Protein Family Classification based on the Protein Sequence using Machine Learning

A PROJECT REPORT

Submitted by

TEAM – Avengers

NAME	REGNO
D.SAI THARUN REDDY	BL.EN. U4AIE19016
SANDEEP PREETHAM MC	BL.EN. U4AIE19058
T. ROHITH	BL.EN. U4AIE19064

for the course

19BIO212- Introduction to Biological Systems

Guided and Evaluated by

Dr. Tripty singh Thakur
Asst. Prof (SG),
Dept. of CSE,

Dr. Amrita
Asst. Prof (SG),
Dept. of CSE,

AMRITA SCHOOL OF ENGINEERING, BANGALORE

AMRITA VISHWA VIDHYAPEETHAM

BANGALORE-560 035

December-2020

Acknowledgement

The satisfaction that accompanies successful completion of any task would be incomplete without mention of people who made it possible, and whose constant encouragement and guidance have been a source of inspiration throughout the course of this project work.

It is a great pleasure to express our gratitude and indebtedness to our project guide **Dr. Tripty singh** assistant professor (SG), Department of Computer Science and Engineering, **Dr. Amrita thakur** assistant professor (SG), Department of Chemistry, Amrita School of Engineering, Bangalore for her valuable guidance, encouragement, moral support and affection throughout the project work.

TABLE OF CONTENTS:

1) ABSTRACT 4

2) INTRODUCTION 5

3) METHOD 8

3.1 SUPERVISED ML METHODS 8

- Naive Bayes Classifier 9

4) MODEL IMPLEMENTATION 11

5) CONCLUSION AND FUTURE WORK 20

ABSTRACT

Classifying an unknown protein into a known protein family is a challenging task and one of the demanding problems is bioinformatics and computational biology. As proteins are the main target molecules while designing the drug for any disease, it is important to study and classify unknown proteins. Machine learning algorithms like Naive Bayes classifiers have been effectively used for such kinds of problems. In this project, our aim is to classify an unknown protein sequence into a known protein family using machine learning algorithms and to find that model performance. Here, the protein feature used for classification purposes is the probability of occurrence of a particular amino acid in the protein sequence. There are mainly 20 amino acids which form a protein. The idea here is proteins having nearly similar probability of occurrence of amino acids belong to the same family.

Keywords*:* Protein Family, Classification, Naive Bayes classifiers, amino acids.

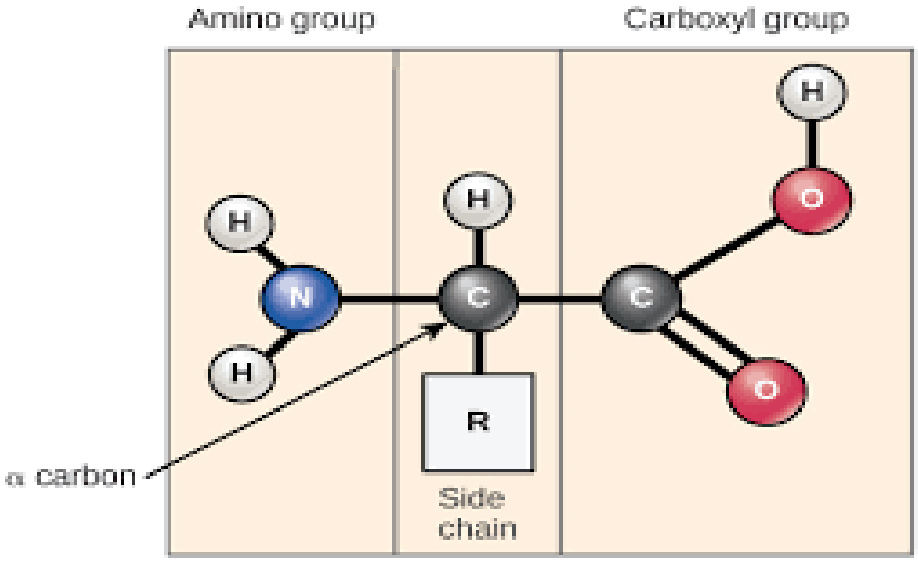
INTRODUCTION:

Protein sequences (also known as polypeptides) are organic compounds made of amino acids arranged in a linear chain and folded into a globular form. The amino acids in a polymer chain are joined together by the peptide bonds between the carboxyl and amino groups of adjacent amino acid residues. The sequence of amino acids in a protein is defined by the sequence of a gene, which is encoded in the genetic code. A protein family is a group of proteins that share a common evolutionary origin and have similarities in sequence or structure.

