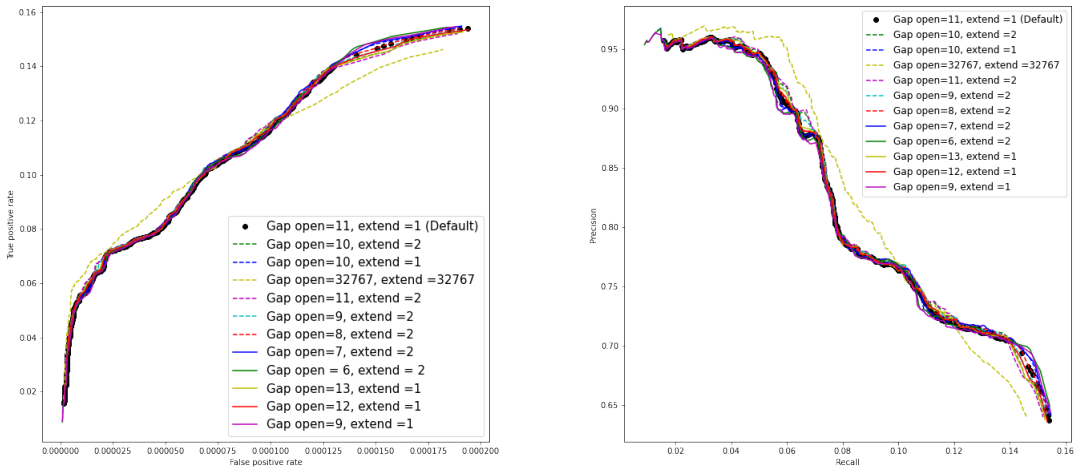


Figure 11: *ROC curve (left) and Precision-recall curve (right) for different gap penalties.*

Gap penalty with gap opening = 12 and gap extension = 1 performs slightly better than

the default gap penalty as displayed in Table 6 by comparing their AUC for the ROC curve and precision-recall curve.

Table 6: *Different metrics for gap penalties for optimum threshold at family level*

Opening	Extend	FPR	TPR	Recall	Precision	AUC (ROC)	AUC (PR)
6	2	0.000184	0.154352	0.154352	0.649019	0.0000203	0.1199286
7	2	0.000191	0.154831	0.154831	0.641490	0.0000214	0.1143908
8	2	0.000190	0.153990	0.153990	0.641380	0.0000212	0.1134075
9	1	0.000189	0.154596	0.154596	0.644178	0.0000208	0.1195106
9	2	0.000191	0.153232	0.153232	0.639720	0.0000213	0.1111300
10	1	0.000191	0.154451	0.154451	0.640534	0.0000213	0.1169126
10	2	0.000190	0.152457	0.152457	0.638994	0.0000212	0.1092184
11	1	0.000194	0.154007	0.154007	0.636957	0.0000217	0.1132590
11	2	0.000189	0.152013	0.152013	0.639646	0.0000210	0.1092626
12	1	0.000194	0.153546	0.153546	0.635442	0.0000218	0.1121785
13	1	0.000191	0.152618	0.152618	0.637662	0.0000213	0.1116719
32767	32767	0.000182	0.146203	0.146203	0.639238	0.0000198	0.1078841

3.2.5 Integration of Low Complexity Regions

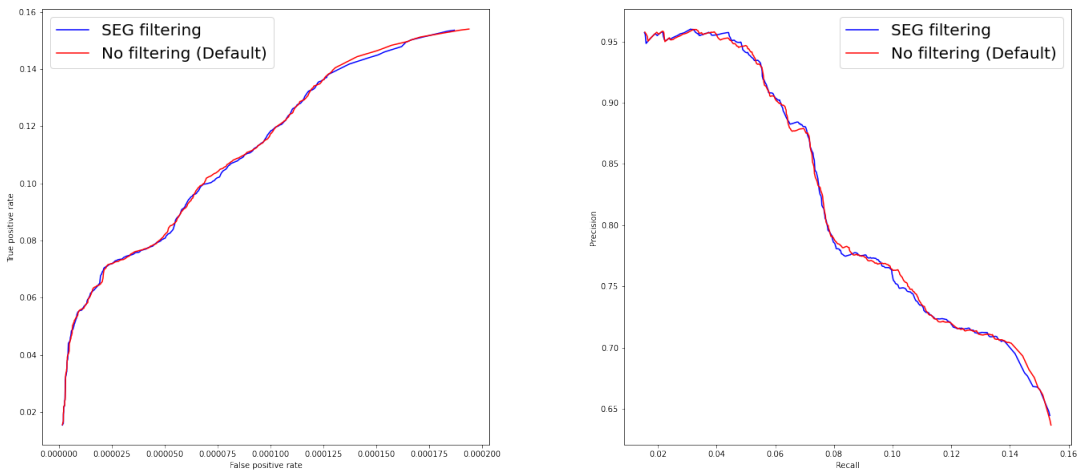


Figure 12: *ROC curve (left) and Precision-recall curve (right) for filtering out LCR and no filtering.*

Figure 12 displays the ROC curve and the precision-recall curves of the performance of BLAST while filtering out the LCR in contrast to not filtering them out. The difference in the results when LCR's are filtered and when not filtered is marginal. This may be because most of our sequences in the query set did not have LCR's and therefore no regions were filtered when SEG was applied. However, the results in Table 7 show the metrics of the two instances are approximately the same with the filtering LCR generating less values since SEG eliminates the LCR which would result into meaningless similarities.

Table 7: *Different metrics for filtering LCR for maximum threshold*

Filtering LCR	FPR	TPR	Recall	Precision	AUC (ROC)	AUC (PR)
Yes	0.000187	0.153596	0.153596	0.644725	0.0000205	0.1130313
No	0.000194	0.154007	0.154007	0.636957	0.0000217	0.1132590

3.3 Super-family

Analysis was also done on the super-family level of SCOPe data and the results are presented in this section.

3.3.1 E-Value

Figure 13 shows that as e-value increases both the true positive rate and false positive increase. However at the start, false positive rate and precision are constant. This is due to the fact that BLAST generates more hits when using a high e-value. However, some of these hits may be TP which contributes to the increase in true positive rate and some are FP which contribute to the false positive rate. Conversely, the precision also reduces further with high values of e-value.

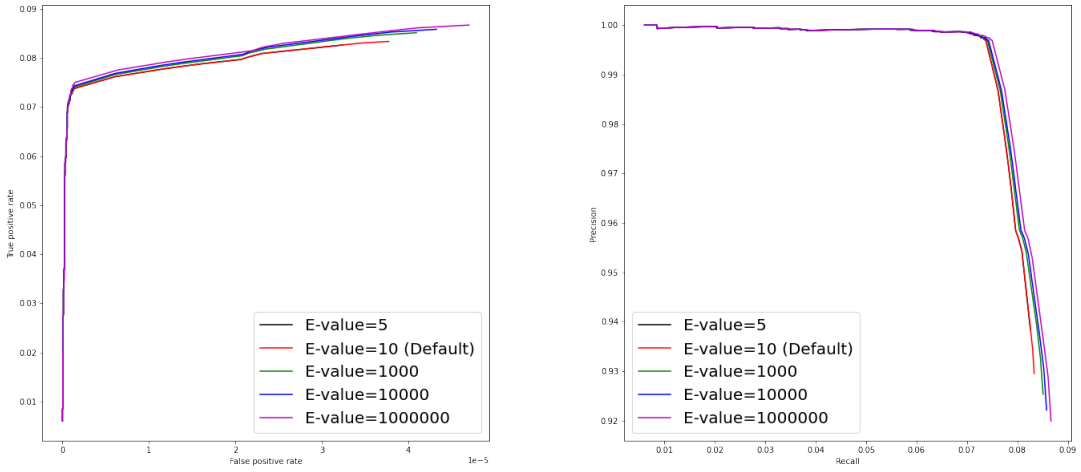


Figure 13: *ROC curve (left) and Precision-recall curve (right) for filtering out LCR and no filtering.*

The AUC in Table 8 increases as e-value increases. This means that BLAST performs better with higher e-value, however it is at the cost of generating more false positives. As more TP are generated, more FP are also generated.

Table 8: *Different metrics for varying e-value on super-family level*

E-value	FPR	TPR	Recall	Precision	AUC(ROC)	AUC (PR)
5	0.000032	0.082613	0.082613	0.938391	0.0000025	0.0762634
10	0.000038	0.083335	0.083335	0.929567	0.0000030	0.0769384
1,000	0.000041	0.085097	0.085097	0.925392	0.0000033	0.0786142
10,000	0.000043	0.085795	0.085795	0.922160	0.0000035	0.0792658
1,000,000	0.000047	0.086663	0.086663	0.919945	0.0000038	0.0800691

3.3.2 K-mers (Word Size)

Figure 14 shows the ROC curve and precision recall curve while using different word sizes at the super-family level. These graphs suggest that initially the all word sizes generate no FP until a threshold = 1.06e-147 where they all start generating FP similarly except for W=3 and W=2 that generates less FP than the other word sizes. This implies that increasing the word size increases the true positive rate or recall but committing more false positives for the ROC (left). On the other hand, the precision for larger word sizes is higher than the precision for small word sizes with the same true positive rate.

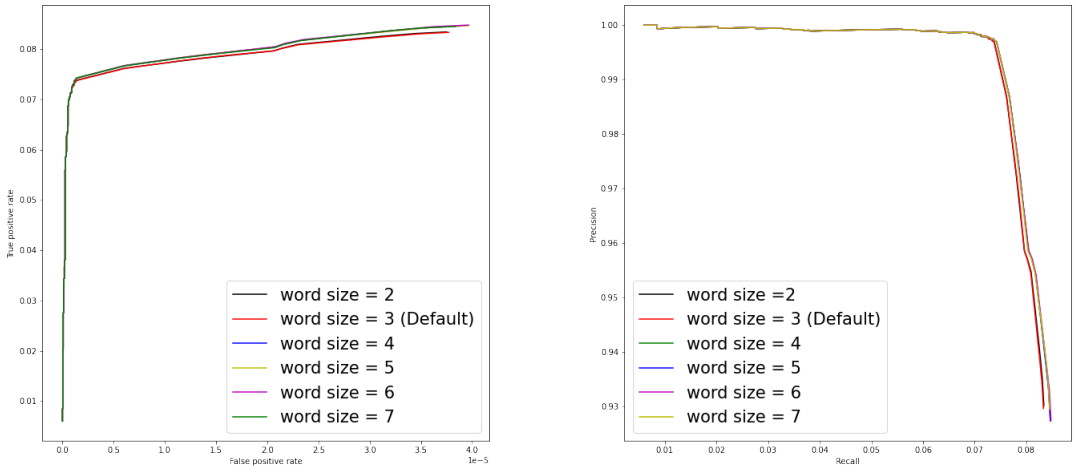


Figure 14: *ROC curve (left) and Precision-recall curve (right) for different word size values based on super-family.*

Word size = 5 and word size = 6 have better performance as compared to other word sizes with the greatest AUC as shown in Table 14, however they still have the highest false positive rate.

