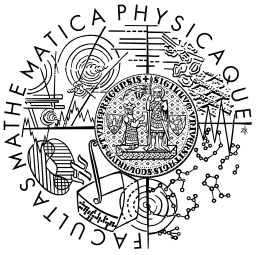


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Prokop Divín

Srovnání sekvenčních a strukturních metod strojového učení pro predikci protein-ligand vazebných reziduí

Katedra softwarového inženýrství

Vedoucí bakalářské práce: doc. RNDr. David Hoksza, Ph.D.
Studijní program: Informatika
Studijní obor: IPP5

Praha 2023

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Poděkování

V první řadě chci poděkovat své rodině a blízkým za podporu a pomoc v mojí cestě životem. Jsem vděčný za to, že mě podrželi, když jsem to potřeboval a stáli na mé straně v těžkých časech.

Zvláště chci poděkovat mému vedoucímu Davidu Hokszovi, který mi představil téma této práce a představil mi jak biologické, tak informatické pozadí nutné k vypracování této práce. Dále mu děkuji za vedení této práce, rady, které mě poskytl a také za navedení na správnou cestu když jsem se při vypracovávání ztratil. Další osoba, které chci poděkovat je Jakub Yaghob, kterému děkuji za pomoc při spouštění fakultního klastru pro provádění výpočtů. Také děkuji Hamza Gamouh za pomoc při generování embeddingů jehož projektu jsem pro tyto účely využil.

Název práce: Srovnání sekvenčních a strukturních metod strojového učení pro predikci protein-ligand vazebných reziduí

Autor: Prokop Divín

Katedra softwarového inženýrství: Katedra softwarového inženýrství

Vedoucí bakalářské práce: doc. RNDr. David Hoksza, Ph.D., Katedra softwarového inženýrství

Abstrakt: Předpověd protein-ligand vazebných míst je důležitým úkolem, dovolujícím nám pochopit interakce mezi proteinem a ligandem, jejichž pochopení je nezbytné při návrhu léčiv a rozvoji některých oblastí biologie. Ačkoliv již byly vytvořeny nástroje strojového učení pro predikci vazebných míst, tak doposud se vytvořené metody zajímaly pouze o predikci ze 3D struktury proteinu, která ale není pro většinu proteinů známá. Proto se v naší práci zajímáme o předpověď ze znalosti pouhé sekvence reziduí představující protein. Srovnáváme zde možné přístupy k řešení tohoto problému. Srovnáváme reprezentaci reziduí pomocí jejich chemicko-fyzikálních vlastností s reprezentaci používající metody z rozpoznávání přirozeného jazyka. Dále porovnáváme zvolené metody strojového učení. Na závěr porovnáme naše výsledky s P2Rank metodou, jakožto s nejmodernější metodou používající k předpovědi protein-ligand vazebných míst 3D strukturu.

Klíčová slova: bioinformatika protein strojové učení molekulární interakce

Title: Comparison of sequence and structure-based machine learning approaches for protein-ligand binding residues

Author: Prokop Divín

Department of Software Engineering: Department of Software Engineering

Supervisor: doc. RNDr. David Hoksza, Ph.D., Department of Software Engineering

Abstract: The prediction of protein-ligand binding sites is an important task, allowing us to understand protein-ligand interactions, the understanding of which is essential in drug design and the development of certain areas of biology. Although machine learning tools for binding site prediction have been developed, the methods developed so far have only been interested in prediction from the 3D structure of the protein, which is unknown for most proteins. Therefore, in our work we are interested in prediction from knowledge of the mere sequence of residues representing the protein. Here we compare possible approaches to solve this problem. We compare the representation of residues using their chemical and physical properties with a representation using methods from natural language recognition. Furthermore, we compare the chosen machine learning methods. Finally, we compare our results with the P2Rank method, as a state-of-the-art method using 3D structure to predict protein-ligand binding sites.