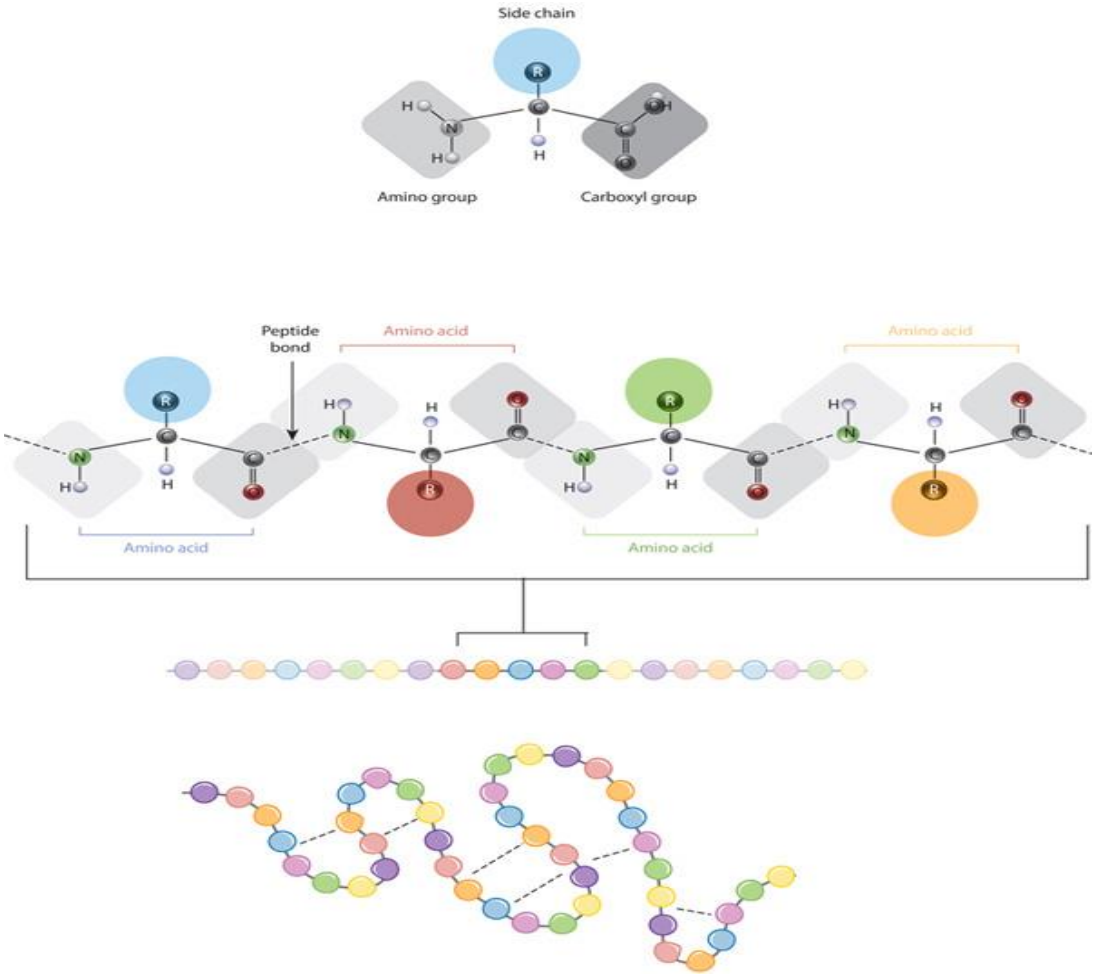
For example, considering we have a protein that belongs to a hydrolase group whose main function is catalyzing hydrolysis (i.e. breaking bonds by adding water) only. Consider another example where we shall consider a protein from the transport family whose main function is to allow other molecules such as sucrose, fructose, or even water to come in and outside of the cell. The proteins are basically classified into 3 types. Namely simple proteins, conjugated proteins and derived proteins. In our project we will be classifying the proteins based on the similarities in their structure using Naive Bayes algorithm.

What are Proteins?

Proteins are large, complex molecules that play many critical roles in the body. They do most of the work in cells and are required for the structure, function, and regulation of the body's tissues and organs, the building blocks of proteins are the amino acids, which are small organic molecules that consist of an alpha carbon atom linked to an amino group, a carboxyl group, a hydrogen atom and a variable component that is called a side chain.



All the amino acids within the protein are linked together by peptide bonds helping in forming a long chain. The linear structure of amino acids within a protein is considered the primary structure of the protein. Proteins are built on a set of twenty amino acids, each amino acid being unique from the other. In proteins the side chain is responsible for determining the charge polarity of the amino acid and hence is specific for each amino acid.