Table 9: *Different metrics for different word sizes for maximum threshold at super-family*

Word size	FPR	TPR	Recall	Precision	AUC(ROC)	AUC(PR)
2	0.000038	0.083418	0.083418	0.930018	0.0000030	0.0770186
3	0.000038	0.083335	0.083335	0.929567	0.0000030	0.0769384
4	0.000039	0.084634	0.084634	0.927894	0.0000031	0.0781986
5	0.000040	0.084682	0.084682	0.927251	0.0000032	0.0782451
6	0.000040	0.084792	0.084792	0.927400	0.0000032	0.0783496
7	0.000038	0.084523	0.084523	0.929424	0.0000031	0.0780945

3.3.3 Substitution Matrices

Substitution matrices at the super-family are more or less the same until threshold = 0.011 when BLOSUM62 has higher true positive and higher precision than BLOSUM45 and BLOSUM90.

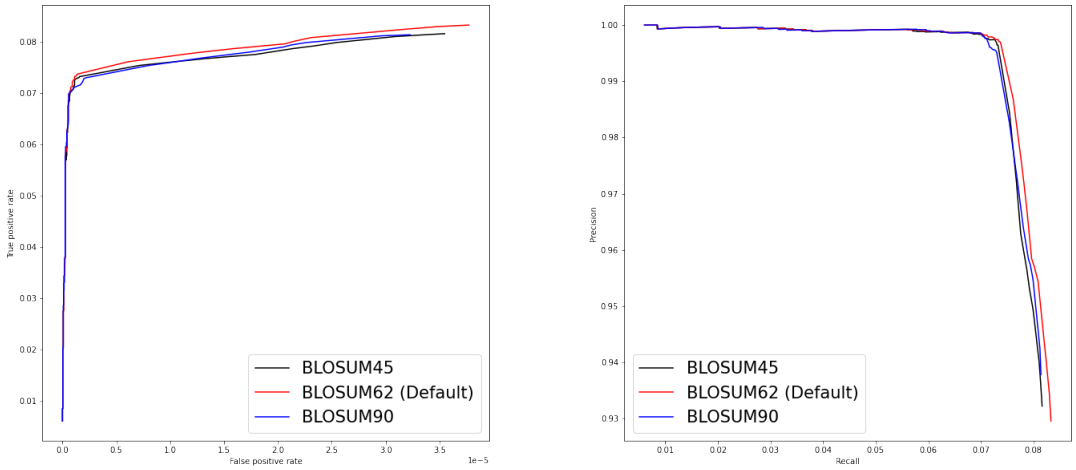


Figure 15: *ROC curve (left) and Precision-recall curve (right) for different substitution matrices based on super-family.*

The metrics in Table 15 show that BLOSUM62 performs better than BLOSUM45 and BLOSUM90 according to the AUC of the curves.

Table 10: *Different metrics for different scoring matrices at super-family*

Scoring Matrix	FPR	TPR	Recall	Precision	AUC (ROC)	AUC(PR)
BLOSUM45	0.000035	0.081650	0.081650	0.932229	0.0000027	0.0752364
BLOSUM62	0.000038	0.083335	0.083335	0.929567	0.0000030	0.0769384
BLOSUM90	0.000032	0.081434	0.081434	0.937825	0.0000025	0.0750716

3.3.4 Gap Penalties

Adjusting for gap penalties shows that the less penalty on gap opening increases the true positive rate and reduces precision much higher than the default while a higher penalty on gap opening results into less true positive and reduces the precision much less as compared to the default gap penalty. Since the penalty for opening a gap is small, it results into high alignment scores hence more hits. The metrics in Table 11 show that gap penalty (6, 2) performs the worst as compared to the rest of the gap penalties because the less the penalty, the more the alignments which may even be meaningless.

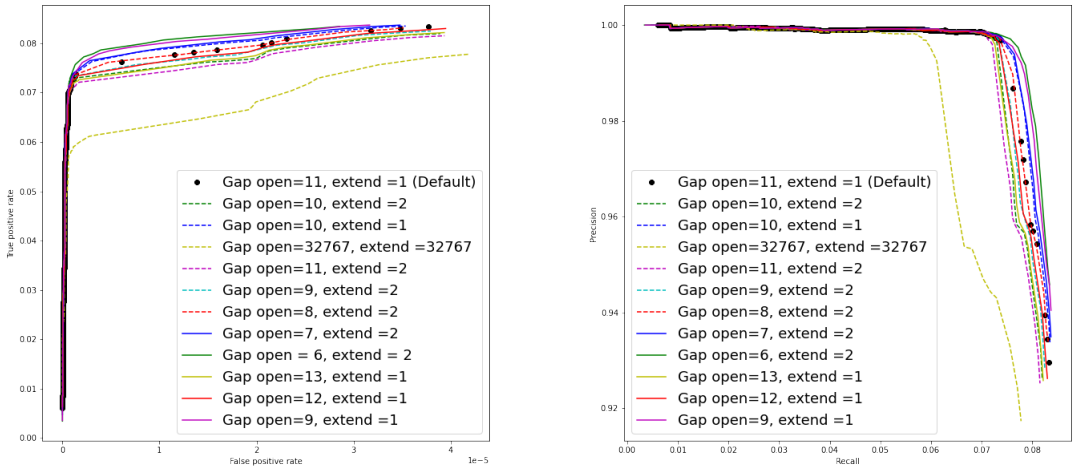


Figure 16: *ROC curve (left) and Precision-recall curve (right) for different gap penalties based on super-family.*

Table 11: *Different metrics for gap penalties for super-family*

Opening	Extend	FPR	TPR	Recall	Precision	AUC (ROC)	AUC (PR)
6	2	0.000029	0.083404	0.083404	0.945811	0.0000023	0.0797779
7	2	0.000035	0.083682	0.083682	0.934959	0.0000028	0.0775262
8	2	0.000035	0.083144	0.083144	0.933931	0.0000028	0.0767478
9	1	0.000032	0.083691	0.083691	0.940479	0.0000025	0.0798868
9	2	0.000038	0.082465	0.082465	0.928493	0.0000030	0.0752420
10	1	0.000035	0.083489	0.083489	0.933806	0.0000028	0.0785703
10	2	0.000039	0.081949	0.081949	0.926352	0.0000030	0.0741285
11	1	0.000038	0.083335	0.083335	0.929567	0.0000030	0.0769384
11	2	0.000039	0.081523	0.081523	0.925232	0.0000030	0.0738146
12	1	0.000039	0.082985	0.082985	0.926259	0.0000030	0.0761846
13	1	0.000039	0.082158	0.082158	0.925829	0.0000030	0.0753821
32767	32767	0.000042	0.077782	0.077782	0.917355	0.0000030	0.0703041

3.3.5 Integration of Low Complexity Regions

Figure 17 suggests that filtering LCR and not filtering modify the BLAST results the same way until threshold = 0.011 where the results when filtering show a slightly less rate in true positive and precision.

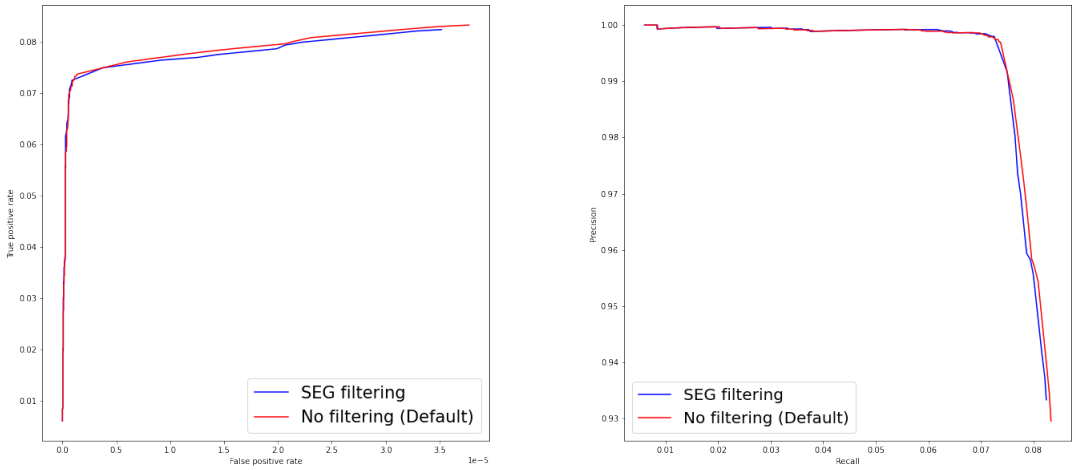


Figure 17: *ROC curve (left) and Precision-recall curve (right) while filtering out LCR and no filtering.*

Table 12 shows that filtering produces lower metric values since BLAST ignores the LCR's that would cause excess FP.

Table 12: *Different metrics for LCR for super-family*

LCR	FPR	TPR	Recall	Precision	AUC (ROC)	AUC(PR)
Filtering	0.000035	0.082446	0.082446	0.933347	0.0000027	0.0761233
No filtering	0.000038	0.083335	0.083335	0.929567	0.0000030	0.0769384

3.4 Fold

BLAST performance was also assessed on the SCOPe fold level and below are the results.

3.4.1 E-Value

Increasing e-value increases the number of alignments returned by BLAST. Figure 18 shows that higher values of e-value result into higher values of the true positive rate , higher values of false positive rate and reduces precision as compared to the small values of e-value. This means that TP are generated more but also FP are committed more as you increase e-value. Table 13 displays higher AUC values for higher values of e-value which would imply better performance level, however these values also generate a lot of FP.

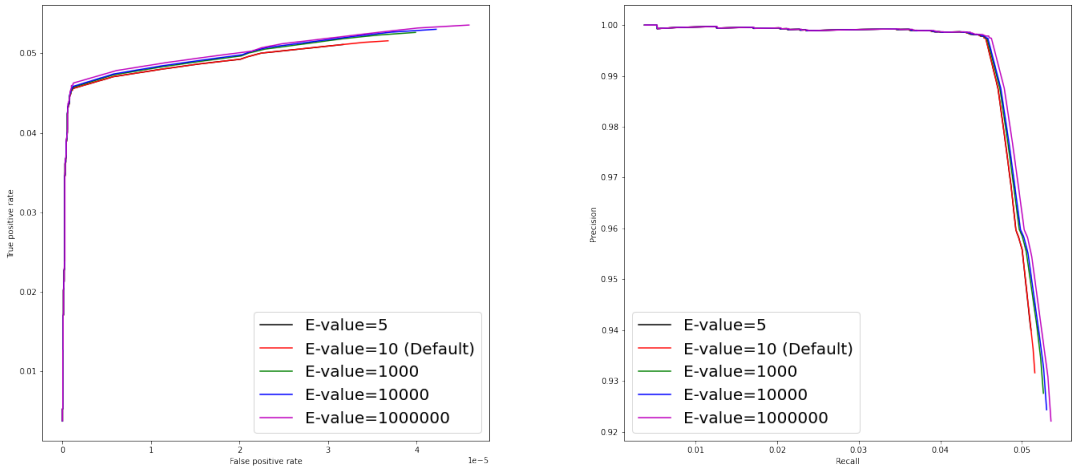


Figure 18: *ROC curve (left) and Precision-recall curve (right) for varying e-values based on fold level.*

Table 13: *Different metrics for varying e-value on fold level*

E-value	FPR	TPR	Recall	Precision	AUC(ROC)	AUC (PR)
5	0.000032	0.051110	0.051110	0.939983	0.0000015	0.0471915
10	0.000037	0.051580	0.051580	0.931606	0.0000018	0.0476319
1,000	0.000040	0.052632	0.052632	0.927584	0.0000020	0.0486354
10,000	0.000042	0.053026	0.053026	0.924333	0.0000021	0.0490040
1,000,000	0.000046	0.053561	0.053561	0.922109	0.0000023	0.0494992

3.4.2 K-mers

The ROC and precision-recall curves in Figure 19 suggest that larger word size values generate higher values of true positive rate as compared to small word sizes. They also reduce precision moderately more than the small word size. The AUC of the ROC curve as displayed in Table 14 increases with increase in word size, however increasing further than 6 drops the AUC hence poor performance of BLAST.