Keywords: bioinformatics protein machine learning molecular interactions

Obsah

Úvod

Proteiny jsou jednou ze základních složek živých organismů. Tyto molekuly slouží jako enzymy, transportní látky, hormony, jsou základním stavebním kamenem buněčných struktur a zastávají v organismu mnoho funkcí. Jednou z funkcí, kterou zastávají, je interakce s ostatními biologickými molekulami nebo jinými proteiny. Molekulám vázajícím se na biologický cíl, kterým často bývá protein, se říká ligand. Interakce mezi ligandem a proteinem jsou jedny z nejzásadnějších interakcí, vyskytujících se v biologických procesech. Jejich pochopení hraje velkou roli při návrhu léčiv a v rozvoji syntetické biologie. Ligand může interagovat s proteinem pouze na určitých místech. Určení takovýchto vazebných míst je prvním krokem k pochopení interakce mezi proteinem a ligandem.

Díky experimentálním metodám je možné označit vazebná místa na daných proteinech. Postupem času díky experimentálním metodám byly vytvořeny rozsáhlé databáze proteinů jako je například UniProt [1], s proteiny s označenými vazebnými místy. Existence těchto databází umožnila vznik bioinformatických nástrojů pro analýzu biologických dat a také umožnilo přistupovat k hledání vazebních míst výpočetně. Ačkoliv experimentální metody jsou účinnější n, tak se vyplatí mít k dispozici i výpočetní metody. Mezi výhody výpočetních metod patří jejich rychlost. Kvůli velkému množství proteinů a časové náročnosti experimentálních metod nelze aplikovat experimentální metody na všechny proteiny. Vyplatí se tedy mít rychlejší metodu, k čemuž mohou být nápomocné právě výpočetní metody.

Nástroje pro predikci vazebných míst vzniklé pomocí výpočetních metod, používají z pravidla k predikci 3D strukturu proteinu a předpovídají, kde na jejím povrchu se vyskytuje vazebné místo. Avšak i díky časové náročnosti experimentálních metod není 3D struktura u většiny proteinů známá. Většina proteinů je známá pouze jen jako sekvence zbytků aminokyselin, kterým se říká rezidua. Tento popis proteinů nedovoluje předpovídat vazebná místa na povrchu molekuly. Dovoluje pouze předpovědět rezidua na které se ligandy vážou a které vazebná místa tvoří. V této práci se budeme zabývat detekcí vazebných reziduí u proteinů popsaných jen pomocí sekvence reziduí a to za pomocí metod strojového učení.

Cíle práce jsou:

* Zvolit metody strojového učení pro vytvoření modelů predikujících vazebná místa.
* Implementovat nastroj pro testování metod strojového učení na detekci protein-ligand vazebných reziduí a následně jej použít na otestování zvolených metod.
* Porovnat reprezentaci reziduí a to jaký mají jejich kontextové a nekontextové vlastnosti vliv na předpověď vazebných míst.
* Ukázat účinnost metod na datasetech rozdělených podle druhů ligandů a porovnat tyto metody
* Ukázat účinnost metod na datasetech nerozdělených podle druhu ligandů.

* Porovnat modely s P2Rank, jednou z nejmodernějších metod předpovídání vazebních míst pro proteiny využívající 3D strukturu proteinu.

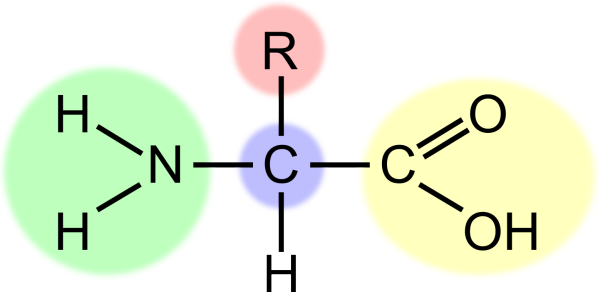
1. Biologický kontext

V této kapitole jsou shrnuty základní znalosti z molekulární biologie, o které se opírá tato práce. Nejprve jsou zde shrnuty základní poznatky o aminokyselinách, které jsou základní jednotkou proteinů a proteinech samotných. Poté si představíme, co je to ligand a popíšeme interakci mezi proteinem a ligandem.

1.1 Aminokyseliny

Jako aminokyseliny jsou v chemii označovány molekuly obsahující karboxylovou skupinu ($-COOH$) a aminovou funkční skupinu ($-NH_2$). My si ale v této práci budeme pod pojmem aminokyselina představovat pouze jednu z 23 aminokyselin, které jsou schopny tvořit proteiny. V praxi se budeme setkávat pouze z 20 druhy aminokyselin, protože 21. a 22. se nacházejí v proteinech jen vzácně a 23. působí jen jako inicializátor pro tvorbu proteinů v bakteriích.