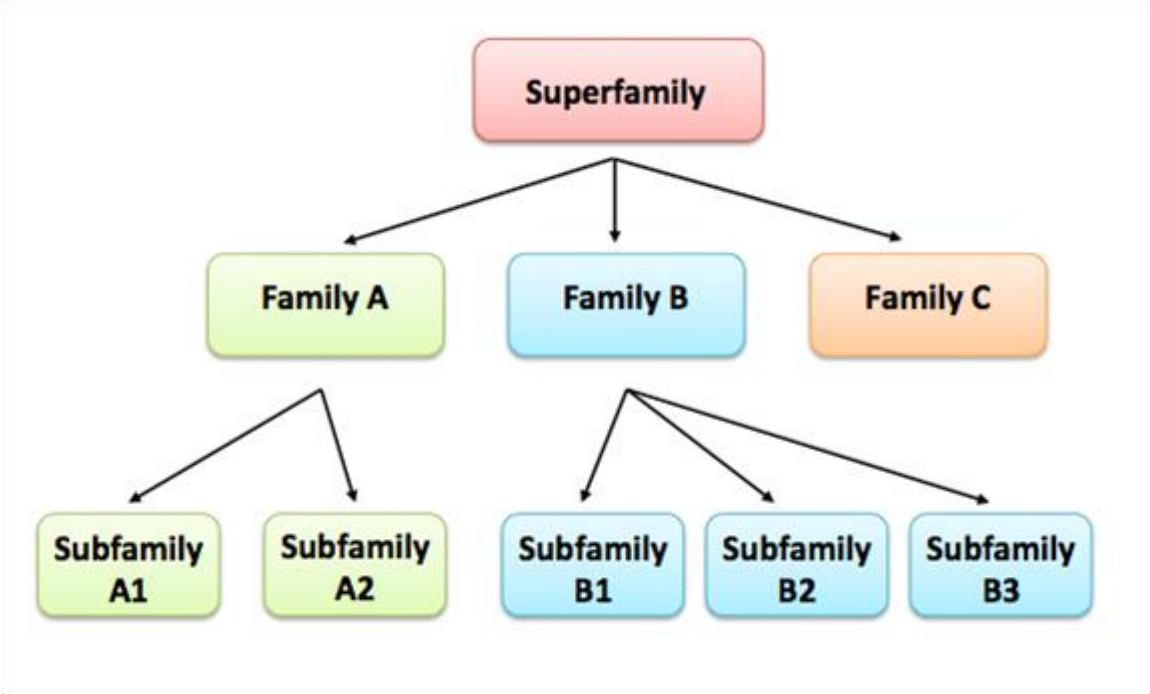
Protein sequence:

Protein sequencing is the practical process of determining the amino acid sequence of all or part of a protein or peptide. Protein Synthesis is the process of assembling amino acids to make proteins. Although DNA stores the information for protein synthesis and RNA carries out the instructions encoded in DNA, most biological activities are carried out by proteins. The accurate synthesis of proteins thus is 7 critical to the proper functioning of cells and organisms. The linear order of amino acids in each protein determines its three-dimensional structure and activity. For this reason, assembly of amino acids in their correct order, as encoded in DNA, is the key to production of functional proteins

What are Protein Families?

All proteins have to bind to other molecules in order to complete their tasks and the precise function of a protein depends on the way its exposed surfaces interact with the other molecules. Proteins with similar shape and sequence tend to perform similar functions within the cell and hence they are classified as a protein family.

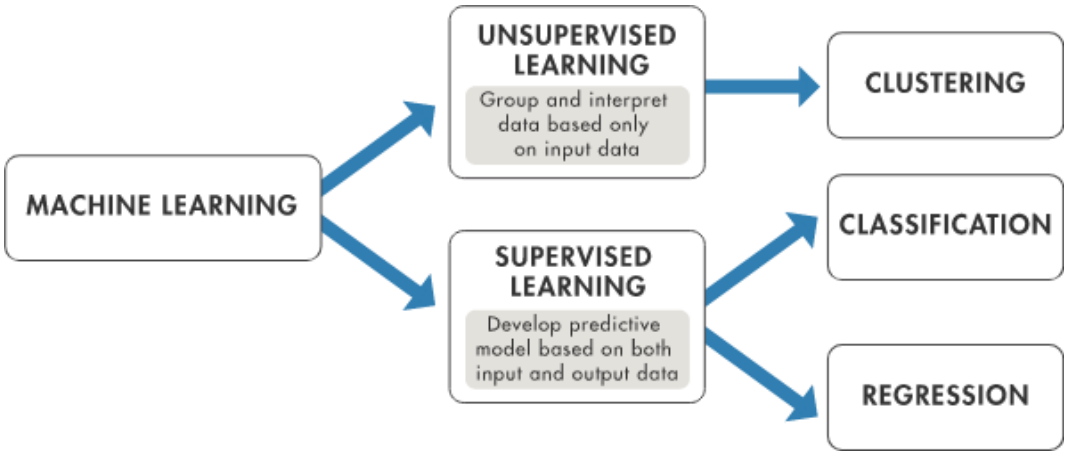
Proteins from the same family also often have long stretches of similar amino acid sequences within their primary structure. These protein families have many members, and they likely have evolved from ancient gene duplications and these duplications further led to modifications of protein functions and expanded the functional repertoire of organisms over time. It is estimated that over 60,000 protein families have been defined.