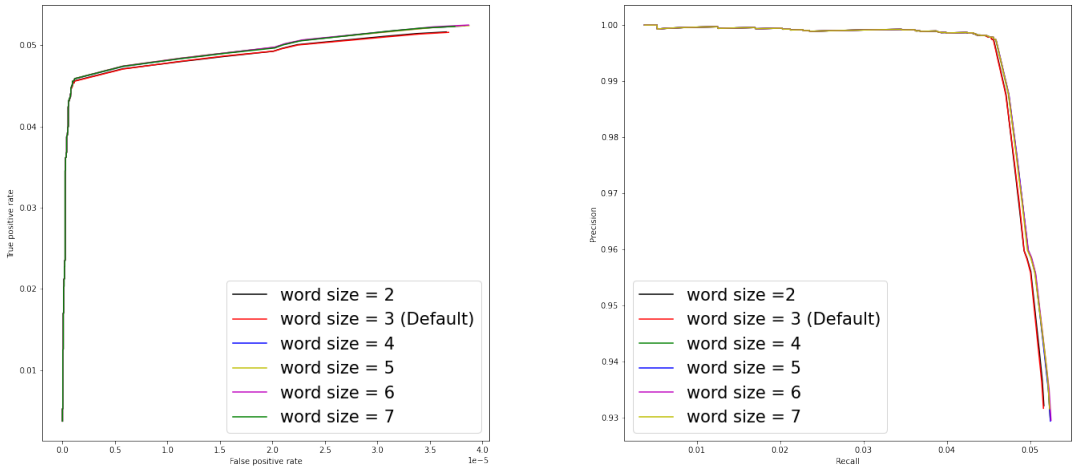


Figure 19: *ROC curve (left) and Precision-recall curve (right) for different word size values based on fold.*

Table 14: *Different metrics for varying word sizes based on fold*

Word size	FPR	TPR	Recall	Precision	AUC (ROC)	AUC(PR)
2	0.000037	0.051633	0.051633	0.932056	0.0000018	0.0476832
3	0.000037	0.051580	0.051580	0.931606	0.0000018	0.0476319
4	0.000038	0.052384	0.052384	0.929965	0.0000019	0.0484129
5	0.000039	0.052414	0.052414	0.929320	0.0000019	0.0484416
6	0.000039	0.052482	0.052482	0.929467	0.0000019	0.0485063
7	0.000037	0.052316	0.052316	0.931501	0.0000018	0.0483488

3.4.3 Substitution Matrices

BLOSUM45 and BLOSUM90 perform quite similar, however BLOSUM62 has a better performance level than BLOSUM45 and BLOSUM90 as displayed in both ROC and precision-recall curves in Figure 20. Further more, the metrics in Table 15 also show that BLOSUM62 has a better overall performance with a higher AUC portrayed in both the ROC and precision-recall.

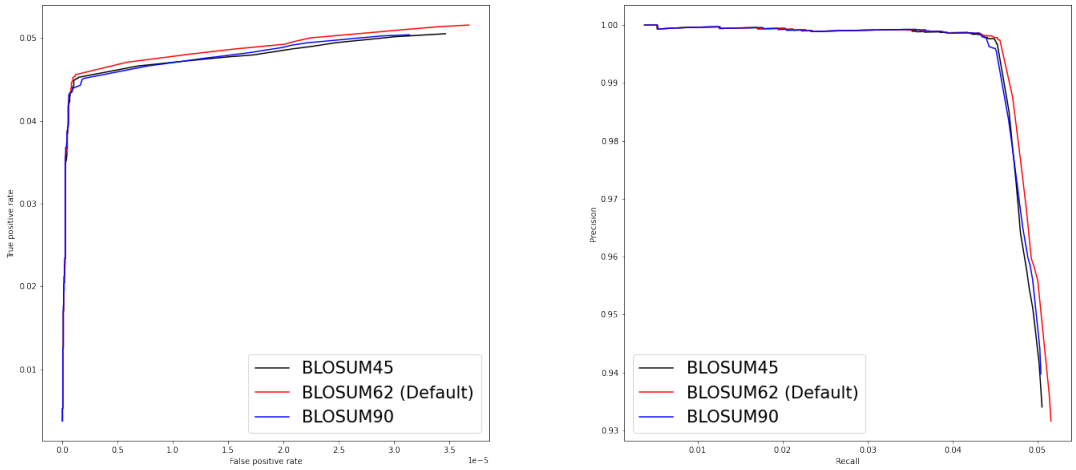


Figure 20: *ROC curve (left) and Precision-recall curve (right) for different substitution matrices based on fold.*

Table 15: *Different metrics for different substitution matrices based on fold*

Scoring matrix	FPR	TPR	Recall	Precision	AUC (ROC)	AUC(PR)
BLOSUM45	0.000035	0.050526	0.050526	0.934038	0.0000017	0.0465684
BLOSUM62	0.000037	0.051580	0.051580	0.931606	0.0000018	0.0476319
BLOSUM90	0.000031	0.050398	0.050398	0.939719	0.0000015	0.0464710

3.4.4 Gap Penalties

Lower values of gap penalty perform better than the higher values of penalty with less false positive rate and high true positive rate as shown in Figure 21. Gap penalty (32767, 32767) performs much worse than all the others gap penalties. Precision on the other hand also shows that gap penalties with less penalty on opening have a better performance level. This is because they result into more hits hence an increase in TP.

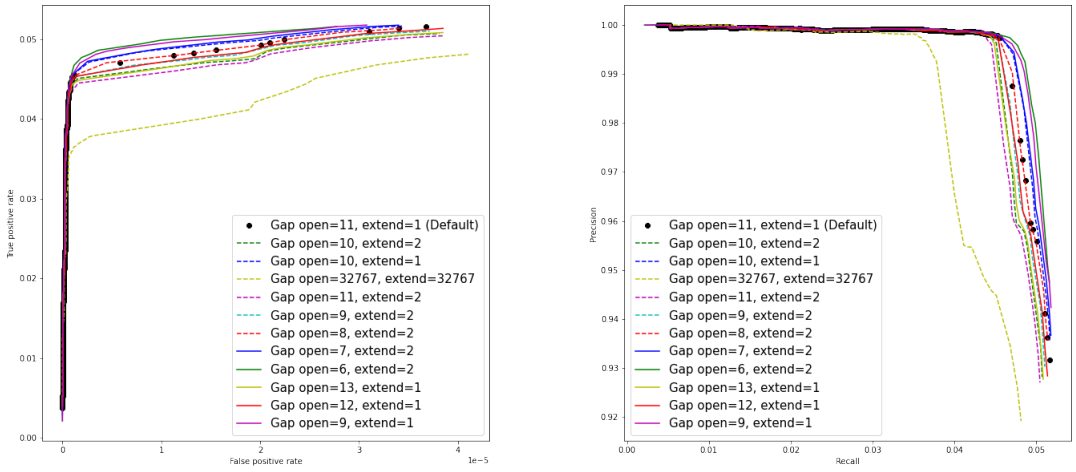


Figure 21: *ROC curve (left) and Precision-recall curve (right) for different gap penalties based on fold.*

Table 16: *Different metrics for gap penalties for fold group*

Opening	Extend	FPR	TPR	Recall	Precision	AUC (ROC)	AUC (PR)
6	2	0.000028	0.051603	0.051603	0.947539	0.0000014	0.0493669
7	2	0.000034	0.051772	0.051772	0.936661	0.0000017	0.0479743
8	2	0.000034	0.051438	0.051438	0.935574	0.0000017	0.0474914
9	1	0.000031	0.051788	0.051788	0.942329	0.0000015	0.0494419
9	2	0.000037	0.051027	0.051027	0.930277	0.0000018	0.0465691
10	1	0.000035	0.051663	0.051663	0.935647	0.0000017	0.0486290
10	2	0.000038	0.050709	0.050709	0.928143	0.0000018	0.0458823
11	1	0.000037	0.051580	0.051580	0.931606	0.0000018	0.0476319
11	2	0.000039	0.050448	0.050448	0.927030	0.0000018	0.0456897
12	1	0.000038	0.051368	0.051368	0.928367	0.0000019	0.0471708
13	1	0.000038	0.050843	0.050843	0.927684	0.0000018	0.0466617
32767	32767	0.000041	0.048136	0.048136	0.919152	0.0000017	0.0435271

Table 16 displays the various metrics of gap penalties at the optimum threshold and it shows that gap penalty (6, 2) produces the worst results with the least AUC and gap penalty (12, 1) performs the best with the highest AUC, however it is not best when regarding the AUC of the precision recall curve.

3.4.5 Integration of Low Complexity Regions

Like the classifications before, that is to say family, super-family, Figure 22 of both the ROC and the precision recall curves show that filtering the LCR's generates less positive rate as compared to running BLAST with LCR's included. The reason is that LCR's causes misaligments which are reported as significant yet they are actually insignificant. This is further displayed in Table 17 which also indicates low metric values of filtering.

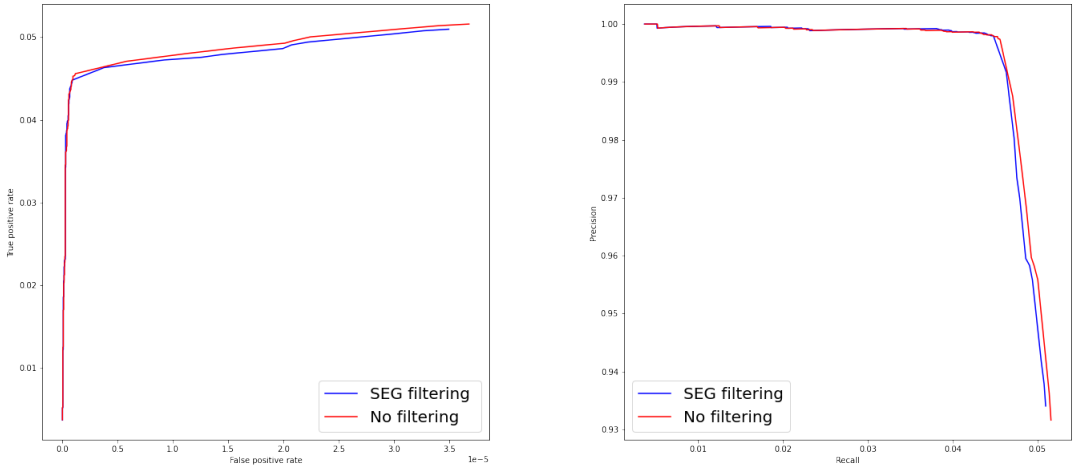


Figure 22: *ROC curve (left) and Precision-recall curve (right) for LCR based on fold level.*

Table 17: *Different metrics for LCR for fold level*

LCR	FPR	TPR	Recall	Precision	AUC (ROC)	AUC(PR)
Filtering	0.000035	0.050955	0.050955	0.934037	0.0000017	0.0470490
No filtering	0.000037	0.051580	0.051580	0.931606	0.0000018	0.0476319

3.5 Class

In regards to the class classification level, the analysis is presented below.