Kromě karboxylové a aminové skupiny nás na aminokyselinách zajímá postranní řetězec, který je hlavní faktor, pomocí kterého se od sebe odlišují. Na obrázku 1.1 je vidět schéma aminokyseliny.



Obrázek 1.1: **Struktura aminokyseliny**
cit. 2023-04-15, Dostupné z: https://publicdomainvectors.org/en/free-clipart/Vector-image-of-amino-acid-scheme/29893.html

Zelenou barvou je zde vyznačena aminoskupina, žlutou karboxylová skupina a R znázorňuje postranní řetězec. Tento postranní řetězec je unikátní pro každou aminokyselinu.

1.1.1 Vlastnosti aminokyselin

Každou z aminokyselin můžeme popsat pomocí jejich vlastností. Tyto vlastnosti zde uvádíme, protože mohou být použity při přípravě dat pro metody strojového učení. Po vyjmenování vlastností následuje tabulka 1.2, kde jsou ke každé aminokyselině přiděleny určité vlastnosti.

Jméno

pro zkrácení názvu se z pravidla používá tří, nebo jedno písmenný kód. Zde můžeme vidět seznam aminokyselin a jejich kódy.

Číslo aminokyseliny	Jméno	3-písmenný kód	1-písmenný kód
1	Alanin	Ala	A
2	Arginin	Arg	R
3	Asparagin	Asn	N
4	Asparagová kyselina	Asp	D
5	Cystein	Cys	C
6	Glutamin	Gln	Q
7	Glutamová kyselina	Glu	E
8	Glycin	Gly	G
9	Histidin	His	H
10	Isoleucin	Ile	I
11	Leucin	Leu	L
12	Lysin	Lys	K
13	Methionin	Met	M
14	Fenylalanin	Phe	F
15	Prolin	Pro	P
16	Serin	Ser	S
17	Threonin	Thr	T
18	Tryptofan	Trp	W
19	Tyrosin	Tyr	Y
20	Valin	Val	V
21	Selenocystein	Sec	U
22	Pyrrolysine	Pyl	O

Tabulka 1.1: Kódy aminokyselin [2]

Třída

Třída aminokyselin se rozlišuje podle chemických vlastností bočního řetězce. Aminokyseliny můžeme dělit na tyto třídy:

1. **Alphiatické** - obsahují v postranním řetězci pouze uhlík a vodík, které spolu tvoří přímý, nebo rozvětvený řetězec, a nebo nearomatický cyklus. [4]

2. **S fixním kationtem** - obsahují kationt, což je kladně nabitá molekula, která vzniká z neutrálního kovu při ztrátě jednoho a více elektronů. U fixních kationtů se tento náboj nemění. [5]

3. **Amidy** - obsahují v postranním řetězci amid, což je sloučenina obsahující karboxylovou skupinu($C = O$) a aminovou skupinu($-NH_2$) [6]

4. **Aniony** - mají v postranním řetězci záporně nabytý iont, to je molekula, které přebývá jeden a více elektronů. Příkladem aniontu je: O^-,O^{2-}, OH^-. [7]

5. **Thioly** - nazývané také sirné alkoholy jsou sloučeniny síry odvoditelné od alkoholů nahrazením nejméně jednoho alkoholového atomu kyslíku sírou. Obecný předpis pro tuto skupinu je $(R-S-H)$, kde R reprezentuje zbytek molekuly. [8]

6. **Aromatické** - součástí jejich postranního řetězce je aromatický uhlovodík. To je uhlovodík obsahující benzenové jádro.

7. **s kationtem** - stejně jako u aminokyselin s fixním kationtem mají v postranním řetězci kladně nabitou molekulu tzv. kationt. Příkladem kationtů je: H^+,NH_4^+ [9]

8. **Cyklické** - do této skupiny jsme zařadili pouze Prolin, který je jedinou proteinogenní aminokyselinou která má svůj postranní řetězec spojený do cyklu přes aminovou skupinu.