MACHINE LEARNING:

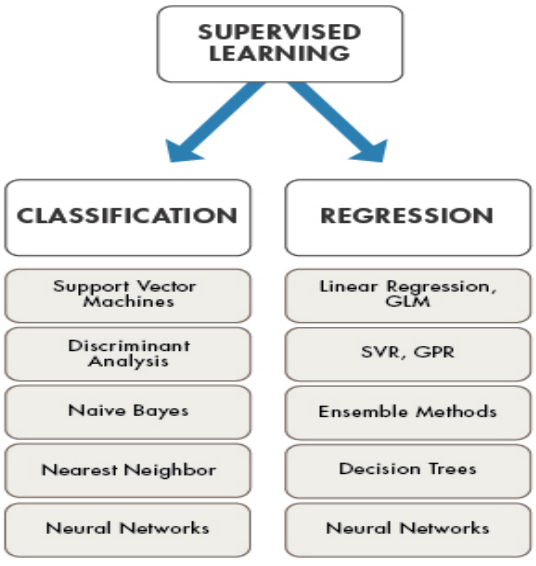
Machine Learning is a subset of artificial Intelligence that relies on the use of statistical learning algorithms. The key task of machine learning systems is to find patterns in existing data and then try to predict similar patterns in new data. To do so, machine learning systems continuously learn and improve themselves by processing structured data. This ability to learn from experience allows machine learning systems to automatically adapt to changes and make better predictions over time. In a supervised learning model, the algorithm learns on a labeled dataset, providing an answer key that the algorithm can use to evaluate its accuracy on training data.

An unsupervised model, in contrast, provides unlabeled data that the algorithm tries to make sense of by extracting features and patterns on its own.



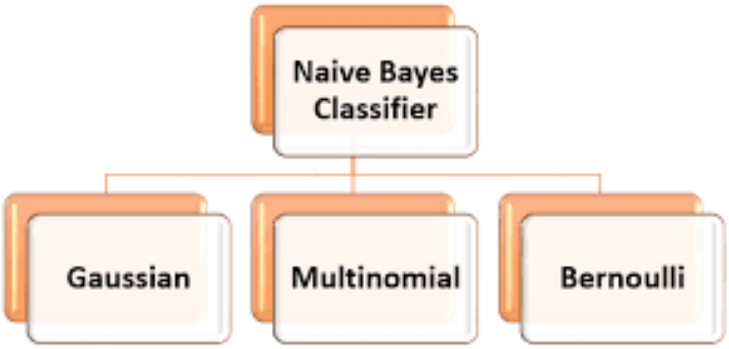
Supervised Learning:

Most machine learning tasks are in the domain of supervised learning. In supervised learning algorithms, the individual instances/data points in the dataset have a class or label assigned to them. This means that the machine learning model can learn to distinguish which features are correlated with a given class and that the machine learning engineer can check the model's performance by seeing how many instances were properly classified. Classification algorithms can be used to discern many complex patterns, as long as the data is labeled with the proper classes. There are two major types of supervised learning; classification and regression. Classification is where an algorithm is trained to classify input data on discrete variables. During training, algorithms are given training input data with a 'class' label.



Naive Bayes Classification:

Naive Bayes is a classification approach that adopts the principle of class conditional independence from the Bayes Theorem. This means that the presence of one feature does not impact the presence of another in the probability of a given outcome, and each predictor has an equal effect on that result. There are three types of Naive Bayes classifiers: Multinomial Naive Bayes, Bernoulli Naive Bayes, and Gaussian Naïve Bayes. This technique is primarily used in text classification, spam identification, and recommendation systems.



The main types of Naive Bayes classifier are mentioned below:

Multinomial Naive Bayes:

These types of classifiers are usually used for the problems of document classification. It checks whether the document belongs to a particular category like sports or technology or political etc. and then classifies them accordingly. The predictors used for classification in this technique are the frequency of words present in the document.

Bernoulli Naive Bayes:

This classifier is also analogous to multinomial naive bayes but instead of words, the predictors are Boolean values. The parameters used to predict the class variable accepts only yes or no values, for example, if a word occurs in the text or not.

Gaussian Naive Bayes:

A Gaussian Naive Bayes algorithm is a special type of NB algorithm. It's specifically used when the features have continuous values. It's also assumed that all the features are following a gaussian distribution i.e., normal distribution.