3.5.1 E-Value

Higher e-values modify BLAST results with more TP and FP. At the start, the ROC and precision-recall curve is the same for all the e-values until threshold = 1.06e-147 where higher values of e-values deviate from the lower values of e-value as displayed in Figure 23. This is because higher values of e-value lead to more sequences (hits) returned by BLAST

which can be classified as TP or FP. Table 18 shows increasing values of AUC for both the ROC curve and the precision-recall curve as the e-values increase. The higher the AUC, the better the performance however such big values of e-values also generate high values of FP since the false positive rate in both Figure 23 and Table 18 show a rising trend.

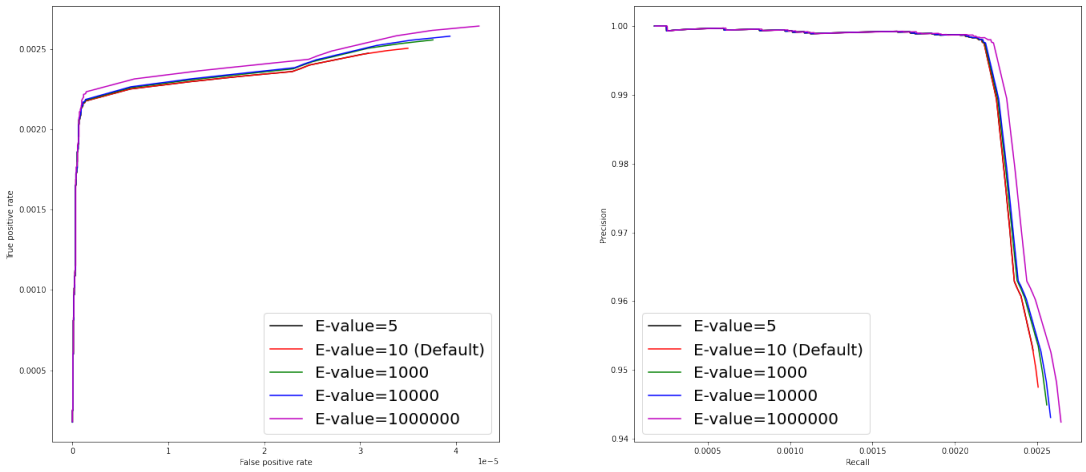


Figure 23: *ROC curve (left) and Precision-recall curve (right) for varying e-values based on class level.*

Table 18: *Different metrics for varying e-value on class level*

E-value	FPR	TPR	Recall	Precision	AUC(ROC)	AUC (PR)
5	0.000031	0.002474	0.002474	0.952859	0.000000071	0.002286511
10	0.000035	0.002505	0.002505	0.947515	0.000000081	0.002315892
1,000	0.000038	0.002557	0.002557	0.944924	0.000000088	0.002366115
10,000	0.000039	0.002580	0.002580	0.943102	0.000000093	0.002387899
1,000,000	0.000042	0.002643	0.002643	0.942434	0.000000100	0.002446726

3.5.2 K-mers

Larger word sizes increase true positive rate and false positive slightly much more than the word size = 2 and word size = 3 as shown in Figure 24. Precision also decreases more when using lower word size values as shown in Figure 24.

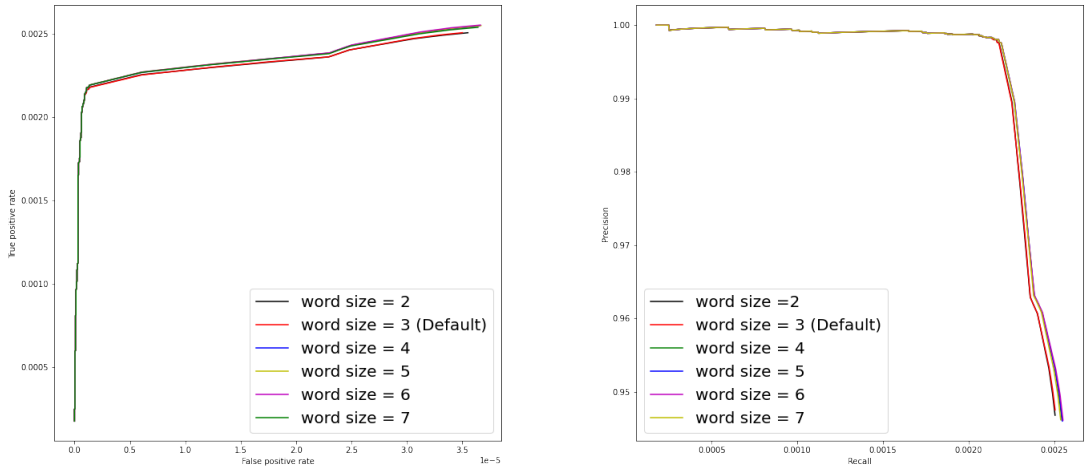


Figure 24: *ROC curve (left) and Precision-recall curve (right) for different word sizes based on class level.*

Word size = 3 generates the least false positive rate, a true positive rate ≈ word size = 2 and the highest precision. This means it predicts the positives much better than the other word sizes. Unfortunately, it has the least AUC for the ROC as displayed in Table 19.

Table 19: *Different word size value metrics on the class level*

Word size	FPR	TPR	Recall	Precision	AUC (ROC)	AUC(PR)
2	0.000036	0.002504	0.002504	0.946800	0.000000082	0.002315499
3	0.000035	0.002505	0.002505	0.947515	0.000000081	0.002315892
4	0.000037	0.002544	0.002544	0.946137	0.000000085	0.002354631
5	0.000037	0.002547	0.002547	0.946005	0.000000086	0.002357409
6	0.000037	0.002551	0.002551	0.946133	0.000000086	0.002360518
7	0.000036	0.002537	0.002537	0.946180	0.000000085	0.002347515

3.5.3 Substitution Matrices

BLOSUM62 generates more true positive rate as compared to BLOSUM45 and BLOSUM90. There is a marginal difference between the curves of BLOSUM45 and BLOSUM90 as shown in Figure 25 however BLOSUM62, BLOSUM90 generate more false positives as they gain more true positives. The precision on the other hand decreases much faster for BLOSUM45 and BLOSUM90 than BLOSUM62.

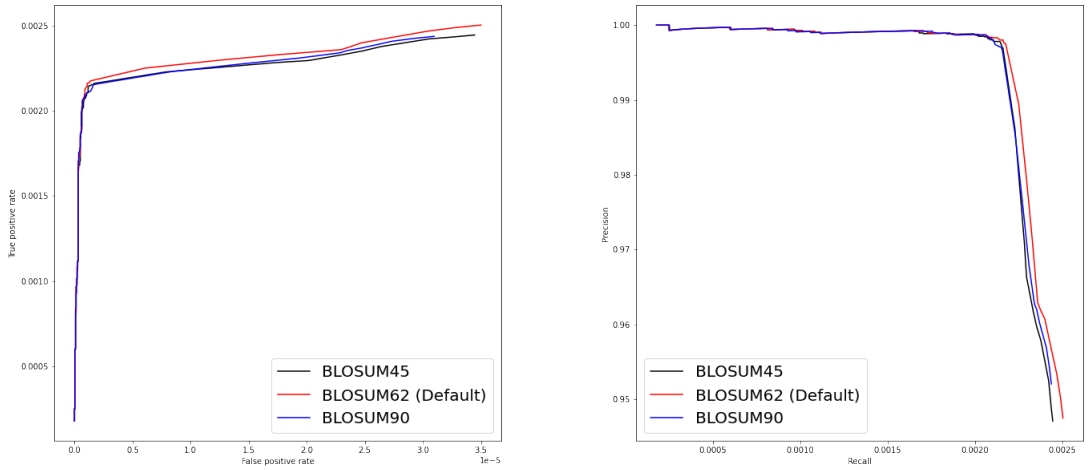


Figure 25: *ROC curve (left) and Precision-recall curve (right) for different substitution matrices based on class level.*

The metrics in Table 20 show that BLOSUM62 performs much better than BLOSUM45 and BLOSUM90 in accordance to the AUC of both the ROC and the precision-recall curves.

Table 20: *Different scoring matrix metrics on the class level*

Scoring matrix	FPR	TPR	Recall	Precision	AUC (ROC)	AUC(PR)
BLOSUM45	0.000034	0.002446	0.002446	0.947119	0.000000078	0.002257007
BLOSUM62	0.000035	0.002505	0.002505	0.947515	0.000000081	0.002315892
BLOSUM90	0.000031	0.002438	0.002438	0.952070	0.000000070	0.002250480

3.5.4 Gap Penalties

Figure 26 shows that a less penalty on gap opening increases the true positive rate and false positive much higher than a higher penalty on gap opening. The precision also decreases faster for the less penalty on gap openings than higher penalty on gap opening. The metrics in Table 21 show that the gap penalty (6, 2) performs the worst according the AUC of the ROC, however according to the AUC of the precision-recall curve, gap penalty (32767, 32767) performs the worst. This is because the gap penalty (32767, 32767) produces more false positives which reduce precision since alignments can not be extended because of the high penalty.