Naives Bayes Working:

The Naives Bayes classifier uses the probabilistic method alongside with the Bayes theorem which is easily scalable with a linear number of parameters. NB is a very efficient method of classification which has a very greater level of accuracy and is easy to learn as it mainly focuses on the dependent attributes and can easily deal with parametric attributes which are continuous. This algorithm has a technique that has the capability of assigning the labels that are given to the class with the problem instance which are represented in the form of the vectors of the values that are featured and the values of the class are taken from the set that is finite.

NB method needs a very little amount of training data set when compared with the other types of classification algorithms gives a greater accuracy.

Bayes ' Theorem provides a way to calculate the probability of a pre-knowledge hypothesis. Bayes ' Theorem is as follows:

$P (h/d) = P (d /h) * P(h)/P(d)$

P(h/d) is the probability of hypotheses based on d. The P(d/h) is posterior probability of data given that the hypothesis is true and after calculating the P (d/h) for a number of different hypotheses, the most probable hypothesis is preferred to be the maximum probable hypothesis and it referred as the maximum a posteriori (MAP) hypothesis.

This can be written as:

$MAP(h) = max (P(h /d))$ or

$MAP(h) = (P(d /h)*P(h) /P(d))$ or

$MAP(h) = max(P (d h) * P(h))$

P(d) is a normalizing term that can be dropped when trying to find the most probable hypothesis as it is constant, and there are even a number of instances in each class of training data, so the probability of each class will be equal. This would be a constant term in the equation and can be dropped, these giving rise to

MAP(h)max (P (d/h))

The representation of naive Bayes is a collection of probabilities contained in a database, which contains the default probabilities of the probabilities of each category in the training dataset and the conditional probabilities of the conditional probabilities of each input value of each class. Learning a naive Bayes model from training data is fast, because it is only necessary to calculate the probability of each class given different input (x) values, and no coefficients need to be fitted through optimization procedures. Conditional chances are the probability of each attribute value of a specified class value determined by the number of instances with that class value.

MODEL IMPLEMENTATION:

1). First we are going to import Libraries and Datasets that we need:

```
 # Import Libraries

import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from    sklearn.metrics    import    accuracy_score,    confusion_matrix,
classification_report

# Import Datasets
df_seq = pd.read_csv('pdb_data_seq.csv')
df_char = pd.read_csv('pdb_data_no_dups.csv')
```

2) Filter and Process Data

With the data loaded into two separate pandas data frames, a filter, project, and a join must be performed to get the data together. Begin by first filtering the datasets where the classification is equal to 'Protein', followed by removing all other variables other than structureId and sequence for the data_seq_csv, and structureId and classification in the no_dups dataset.

```
#Filter and Process Dataset

# Filter for only proteins
protein_char = df_char[df_char.macromoleculeType == 'Protein']
protein_seq = df_seq[df_seq.macromoleculeType == 'Protein']
```

```
# Select only necessary variables to join
protein_char = protein_char[['structureId','classification']]
protein_seq = protein_seq[['structureId','sequence']]
protein_seq.head()
```

Output:

	structureId	sequence
4	101M	MVLSEGEWQLVLHVWAKVEADVAGHGQDILIRLFKSHPETLEKFDR...
7	102L	MNIFEMLRIDEGLRLKIYKDTEGYYTIGIGHLLTKSPSLNAAAKSE...
8	102M	MVLSEGEWQLVLHVWAKVEADVAGHGQDILIRLFKSHPETLEKFDR...
11	103L	MNIFEMLRIDEGLRLKIYKDTEGYYTIGIGHLLTKSPSLNSLDAAK...
12	103M	MVLSEGEWQLVLHVWAKVEADVAGHGQDILIRLFKSHPETLEKFDR...

```
#print the first five rows of protein_char
protein_char.head()
```

Output:

	structureId	classification
2	101M	OXYGEN TRANSPORT
4	102L	HYDROLASE(O-GLYCOSYL)
5	102M	OXYGEN TRANSPORT
7	103L	HYDROLASE(O-GLYCOSYL)
8	103M	OXYGEN TRANSPORT

The protein_seq correctly contains proteins as different sequences start with Methionine, as well as sequences containing more letters than 'ACTG'. We can now perform a join using structureId as the index. We'll utilize pandas 'join method. To do this, we have to set the index for each data frame to be 'structureId'.

```
# Join two datasets on structureId
model_f=protein_char.set_index('structureId').join(protein_seq.set_index('structureId'))
model_f.head()
```

Output:

structureId	classification	sequence
101M	OXYGEN TRANSPORT	MVLSEGEWQLVLHVWAKVEADVAGHGQDILIRLFKSHPETLEKFDR...
102L	HYDROLASE(O-GLYCOSYL)	MNIFEMLRIDEGLRLKIYKDTEGYYTIGIGHLLTKSPSLNAAAKSE...
102M	OXYGEN TRANSPORT	MVLSEGEWQLVLHVWAKVEADVAGHGQDILIRLFKSHPETLEKFDR...
103L	HYDROLASE(O-GLYCOSYL)	MNIFEMLRIDEGLRLKIYKDTEGYYTIGIGHLLTKSPSLNSLDAAK...
103M	OXYGEN TRANSPORT	MVLSEGEWQLVLHVWAKVEADVAGHGQDILIRLFKSHPETLEKFDR...