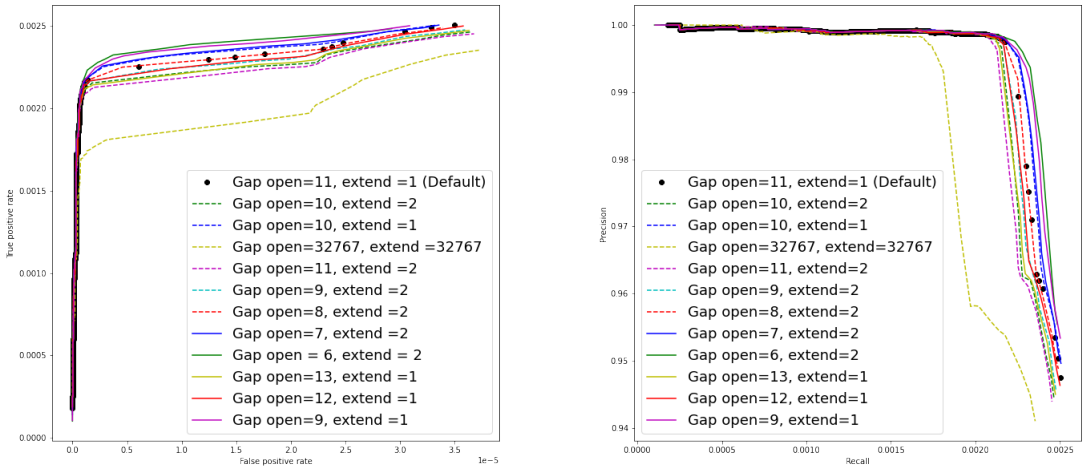


Figure 26: *ROC curve (left) and Precision-recall curve (right) for different gap penalties based on class level.*

Table 21: *Different metrics for gap penalties for class group*

Opening	Extend	FPR	TPR	Recall	Precision	AUC (ROC)	AUC (PR)
6	2	0.000029	0.002484	0.002484	0.955419	0.000000069	0.002377492
7	2	0.000034	0.002506	0.002506	0.949602	0.000000079	0.002324131
8	2	0.000034	0.002491	0.002491	0.948788	0.000000078	0.002301825
9	1	0.000031	0.002501	0.002501	0.953356	0.000000073	0.002389299
9	2	0.000036	0.002476	0.002476	0.945581	0.000000082	0.002263087
10	1	0.000033	0.002505	0.002505	0.950031	0.000000078	0.002359296
10	2	0.000036	0.002464	0.002464	0.944540	0.000000083	0.002233024
11	1	0.000035	0.002505	0.002505	0.947515	0.000000081	0.002315892
11	2	0.000037	0.002453	0.002453	0.943906	0.000000083	0.002225098
12	1	0.000036	0.002500	0.002500	0.946327	0.000000083	0.002299170
13	1	0.000036	0.002473	0.002473	0.944990	0.000000083	0.002272833
32767	32767	0.000037	0.002353	0.002353	0.940999	0.000000075	0.002134233

3.5.5 Integration of Low Complexity Regions

Filtering increases the true positive rate but reducing precision. However, the increase is slightly less as compared to when the low complexity regions are not filtered out as shown in Figure 27. The metrics in Table 22 also shows that filtering out the LCR results in less

metrics.

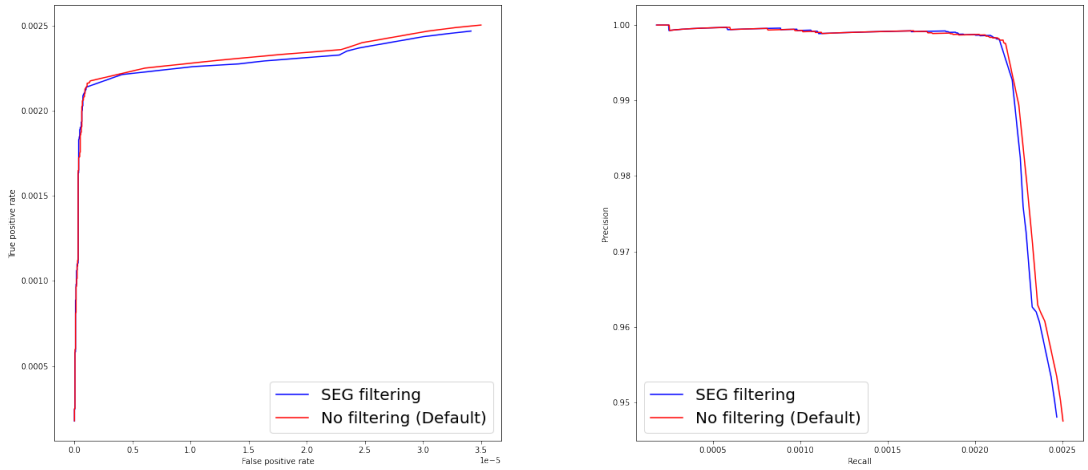


Figure 27: *ROC curve (left) and Precision-recall curve (right) for LCR based on class level.*

Table 22: *Different metrics for LCR for class classification*

LCR	FPR	TPR	Recall	Precision	AUC (ROC)	AUC(PR)
Filtering	0.000034	0.002469	0.002469	0.948044	0.000000078	0.002282623
No filtering	0.000035	0.002505	0.002505	0.947515	0.000000081	0.002315892

Further more, an ROC and precision-recall curve of all the BLAST results with default parameters are shown in Figure 28. The plot suggests family level has the highest true positive rate, false positive rate and the least precision while class level has the least true positive rate and least precision at the maximum threshold. This shows that BLAST retrieves more true positives at the family level and keeps reducing in performance from super-family to class.

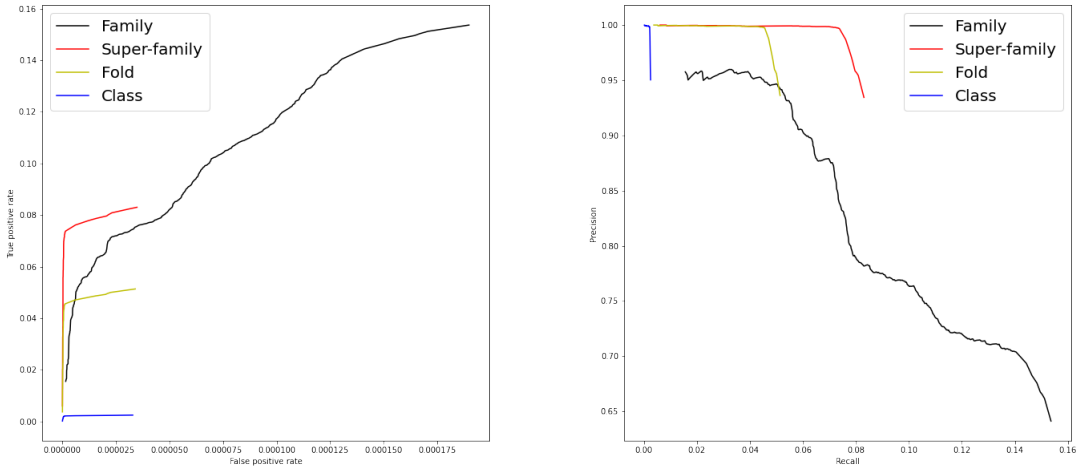


Figure 28: *ROC (left) and Precision-recall curve (right) for BLAST results with default parameters at different SCOPe levels.*

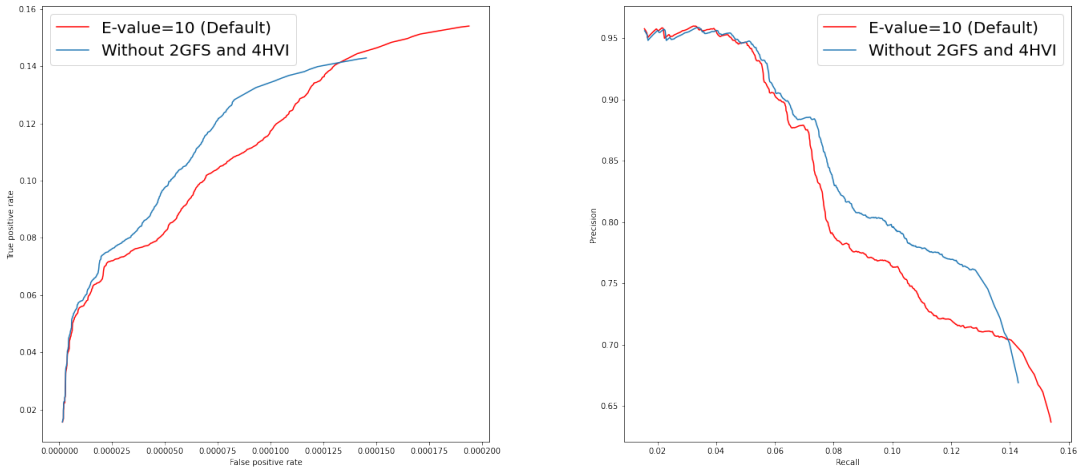


Figure 29: *ROC (left) and Precision-recall (right) of the default parameters with and without the sequences with the highest number of sequences returned*

Figure 7 showed sequences 2GFS and 4HVI had the highest sequences returned by BLAST. An ROC and precision-recall curve with and without the sequences are shown in Figure 29. The plots show that the two sequences have an effect on the true positive rate, false

positive rate and precision. The figure shows the curve without 2GFS and 4HVI has a higher true positive with a low false positive rate. The precision-recall curve shows that the results with 2GFS and 4HVI reduce the precision much faster and further.

4 DISCUSSION

BLAST is a heuristic based algorithm for evaluating sequence similarity searching. However, it performs this analysis based on a short sequence from a query sequence that is common in both the query and the target sequences with a risk of missing relevant homologies. In this project, we assessed the variation of BLAST performance using different values of parameters (e-value, k-mers, scoring matrix, gap penalties, low complexity regions) with respect to the default ones. The evaluation of BLAST was done basing on four levels (classifications) of SCOPe: family, super-family, fold and class.

Considering the family group, the performance of BLAST with the default parameters and the modified parameters were marginal according to the ROC curve and the precision-recall curves. A big change was seen in gap penalties where gap penalty (32767, 32767) curves drift from the default curves. The AUC was approximately the same for all BLAST results ≈ 0.00002 for ROC. Additionally, the AUC of the precision-recall curve was roughly around 0.11 for all the BLAST results, false positive rate was around 0.0002, true positive rate was approximately 0.1500 and precision was about 0.6400.

The super-family analysis did not show numerous difference as well albeit, the metrics were adjusted. The false positive rate reduced to around 0.00004, the true positive rate decreased to about 0.08000 and precision rose to 0.93000. The AUC of the curves at the super-family level for the ROC and precision recall curves were approximately 0.000003 and 0.070000 respectively which implies BLAST performs less better than at the family level.

When looking at the fold group, the performance of BLAST between the modified and the default parameters was more or less the same except for the gap penalty (32767, 32767) which showed further performance from other gap penalties. The false positive rate is about 0.00004, which means no new FP are accumulated that were initially TP in the previous level. The true positive rate drops to about 0.05 and precision at this SCOPe classification was approximately 0.93. Both the ROC curve and the precision-recall curve show a reduction in performance of BLAST at this level since their values reduce to about 0.0000017 for the ROC and 0.05 for the precision-recall curve.

Regarding the class group, the variation in BLAST performance between different parameters was not significant with AUC of about 0.0000001 for the ROC and 0.002 for the precision-recall curve. The performance metrics, true positive rate was about 0.00003, true

positive rate was around 0.002 and precision increased to 0.94.

The false positive rate reduced because of a reduction in FP that are used to compute it at every level since a sequence classified as a FP at a family level can be classified as a TP at super-family, fold level and class level. There seemed to be an increase in the FN from group to group which also leads to reduction in the true positive rate. The increase in TP increase the precision at each classification.

Other studies also show using larger word size improves performance of the program. Longer seeds would be expected to improve sensitivity of BLAST searches [22]. Further more, BLAST with default parameters is not efficient enough for LCR analysis which is due to the fact that it is designed to search for evolutionary relationships between regular protein sequences and significance of matches in the presence of low complexity segments is over-estimated hence meaningless similarities are found significant [23, 20]. When considering whether to change gap penalties to improve search selectivity for a particular protein family, gap penalties should be increased [12].

Although ROC is a good measure for performance evaluation of an algorithm, it is appropriate when the observations between the classes (positives and negatives) are balanced.

As BLAST is based on a heuristic approach, it is widely used because of its speed and flexibility, nevertheless, BLAST's running time has been found to be proportional to the size of the database [24] and since the amount of biological data produced is constantly increasing, this affects BLAST's efficiency. In addition to that, the overall AUC did not indicate BLAST as a good performing algorithm.

5 CONCLUSION

Generally, higher e-values, larger word sizes, less gap penalties, BLOSUM62 and filtering with SEG had the better performance among the results. Higher e-values generate more hits which improves performance. Larger word sizes reduce the number of seeds generated but improves sensitivity and therefore improves the true positive rate. Less gap penalties perform better since they have low penalties and therefore lead to longer alignments. More alignments can be extended with less penalties hence more hits. BLOSUM62 produces sensitive results since it extends the alignments to as far as the non-homologous regions. The metrics keep reducing as you change from the family level to super-family, fold and class. We would recommend further analysis while changing more than one parameter.

6 REFERENCES

[1] M. W. Gonzalez and W. R. Pearson, Nucleic Acids Research, Vol. 38, No. 7, 2177–2189 (2010)

[2] Donkor, Eric & Dayie, Nicholas & Adiku, Theophilus. (2014). Bioinformatics with basic local alignment search tool (BLAST) and fast alignment (FASTA). Journal of Bioinformatics and Sequence Analysis. 6. 1-6. 10.5897/IJBC2013.0086.

[3] Gonzalez MW, Pearson WR. Homologous over-extension: a challenge for iterative similarity searches. Nucleic Acids Res. 2010;38(7):2177-2189. doi:10.1093/nar/gkp1219

[4] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, David J. Lipman, Basic local alignment search tool, Journal of Molecular Biology, Volume 215, Issue 3, 1990, Pages 403-410.

[5] Ye J, McGinnis S, Madden TL. BLAST: improvements for better sequence analysis. Nucleic Acids Res. 2006;34(Web Server issue):W6-W9. doi:10.1093/nar/gkl164

[6] Berman HM, Westbrook J, Feng Z, Gilliland G, Bhat TN, Weissig H, Shindyalov IN, Bourne PE. The Protein Data Bank. Nucleic Acids Res. 2000 Jan 1;28(1):235-42. doi: 10.1093/nar/28.1.235. PMID: 10592235; PMCID: PMC102472.

[7] Berman, Helen. (2008). The Protein Data Bank: A historical perspective. Acta crystallographica. Section A, Foundations of crystallography. 64. 88-95. 10.1107/S0108767307035623.

[8] `http://www.rcsb.org/`

[9] Lo Conte L, Ailey B, Hubbard TJ, Brenner SE, Murzin AG, Chothia C. SCOP: a structural classification of proteins database. Nucleic Acids Res. 2000;28(1):257-259. doi:10.1093/nar/28.1.257

[10] Fox, Naomi & Brenner, Steven & Chandonia, John-Marc. (2013). SCOPe: Structural Classification of Proteins - Extended, integrating SCOP and ASTRAL data and classification of new structures. Nucleic acids research. 42. 10.1093/nar/gkt1240.

[11] Mills LJ, Pearson WR. Adjusting scoring matrices to correct overextended alignments. Bioinformatics. 2013;29(23):3007-3013. doi:10.1093/bioinformatics/btt517

[12] Pearson WR. Selecting the Right Similarity-Scoring Matrix. Curr Protoc Bioinformatics. 2013;43:3.5.1-3.5.9. doi:10.1002/0471250953.bi0305s43

[13] Fawcett, Tom. (2004). ROC Graphs: Notes and Practical Considerations for Researchers. Machine Learning. 31. 1-38.

[14] Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. An Introduction to Statistical Learning : with Applications in R. New York :Springer, 2013.

[15] Dayhoff MO, Schwartz RM, Orcutt BC: A model of evolutionary change in proteins. In: Atlas of Protein Sequence and Structure, vol. 5. Edited by Dayhoff MO. Washington DC: National Biomedical Research Foundation;. 1978, 345-352.

[16] Henikoff S, Henikoff JG: Amino acid substitution matrices from protein blocks. Proc Natl Acad Sci USA. 1992, 89: 10915-10919.

[17] Altschul, Stephen. (2008). Substitution Matrices. 10.1002/9780470015902.a0005265.pub2.

[18] Pertsemlidis, A., Fondon, J.W. Having a BLAST with bioinformatics (and avoiding BLASTphemy). Genome Biol 2, reviews2002.1 (2001). https://doi.org/10.1186/gb-2001-2-10-reviews2002

[19] Coletta A, Pinney JW, Solís DY, Marsh J, Pettifer SR, Attwood TK. Low-complexity regions within protein sequences have position-dependent roles. BMC Syst Biol. 2010;4:43. Published 2010 Apr 13. doi:10.1186/1752-0509-4-43

[20] Sharon, Itai & Birkland, Aaron & Chang, Kuan & El-Yaniv, Ran & Yona, Golan. (2005). Correcting BLAST e-Values for Low-Complexity Segments. Journal of computational biology : a journal of computational molecular cell biology. 12. 980-1003. 10.1089/cmb.2005.12.980.

[21] Wootton, J. C. & Federhen, S. (1993). Statistics of local complexity in aminoacid sequences and sequence databases. Comp. Chem. 17, 149-163

[22] Shiryev SA, Papadopoulos JS, Schäffer AA, Agarwala R. Improved BLAST searches using longer words for protein seeding. Bioinformatics. 2007 Nov 1;23(21):2949-51. doi: 10.1093/bioinformatics/btm479. Epub 2007 Oct 6. PMID: 17921491.

[23] Jarnot, Patryk & Ziemska-Legiecka, Joanna & Grynberg, Marcin & Gruca, Aleksandra. (2020). LCR-BLAST—A New Modification of BLAST to Search for Similar Low Complexity Regions in Protein Sequences. 10.1007/978-3-030-31964-9_16.

[24] Wattanapomprom, Warin & Nupairoj, Natawut & Chongstitvatana, Prabhas. (2002). Improving the Performance of BLAST in a Memory Limited Environment.

7 APPENDIX

7.1 Python Codes

Query set Preparation

```
import pandas as pd
pd.options.mode.chained_assignment = None
import numpy as np
import os
import matplotlib.pyplot as plt
from matplotlib import pyplot

df = pd.read_table("/Users/violet/Downloads/ncbi-blast-2.11.0+/blast/db/SCOPe/
    dir.cla.scope.2.07-stable.txt", delim_whitespace=True, header = None, skiprows = 4,
    names = ["annotation_id", "pdb_id", "domain", "scope_family", "", " "])
df_sub = df[df['pdb_id'].isin(df['pdb_id'].value_counts()
         [df['pdb_id'].value_counts()==1].index)]
df_sub2 = df_sub[df_sub['domain'].map(len) == 2]
n = 100
top_fam = df_sub2['scope_family'].value_counts()[:n].index.tolist()
df_sub2 = df_sub2[df_sub2['scope_family'].isin(top_fam)]
df_sub3 = df_sub2.groupby("scope_family", group_keys=False).apply(lambda group_df:
          group_df.sample(1, random_state=1))
queryset = df_sub3.sample(n = 100, replace = False, random_state = 1)
queryset.to_csv("queryset.csv", header=True, index=False)
```

Running BLAST in bash with default parameters

```
# This was used to run Query set against the PDB while changing the parameters of interest

#!/bin/bash
for input in *.fasta
do
    echo "Processing $input"
    infile=$input
    prot=pdbaa
    output="results_"$infile$".txt"
    blastp -query $infile -db $prot -max_target_seqs 5000 -evalue 10
    -outfmt 6 -out $output
done
```

Summary statistics

```
# Calculating sequence length
path = "/Users/violet/Downloads/ncbi-blast-2.11.0+/blast/db/Queries"
i = 0
nested = []
for file in os.listdir(path):
    if file.endswith(".fasta"):
        i += 1
        row = []
        row.append(file.split("_")[0])
        file_path = os.path.join(path, file)
        f = open(file_path, "r")
        lines = f.readlines()[1:]
        for line in lines:
            length = len(line. rstrip())
            row.append(length)
        nested.append(row)
print(i)

column_names = ["Sequence", "length"]
sequence_length = pd.DataFrame(nested, columns = column_names)
# Plot histogram
Sequence = sequence_length['Sequence'].tolist()
Sequence_length = sequence_length['length'].tolist()
plt.figure()
plt.figure(figsize = (7, 7))
plt.hist(Sequence_length, density=False, bins=50)
plt.ylabel('frequency')
plt.xlabel('Length')
#plt.show()
plt.savefig('Images/histogram_sequence.png')

# Number of sequences returned by BLAST
path = "/Users/violet/Downloads/ncbi-blast-2.11.0+/
        blast/db/e_value/results_evalue_10"
file_length = []
for file in os.listdir(path):
    if file.endswith(".fasta.txt"):
        file_b = []
```

```
        file_path = os.path.join(path, file)
        blast_result = pd.read_table(file_path, delim_whitespace=True,
                     names=["qseqid", "sseqid", "pident", "length",
                     "mismatch", "gapopen", "qstart", "qend", "sstart",
                     "send", "evalue", "bitscore"])
        blast_df = blast_result.drop_duplicates(subset='sseqid',
                   inplace=False).reset_index(drop=True)
        query_id = blast_df.iloc[0,0].split("|")[0].split("_")[0]
        file_b.append(query_id)
        file_b.append(len(blast_df))
        file_length.append(file_b)
column_names = ["Pdb_id", "length"]
query_length = pd.DataFrame(file_length, columns = column_names)

# Bar plot
id = query_length['Pdb_id'].tolist()
length = query_length['length'].tolist()

fig = plt.figure(figsize = (20, 10))
plt.xticks(rotation=90)
plt.bar(id, length, color ='blue',
        width = 0.4)

plt.xlabel("Protein ID")
plt.ylabel("Length")
plt.savefig('Images/barplot.png')

# Histogram
plt.figure()
plt.figure(figsize = (7, 7))
plt.hist(length, density=False, bins=30)
plt.ylabel('frequency')
plt.xlabel('Length')
plt.savefig('Images/histogram.png')
```

Classifying BLAST results

```
subset = df[['annotation_id', 'pdb_id', 'domain', 'scope_family']]
def pdb_chain(pdb_id, domain):
    return pdb_id.upper() + '_' + domain[0].upper()
```

```
subset['pdb_chain'] = subset.apply(lambda row: pdb_chain(row['pdb_id'],
                      row['domain']), axis=1)
queries = pd.read_csv("/Users/violet/Documents/UHasselt/Sem4/Master_Thesis/
          queryset.csv", usecols = ['pdb_id','domain', 'scope_family'])
def unique_scope_family(scope_family_in):
    return ' '.join(set(scope_family_in.split(' ')))
```