The two data frames have officially been joined into one with 346,325 proteins. The data processing is not finished as it's important to take a look at the missingness associated with the columns.

```
# Check NA counts
model_f.isnull().sum()
```

Output:

```
classification    1
sequence          3
dtype: int64
```

```
# Drop rows with missing values
model_f = model_f.dropna()
print('%d is the number of proteins that have a classification and sequence' %model_f.shape[0])
```

Output:

```
346321 is the number of proteins that have a classification and sequence
```

Lastly, it's important to look at the types of family groups that classification can be.

```
# Look at classification type counts
counts = model_f.classification.value_counts()
print(counts)
```

Output:

```
HYDROLASE                                   46336
TRANSFERASE                                 36424
OXIDOREDUCTASE                              34321
IMMUNE SYSTEM                               15615
LYASE                                       11682
                                            ...
Actin-BINDING PROTEIN                           1
HYDROLASE, REPLICATION                          1
DNA BINDING PROTEIN,TRANSCRIPTION               1
BETA BETA ALPHA MOTIF                           1
HYDROLASE (ORGANOPHOSPHATE-DEGRADING)           1
Name: classification, Length: 4468, dtype: int64
```

There appears to be a wide distribution of counts for family types. It may be a good idea to filter for having a certain amount of records that are of a specific family type. 1,000 seems like a solid number that will allow a machine learning model to learn a pattern for a specific class.

```
# Get classification types where counts are over 1000
types = np.asarray(counts[(counts > 1000)].index)
print(len(types))
# Filter dataset's records for classification types > 1000
data = model_f[model_f.classification.isin(types)]
print(types)
print('%d is the number of records in the final filtered dataset'
%data.shape[0])
```

```
43
['HYDROLASE' 'TRANSFERASE' 'OXIDOREDUCTASE' 'IMMUNE SYSTEM' 'LYASE'
 'HYDROLASE/HYDROLASE INHIBITOR' 'TRANSCRIPTION' 'VIRAL PROTEIN'
 'TRANSPORT PROTEIN' 'VIRUS' 'SIGNALING PROTEIN' 'ISOMERASE' 'LIGASE'
 'MEMBRANE PROTEIN' 'PROTEIN BINDING' 'STRUCTURAL PROTEIN' 'CHAPERONE'
 'STRUCTURAL GENOMICS, UNKNOWN FUNCTION' 'SUGAR BINDING PROTEIN'
 'DNA BINDING PROTEIN' 'PHOTOSYNTHESIS' 'ELECTRON TRANSPORT'
 'TRANSFERASE/TRANSFERASE INHIBITOR' 'METAL BINDING PROTEIN'
 'CELL ADHESION' 'UNKNOWN FUNCTION' 'PROTEIN TRANSPORT' 'TOXIN'
 'CELL CYCLE' 'RNA BINDING PROTEIN' 'DE NOVO PROTEIN' 'HORMONE'
 'GENE REGULATION' 'OXIDOREDUCTASE/OXIDOREDUCTASE INHIBITOR' 'APOPTOSIS'
 'MOTOR PROTEIN' 'PROTEIN FIBRIL' 'METAL TRANSPORT'
 'VIRAL PROTEIN/IMMUNE SYSTEM' 'CONTRACTILE PROTEIN' 'FLUORESCENT PROTEIN'
 'TRANSLATION' 'BIOSYNTHETIC PROTEIN']
278866 is the number of records in the final filtered dataset
```

3).Train and Test Split:

After finally filtering the dataset, a split on the data to create a training and testing set must be performed. After splitting the data, it's important to utilize the CountVectorizer to create a dictionary composed from the training dataset. This will extract individual characters or subsets of characters to gain features. An important note about using the CountVectorizer is the specification of the ngram_range.

In a protein, it's not the individual amino acid in the protein that gives identification to what it's purpose is. There are secondary and tertiary structures that are formed via the bonds of the amino acids in the sequence. Furthermore, different parts of the chain can be more basic, and others can be more acidic, indicating its importance to use features that are larger than just one unit.