Positives

```
def assigning(path):
    directory = os.listdir(path)
    for file in directory:
        if file.endswith(".fasta.txt"):
            print(file, end = ", ")
            file_path = os.path.join(path, file)
            if os.stat(file_path).st_size != 0:
                blast_result = pd.read_table(file_path, delim_whitespace=True,
                               names=["qseqid", "sseqid", "pident", "length", "mismatch",
                               "gapopen", "qstart", "qend", "sstart", "send",
                               "evalue", "bitscore"])
                blast_df = blast_result.drop_duplicates(subset='sseqid',
                           inplace=False).reset_index(drop=True)
                df_merge_tmp = pd.merge(blast_df, subset, left_on='sseqid',
                               right_on='pdb_chain', how = "left").fillna('-')
                df_groupby = df_merge_tmp.fillna('-').groupby('sseqid',
                             as_index=False).agg({'scope_family': ' '.join})
                df_groupby['scope_family'] = df_groupby.apply(lambda row:
                                  unique_scope_family(row['scope_family']), axis=1)
                df_merge = pd.merge(blast_df, df_groupby, on = "sseqid",
                           how = "left")
                blast_df_merge = df_merge[df_merge['scope_family']
                                 != '-'].copy().reset_index(drop=True)
                query_id=blast_df.iloc[0,0].split("|")[0].split("_")[0].lower()
                query_scope_id=queries[queries['pdb_id']==query_id]['scope_family'].
                               iloc[0]
                scope_id = query_scope_id
                for item in blast_df_merge.index:
                    if len(set(blast_df_merge['scope_family'][item].split(' '))
                          & set([scope_id])) > 0:
                        blast_df_merge.at[item, 'scope_family_annotation'] = "TP"
                    else:
```

```
                    blast_df_merge.at[item, 'scope_family_annotation'] = "FP"

            for item in blast_df_merge.index:
                if len(set(['.'.join(i.split('.')[:3]) for i in
                        blast_df_merge['scope_family'][item].split(' ')]) &
                        set(['.'.join(scope_id.split('.')[:3])])) > 0:
                    blast_df_merge.at[item, 'superfamily_annotation'] = "TP"
                else:
                    blast_df_merge.at[item, 'superfamily_annotation'] = "FP"
            for item in blast_df_merge.index:
                if len(set(['.'.join(i.split('.')[:2]) for i in
                       blast_df_merge['scope_family'][item].split(' ')]) &
                       set(['.'.join(scope_id.split('.')[:2])])) > 0:
                    blast_df_merge.at[item, 'fold_annotation'] = "TP"
                else:
                    blast_df_merge.at[item, 'fold_annotation'] = "FP"
            for item in blast_df_merge.index:
                if len(set(['.'.join(i.split('.')[:1]) for i in
                     blast_df_merge['scope_family'][item].split(' ')]) &
                     set(['.'.join(scope_id.split('.')[:1])])) > 0:
                    blast_df_merge.at[item, 'class_annotation'] = "TP"
                else:
                    blast_df_merge.at[item, 'class_annotation'] = "FP"
            blast_df_merge.to_csv(file_path+"_annotated.csv", header=True,
                    index=False)
        else:
            print(file_path + " is empty")
    else:
        continue
return None
```

Negatives

```
def negatives(path):
    directory = os.listdir(path)
    for file in directory:
        if file.endswith(".fasta.txt"):
            print(file, end = ", ")
            file_path = os.path.join(path, file)
            if os.stat(file_path).st_size != 0:
                blast_result = pd.read_table(file_path, delim_whitespace=True,
```

```
                        names=["qseqid", "sseqid", "pident", "length",
                        "mismatch", "gapopen", "qstart", "qend", "sstart",
                        "send", "evalue", "bitscore"])
            blast_df = blast_result.drop_duplicates(subset='sseqid',
                       inplace=False).reset_index(drop=True)
            query_id = blast_df.iloc[0,0].split("|")[0].split("_")[0].lower()
            scope_id=queries[queries['pdb_id']==query_id]['scope_family'].iloc[0]
            merge_blast_scope= pd.merge(blast_df, subset, left_on='sseqid',
                           right_on='pdb_chain', how = "inner")
            blast_df_negatives=subset[(~subset.pdb_chain.isin
                        (merge_blast_scope.sseqid))].reset_index(drop=True)
            blast_df_negatives['scope_family_annotation'] = np.where
                   (blast_df_negatives['scope_family'].values== scope_id, "FN", "TN")
            for index in blast_df_negatives.index:
                if '.'.join(blast_df_negatives.at[index, 'scope_family'].
                           split('.')[:3]) == '.'.join(scope_id.split('.')[:3]):
                    blast_df_negatives.at[index, 'superfamily_annotation'] = "FN"
                else:
                    blast_df_negatives.at[index, 'superfamily_annotation'] = "TN"
            for index in blast_df_negatives.index:
                if '.'.join(blast_df_negatives.at[index, 'scope_family'].
                           split('.')[:2]) == '.'.join(scope_id.split('.')[:2]):
                    blast_df_negatives.at[index, 'fold_annotation'] = "FN"
                else:
                    blast_df_negatives.at[index, 'fold_annotation'] = "TN"

            for index in blast_df_negatives.index:
                if '.'.join(blast_df_negatives.at[index, 'scope_family'].
                            split('.')[:1]) == '.'.join(scope_id.split('.')[:1]):
                    blast_df_negatives.at[index, 'class_annotation'] = "FN"
                else:
                    blast_df_negatives.at[index, 'class_annotation'] = "TN"
            blast_df_negatives.to_csv(file_path+"_negatives.csv",
                 header=True, index=False)
        else:
            print(file_path + " is empty")
    else:
        continue
return None
```

Calculating Metrics

Threshold

```
path = "/Users/violet/Downloads/ncbi-blast-2.11.0+/blast/db/e_value/
        results_evalue_10"
directory = os.listdir(path)
frames = []
for file in directory:
    if file.endswith(".fasta.txt"):
        file_path = os.path.join(path, file)
        if os.stat(file_path).st_size != 0:
            frame = pd.read_table(file_path, delim_whitespace=True,
                    names=["qseqid", "sseqid", "pident", "length",
                    "mismatch", "gapopen", "qstart", "qend", "sstart",
                    "send", "evalue", "bitscore"])
            frames.append(frame)
        else:
            print(file_path + " is empty")
    else:
        continue
all_results = pd.concat(frames)
thr = all_results['evalue'].unique().tolist()
thr.sort()
thr = thr[::30] + [10]
```

Family group

```
def calculates_negatives(path):
    scope_TN = 0
    scope_FN = 0
    i = 0
    for file in os.listdir(path):
        if file.endswith("_negatives.csv"):
            i += 1
            file_path = os.path.join(path, file)
            if os.stat(file_path).st_size != 0:
                df_neg = pd.read_csv(file_path, sep = ",")
                tn_df = len(df_neg[df_neg['scope_family_annotation'] == "TN"])
                scope_TN += tn_df
                fn_df = len(df_neg[df_neg['scope_family_annotation'] != "TN"])
```

```
                scope_FN += fn_df
        else:
            continue
    print(i)
    return scope_TN, scope_FN


def calculates_family_metrics(path, thresholds):
    fpr = []
    tpr = []
    recall = []
    precision = []
    metrics = []
    scope_TN, scope_FN = calculates_negatives(path)
    print(scope_TN, scope_FN)
    for threshold in thresholds:
        #print(threshold, end =", ")
        new_row = []
        new_row.append(threshold)
        TP = 0
        FP = 0
        TN = scope_TN
        FN = scope_FN
        for file in os.listdir(path):
            if file.endswith("_annotated.csv"):
                file_path = os.path.join(path, file)
                if os.stat(file_path).st_size != 0:
                    blast_assigned = pd.read_csv(file_path, sep = ",")
                    query_id = blast_assigned.iloc[0,0].
                               split("|")[0].split("_")[0].lower()
                    scope_id = queries[queries['pdb_id']==query_id]
                               ['scope_family'].iloc[0]
                    positives = blast_assigned[blast_assigned['evalue']<=threshold]
                    TP_b = len(positives[positives['scope_family_annotation'] == "TP"])
                    TP += TP_b
                    FP_b = len(positives[positives['scope_family_annotation'] != "TP"])
                    FP += FP_b
                    negatives = blast_assigned[blast_assigned['evalue']>threshold]
                    negatives['scope_family_annotation'] = np.where
                    (negatives['scope_family_annotation'].values== "FP", "TN", "FN")
```

```
                    TN_b = len(negatives[negatives['scope_family_annotation'] == "TN"])
                    FN_b = len(negatives[negatives['scope_family_annotation'] != "TN"])
                    TN += TN_b
                    FN += FN_b
                else:
                    continue
            else:
                continue
        new_row.append(TP)
        new_row.append(FP)
        new_row.append(TN)
        new_row.append(FN)
        fpr.append(FP/(FP + TN))
        new_row.append(FP/(FP + TN))
        tpr.append(TP/(TP + FN))
        new_row.append(TP/(TP + FN))
        recall.append(TP/(TP + FN))
        new_row.append(TP/(TP + FN))
        precision.append(TP/(TP + FP))
        new_row.append(TP/(TP + FP))
        metrics.append(new_row)
    return fpr, tpr, recall, precision, metrics
```

Super-family

```
def calculates_negatives(path):
    scope_TN = 0
    scope_FN = 0
    i = 0
    for file in os.listdir(path):
        if file.endswith("_negatives.csv"):
            i += 1
            file_path = os.path.join(path, file)
            if os.stat(file_path).st_size != 0:
                df_neg = pd.read_csv(file_path, sep = ",")
                tn_df = len(df_neg[df_neg['superfamily_annotation'] == "TN"])
                scope_TN += tn_df
                fn_df = len(df_neg[df_neg['superfamily_annotation'] != "TN"])
                scope_FN += fn_df
        else:
```

```
            continue
    print(i)
    return scope_TN, scope_FN

def calculates_superfamily_metrics(path, thresholds):
    fpr = []
    tpr = []
    recall = []
    precision = []
    metrics = []
    scope_TN, scope_FN = calculates_negatives(path)
    print(scope_TN, scope_FN)
    for threshold in thresholds:
        #print(threshold, end =", ")
        new_row = []
        new_row.append(threshold)
        TP = 0
        FP = 0
        TN = scope_TN
        FN = scope_FN
        for file in os.listdir(path):
            if file.endswith("_annotated.csv"):
                file_path = os.path.join(path, file)
                if os.stat(file_path).st_size != 0:
                    blast_assigned = pd.read_csv(file_path, sep = ",")
                    query_id = blast_assigned.iloc[0,0].split("|")[0].split("_")[0].
                             lower()
                    scope_id = queries[queries['pdb_id']==query_id]['scope_family'].
                            iloc[0]
                    positives = blast_assigned[blast_assigned['evalue']<=threshold]
                    TP_b = len(positives[positives['superfamily_annotation'] == "TP"])
                    TP += TP_b
                    FP_b = len(positives[positives['superfamily_annotation'] != "TP"])
                    FP += FP_b
                    negatives = blast_assigned[blast_assigned['evalue']>threshold]
                    negatives['superfamily_annotation'] = np.where
                        (negatives['superfamily_annotation'].values== "FP", "TN", "FN")
                    TN_b = len(negatives[negatives['superfamily_annotation'] == "TN"])
                    FN_b = len(negatives[negatives['superfamily_annotation'] != "TN"])
                    TN += TN_b
```

```
                    FN += FN_b
                else:
                    continue
            else:
                continue
        new_row.append(TP)
        new_row.append(FP)
        new_row.append(TN)
        new_row.append(FN)
        fpr.append(FP/(FP + TN))
        new_row.append(FP/(FP + TN))
        tpr.append(TP/(TP + FN))
        new_row.append(TP/(TP + FN))
        recall.append(TP/(TP + FN))
        new_row.append(TP/(TP + FN))
        precision.append(TP/(TP + FP))
        new_row.append(TP/(TP + FP))
        metrics.append(new_row)
    return fpr, tpr, recall, precision, metrics
```

Fold group

```
def calculates_negatives(path):
    scope_TN = 0
    scope_FN = 0
    i = 0
    for file in os.listdir(path):
        if file.endswith("_negatives.csv"):
            i += 1
            file_path = os.path.join(path, file)
            if os.stat(file_path).st_size != 0:
                df_neg = pd.read_csv(file_path, sep = ",")
                tn_df = len(df_neg[df_neg['fold_annotation'] == "TN"])
                scope_TN += tn_df
                fn_df = len(df_neg[df_neg['fold_annotation'] != "TN"])
                scope_FN += fn_df
        else:
            continue
    print(i)
    return scope_TN, scope_FN
```

```
def calculates_fold_metrics(path, thresholds):
    fpr = []
    tpr = []
    recall = []
    precision = []
    metrics = []
    scope_TN, scope_FN = calculates_negatives(path)
    print(scope_TN, scope_FN)
    for threshold in thresholds:
        #print(threshold, end =", ")
        new_row = []
        new_row.append(threshold)
        TP = 0
        FP = 0
        TN = scope_TN
        FN = scope_FN
        for file in os.listdir(path):
            if file.endswith("_annotated.csv"):
                file_path = os.path.join(path, file)
                if os.stat(file_path).st_size != 0:
                    blast_assigned = pd.read_csv(file_path, sep = ",")
                    query_id = blast_assigned.iloc[0,0].split("|")[0].split("_")[0].
                            lower()
                    scope_id = queries[queries['pdb_id']==query_id]['scope_family'].
                            iloc[0]
                    positives = blast_assigned[blast_assigned['evalue']<=threshold]
                    TP_b = len(positives[positives['fold_annotation'] == "TP"])
                    TP += TP_b
                    FP_b = len(positives[positives['fold_annotation'] != "TP"])
                    FP += FP_b
                    negatives = blast_assigned[blast_assigned['evalue']>threshold]
                    negatives['fold_annotation'] = np.where
                          (negatives['fold_annotation'].values== "FP", "TN", "FN")
                    TN_b = len(negatives[negatives['fold_annotation'] == "TN"])
                    FN_b = len(negatives[negatives['fold_annotation'] != "TN"])
                    TN += TN_b
                    FN += FN_b
                else:
                    continue
```

```
            else:
                continue
        new_row.append(TP)
        new_row.append(FP)
        new_row.append(TN)
        new_row.append(FN)
        fpr.append(FP/(FP + TN))
        new_row.append(FP/(FP + TN))
        tpr.append(TP/(TP + FN))
        new_row.append(TP/(TP + FN))
        recall.append(TP/(TP + FN))
        new_row.append(TP/(TP + FN))
        precision.append(TP/(TP + FP))
        new_row.append(TP/(TP + FP))
        metrics.append(new_row)
    return fpr, tpr, recall, precision, metrics
```

Class group

```
def calculates_negatives(path):
    scope_TN = 0
    scope_FN = 0
    i = 0
    for file in os.listdir(path):
        if file.endswith("_negatives.csv"):
            i += 1
            file_path = os.path.join(path, file)
            if os.stat(file_path).st_size != 0:
                df_neg = pd.read_csv(file_path, sep = ",")
                tn_df = len(df_neg[df_neg['class_annotation'] == "TN"])
                scope_TN += tn_df
                fn_df = len(df_neg[df_neg['class_annotation'] != "TN"])
                scope_FN += fn_df
        else:
            continue
    print(i)
    return scope_TN, scope_FN

def calculates_class_metrics(path, thresholds):
    fpr = []
```

```
tpr = []
recall = []
precision = []
metrics = []
scope_TN, scope_FN = calculates_negatives(path)
print(scope_TN, scope_FN)
for threshold in thresholds:
    #print(threshold, end =", ")
    new_row = []
    new_row.append(threshold)
    TP = 0
    FP = 0
    TN = scope_TN
    FN = scope_FN
    for file in os.listdir(path):
        if file.endswith("_annotated.csv"):
            file_path = os.path.join(path, file)
            if os.stat(file_path).st_size != 0:
                blast_assigned = pd.read_csv(file_path, sep = ",")
                query_id = blast_assigned.iloc[0,0].split("|")[0].split("_")[0].
                        lower()
                scope_id = queries[queries['pdb_id']==query_id]['scope_family'].
                          iloc[0]
                positives = blast_assigned[blast_assigned['evalue']<=threshold]
                TP_b = len(positives[positives['class_annotation'] == "TP"])
                TP += TP_b
                FP_b = len(positives[positives['class_annotation'] != "TP"])
                FP += FP_b
                negatives = blast_assigned[blast_assigned['evalue']>threshold]
                negatives['class_annotation'] = np.where
                      (negatives['class_annotation'].values== "FP", "TN", "FN")
                TN_b = len(negatives[negatives['class_annotation'] == "TN"])
                FN_b = len(negatives[negatives['class_annotation'] != "TN"])
                TN += TN_b
                FN += FN_b
            else:
                continue
        else:
            continue
    new_row.append(TP)
```

```
        new_row.append(FP)
        new_row.append(TN)
        new_row.append(FN)
        fpr.append(FP/(FP + TN))
        new_row.append(FP/(FP + TN))
        tpr.append(TP/(TP + FN))
        new_row.append(TP/(TP + FN))
        recall.append(TP/(TP + FN))
        new_row.append(TP/(TP + FN))
        precision.append(TP/(TP + FP))
        new_row.append(TP/(TP + FP))
        metrics.append(new_row)
    return fpr, tpr, recall, precision, metrics

# Without 2GFS and 4HVI
def without_2GFS_negatives(path):
    scope_TN = 0
    scope_FN = 0
    i = 0
    for file in os.listdir(path):
        if file.endswith("_negatives.csv"):
            if file.split("_")[1] == "2GFS" or file.split("_")[1] == "4HVI":
                continue
            else:
                i += 1
                file_path = os.path.join(path, file)
                if os.stat(file_path).st_size != 0:
                    df_neg = pd.read_csv(file_path, sep = ",")
                    tn_df = len(df_neg[df_neg['scope_family_annotation'] == "TN"])
                    scope_TN += tn_df
                    fn_df = len(df_neg[df_neg['scope_family_annotation'] != "TN"])
                    scope_FN += fn_df
        else:
            continue
    print(i)
    return scope_TN, scope_FN

def without_2GFS(path, thresholds):
    fpr = []
    tpr = []
```

```
recall = []
precision = []
metrics = []
scope_TN, scope_FN = without_2GFS_negatives(path)
print(scope_TN, scope_FN)
for threshold in thresholds:
    #print(threshold, end =", ")
    new_row = []
    new_row.append(threshold)
    TP = 0
    FP = 0
    TN = scope_TN
    FN = scope_FN
    i = 0
    for file in os.listdir(path):
        if file.endswith("_annotated.csv"):
            if file.split("_")[1] == "2GFS" or file.split("_")[1] == "4HVI":
                continue
            else:
                i += 1
                file_path = os.path.join(path, file)
                if os.stat(file_path).st_size != 0:
                    blast_assigned = pd.read_csv(file_path, sep = ",")
                    query_id = blast_assigned.iloc[0,0].split("|")[0].split("_")[0].
                           lower()
                    scope_id = queries[queries['pdb_id']==query_id]['scope_family'].
                            iloc[0]
                    positives = blast_assigned[blast_assigned['evalue']<=threshold]
                    TP_b = len(positives[positives['scope_family_annotation'] == "TP"])
                    TP += TP_b
                    FP_b = len(positives[positives['scope_family_annotation'] != "TP"])
                    FP += FP_b
                    negatives = blast_assigned[blast_assigned['evalue']>threshold]
                    negatives['scope_family_annotation'] = np.where
                       (negatives['scope_family_annotation'].values== "FP", "TN", "FN")
                    TN_b = len(negatives[negatives['scope_family_annotation'] == "TN"])
                    FN_b = len(negatives[negatives['scope_family_annotation'] != "TN"])
                    TN += TN_b
                    FN += FN_b
                else:
```

```
                    continue
            else:
                continue
        print(i, end = ", ")
        new_row.append(TP)
        new_row.append(FP)
        new_row.append(TN)
        new_row.append(FN)
        fpr.append(FP/(FP + TN))
        new_row.append(FP/(FP + TN))
        tpr.append(TP/(TP + FN))
        new_row.append(TP/(TP + FN))
        recall.append(TP/(TP + FN))
        new_row.append(TP/(TP + FN))
        precision.append(TP/(TP + FP))
        new_row.append(TP/(TP + FP))
        metrics.append(new_row)
    return fpr, tpr, recall, precision, metrics
```

ROC

```
# The ROC and precision-recall curve code was used to plot the ROC curves and
precision-recall curves by changing the parameter (e-value, wordsize, matrix, LCR)
and the classification (family, super-family, fold and class)

plt.figure(1)
plt.figure(figsize=(10, 10))
plt.plot(fpr_fam_word2, tpr_fam_word2, "k", label="word size = 2")
plt.plot(fpr_fam_word3, tpr_fam_word3, "r", label="word size = 3 (Default)")
plt.plot(fpr_fam_word4, tpr_fam_word4, "b", label="word size = 4")
plt.plot(fpr_fam_word5, tpr_fam_word5, "y", label="word size = 5")
plt.plot(fpr_fam_word6, tpr_fam_word6, "m", label="word size = 6")
plt.plot(fpr_fam_word7, tpr_fam_word7, "g", label="word size = 7")
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.legend()
#plt.show()
plt.savefig('Images/Family/ROC_word_size_fam.png')
```

Precision-recall curve

```
plt.figure(2)
```

```
plt.figure(figsize=(10, 10))
plt.plot(recall_fam_word2, precision_fam_word2, "k", label="word size =2")
plt.plot(recall_fam_word3, precision_fam_word3, "r", label="word size = 3
          (Default)")
plt.plot(recall_fam_word4, precision_fam_word4, "g", label="word size = 4")
plt.plot(recall_fam_word5, precision_fam_word5, "b", label="word size = 5")
plt.plot(recall_fam_word6, precision_fam_word6, "m", label="word size = 6")
plt.plot(recall_fam_word7, precision_fam_word7, "y", label="word size = 7")
pyplot.xlabel('Recall')
pyplot.ylabel('Precision')
plt.legend()
#plt.show()
plt.savefig('Images/Family/PR_word_size_fa.png')
```