As a result, using an ngram_range of (4,4) seems to be a legitimate choice for feature extraction. This will extract different subsets that are of length 4 allowing the amino acids to use their neighbors to aid in our classification.

```
## Could add n-grams
def char_grams(text,n=3,jump_size=2):

    return [text[i:i+n] for i in range(0,len(text)-n+1,jump_size)]

data.head(3).sequence.apply(char_grams)
```

Output:

```
structureId
10GS    [PPY, YTV, VVY, YFP, PVR, RGR, RCA, AAL, LRM, ...
10GS    [PPY, YTV, VVY, YFP, PVR, RGR, RCA, AAL, LRM, ...
117E    [TYT, TTR, RQI, IGA, AKN, NTL, LEY, YKV, VYI, ...
Name: sequence, dtype: object
```

```
data["3mers"] = data.sequence.apply(char_grams)
```

```
data.head()
```

Output:

structureId	classification	sequence	3mers
10GS	TRANSFERASE/TRANSFERASE INHIBITOR	PPYTVVYFPVRGRCAALRMLLADQGQSWKEEVVTVETWQEGSLKAS...	[PPY, YTV, VVY, YFP, PVR, RGR, RCA, AAL, LRM, ...
10GS	TRANSFERASE/TRANSFERASE INHIBITOR	PPYTVVYFPVRGRCAALRMLLADQGQSWKEEVVTVETWQEGSLKAS...	[PPY, YTV, VVY, YFP, PVR, RGR, RCA, AAL, LRM, ...
117E	HYDROLASE	TYTTRQIGAKNTLEYKVYIEKDGKPVSAFHDIPLYADKENNIFNMV...	[TYT, TTR, RQI, IGA, AKN, NTL, LEY, YKV, VYI, ...
117E	HYDROLASE	TYTTRQIGAKNTLEYKVYIEKDGKPVSAFHDIPLYADKENNIFNMV...	[TYT, TTR, RQI, IGA, AKN, NTL, LEY, YKV, VYI, ...
11AS	LIGASE	MKTAYIAKQRQISFVKSHFSRQLEERLGLIEVQAPILSRVGDGTQD...	[MKT, TAY, YIA, AKQ, QRQ, QIS, SFV, VKS, SHF, ...

```
# Train Test Split

# Split Data
X_train,   X_test,y_train,y_test   =   train_test_split(data['sequence'],
data['classification'], test_size = 0.2, random_state = 1)

# Create a Countvectorizer to gather the unique elements in sequence
vect = CountVectorizer(analyzer = 'char_wb', ngram_range = (4,4))

# Fit and Transform CountVectorizer
vect.fit(X_train)
X_train_df = vect.transform(X_train)
X_test_df = vect.transform(X_test)

#Print a few of the features
print(vect.get_feature_names()[20:])
```

Output:

```
[' aay', ' aca', ' acd', ' ace', ' acf', ' acg', ' ach', ' ack', ' acm', ' acn', ' acp', ' acq', ' acr', ' acs', ' act', ' acv', ' acw',
```

4). Machine Learning Models

With the features extracted, it's time to use machine learning models. Traditionally a Naive Bayes approach works well for these types of count vectorized features. Adaboost will be used as well for comparison.

```
# Machine Learning Models

# Make a prediction dictionary to store accuracies

prediction = dict()

# Naive Bayes Model

from sklearn.naive_bayes import MultinomialNB

model = MultinomialNB()

model.fit(X_train_df, y_train)

NB_pred = model.predict(X_test_df)

prediction["MultinomialNB"] = accuracy_score(NB_pred, y_test)

print( prediction['MultinomialNB'])
```

Output:

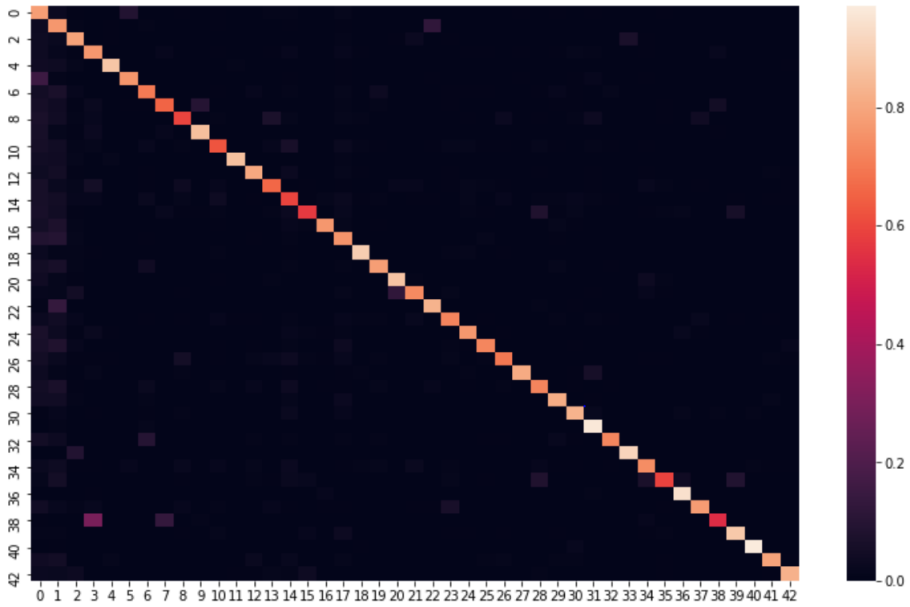
```
0.7638505396779861
```

5). Visualize Metrics

A visualization of a confusion matrix and a classification report for the Naive Bayes.

```
#Plot Confusion Matrix for NB

# Plot confusion matrix

conf_mat = confusion_matrix(y_test, NB_pred, labels = types)

#Normalize confusion_matrix

conf_mat= conf_mat.astype('float')/ conf_mat.sum(axis=1)[:, np.newaxis]

# Plot Heat Map

fig , ax = plt.subplots()

fig.set_size_inches(13, 8)

sns.heatmap(conf_mat)
```

Output:



Lastly, a matrix of the classification report to show case metrics for each class should be insightful

```
#Print F1 score metrics

print(classification_report(y_test, NB_pred, target_names = types))
```

Output:

	precision	recall	f1-score	support
HYDROLASE	0.47	0.74	0.57	250
TRANSFERASE	0.60	0.82	0.70	211
OXIDOREDUCTASE	0.75	0.77	0.76	589
IMMUNE SYSTEM	0.66	0.72	0.69	509
LYASE	0.91	0.76	0.83	859
HYDROLASE/HYDROLASE INHIBITOR	0.66	0.88	0.76	224
TRANSCRIPTION	0.58	0.83	0.68	326
VIRAL PROTEIN	0.71	0.78	0.75	622
TRANSPORT PROTEIN	0.60	0.74	0.66	601
VIRUS	0.91	0.97	0.94	209
SIGNALING PROTEIN	0.71	0.73	0.72	309
ISOMERASE	0.49	0.96	0.65	293
LIGASE	0.77	0.78	0.77	9274
MEMBRANE PROTEIN	0.67	0.76	0.71	2228
PROTEIN BINDING	0.87	0.76	0.82	3204
STRUCTURAL PROTEIN	0.94	0.86	0.90	1312
CHAPERONE	0.87	0.80	0.84	963
STRUCTURAL GENOMICS, UNKNOWN FUNCTION	0.94	0.87	0.90	2324
SUGAR BINDING PROTEIN	0.68	0.66	0.67	991
DNA BINDING PROTEIN	0.70	0.72	0.71	607
PHOTOSYNTHESIS	0.62	0.78	0.69	237
ELECTRON TRANSPORT	0.52	0.59	0.55	251
TRANSFERASE/TRANSFERASE INHIBITOR	0.93	0.79	0.85	6895
METAL BINDING PROTEIN	0.37	0.91	0.53	308
CELL ADHESION	0.81	0.87	0.84	686
UNKNOWN FUNCTION	0.53	0.59	0.56	953
PROTEIN TRANSPORT	0.81	0.94	0.87	269
TOXIN	0.73	0.70	0.71	526
CELL CYCLE	0.64	0.81	0.72	396
RNA BINDING PROTEIN	0.76	0.62	0.68	1284
DE NOVO PROTEIN	0.48	0.76	0.59	710
HORMONE	0.86	0.57	0.69	884
GENE REGULATION	0.84	0.89	0.86	690
OXIDOREDUCTASE/OXIDOREDUCTASE INHIBITOR	0.79	0.81	0.80	541
APOPTOSIS	0.81	0.70	0.75	1652
MOTOR PROTEIN	0.78	0.77	0.77	7262
PROTEIN FIBRIL	0.35	0.83	0.49	614
METAL TRANSPORT	0.69	0.79	0.73	209
VIRAL PROTEIN/IMMUNE SYSTEM	0.82	0.59	0.69	1641
CONTRACTILE PROTEIN	0.73	0.73	0.73	583
FLUORESCENT PROTEIN	0.86	0.66	0.74	1703
TRANSLATION	0.41	0.54	0.46	224
BIOSYNTHETIC PROTEIN	0.87	0.86	0.86	1351
accuracy			0.76	55774
macro avg	0.71	0.77	0.73	55774
weighted avg	0.79	0.76	0.77	55774

CONCLUSION AND FUTURE WORK:

In our work we have analyzed the performance of Machine Learning model Naive Bayes Classification to classify the protein sequences to their respective families. This study performs the classification of protein sequences in the dataset of structural protein sequences obtained Kaggle. The results of the study suggest that the Naive Bayes Classification can be used for the classification of protein sequences.

Further improvement in protein classification performance may be attempted in the context of deep learning approaches and further utilizing factors such as pH, molecular weight, and other components may be able to yield more information on family groups. Furthermore, if possible, increase the length of the ngram_range to include more than just 4 characters to allow for higher interaction between the amino acids as reflected in reality