9. **Hydroxylické** - mají v postranním řetězci hydroxylovou skupinu ($-OH$)

10. **Thioethery** -ve svém postranním řetězci nahrazuje atom kyslíku síra, jejich obecný vzorec je $R - S - R$ kde R je zbytek molekuly. [10]

Polarita

Polarita aminokyseliny je určena bočním řetězcem. Aminokyseliny dělíme podle polarity na:

1. **Nepolární** - obsahují uhlovodíkové zbytky, které jsou nepolární (hydrofobní) v proteinech se shlukují tak aby se vyhnuli kontaktu s vodou.

2. **Polární** -na rozdíl od nepolárních jsou tuto molekuly hydrofilní

3. **Brønstedova kyselina** - sloučenina, která je schopná darovat proton (H^+)

4. **Brønstedova zásada** - sloučenina, která je schopná přijmout proton (H^+)

5. **Basic polar** - jejich boční řetězec je polární ale není nabitý.

Náboj

Podle náboje dělíme aminokyseliny na:

1. **Neutrální**

2. **Pozitivní**

3. **Negativní**

Hydropatický index

Je číslo, které reprezentuje, do jaké míry je daná aminokyselina hydrofobní, nebo hydrofilní.

Molární hmotnost

Molární hmotnost je hmotnost jednoho molu látku, kde mol je $6.023 * 10^23$ částic dané látky. V tabulce níže je uvedena molární hmotnost v g/mol^{-1}.

Četnost

Je číslo v procentech, které udává, jak velkou část proteinů jednotlivé aminokyseliny tvoří.

Kód	Třída	Polarita	Náboj	Hydr.ind.	Hmotnost	Četnost
A	Aliphatic	Nonpolar	Neutral	1.8	89.094	8.76
R	Fixed cation	Basic polar	Positive	-4.5	174.203	5.78
N	Amide	Polar	Neutral	-3.5	132.119	3.93
D	Anion	Bronsted base	Negative	-3.5	133.104	5.49
C	Thiol	Bronsted acid	Neutral	2.5	121.154	1.38
E	Anion	Bronsted base	Negative	-3.5	147.131	6.32
Q	Amide	Polar	Neutral	-3.5	146.146	3.9
G	Aliphatic	Nonpolar	Neutral	-0.4	75.067	7.03
H	Aromatic cation	Bronsted acid and base	Neutral	-3.2	155	2.26
I	Aliphatic	Nonpolar	Neutral	4.5	131.175	5.49
L	Aliphatic	Nonpolar	Neutral	3.8	131.175	9.68
K	Cation	Bronsted acid	Positive	-3.9	146.189	5.19
M	Thioether	Nonpolar	Neutral	1.9	149.208	2.32
F	Aromatic	Nonpolar	Neutral	2.8	165.192	3.87
P	Cyclic	Nonpolar	Neutral	-1.6	115.132	5.02
S	Hydroxylic	Polar	Neutral	-0.8	105.093	7.14
T	Hydroxylic	Polar	Neutral	-0.7	119.119	5.53
W	Aromatic	Nonpolar	Neutral	-0.9	204.228	1.25
Y	Aromatic	Bronsted acid	Neutral	-1.3	181.191	2.91
V	Aliphatic	Nonpolar	Neutral	4.2	117.148	6.73
X	xxx	xxx	xxx	0	0	0

Tabulka 1.2: Vlastnosti Aminokyselin [2]

Tabulka vlastností pro jednotlivé aminokyseliny

Některé aminokyseliny mohou patřit do více tříd, proto byla vždy vybrána co nejkonkrétnější třída. Poslední řádek v tabulce reprezentuje neznámou aminokyselinu. Při zkoumání proteinů nemusejí být některé aminokyseliny určeny.

Peptidická vazba

Peptidická vazba je druh vazby přes kterou se mohou aminokyseliny spojit. Vzniká kondenzací karboxylové skupiny jedné aminokyseliny a aminové skupiny druhé aminokyseliny. Pomocí těchto vazeb vzniká polypeptidický řetězec. Jednomu a více takovýchto spojených řetězců říkáme protein.

1.2 Proteiny

Řetězci aminokyselin propojených peptidickou vazbou se říká protein. Délka řetězce se liší ale je omezená shora hranicí danou poklesem stability proteinu s jeho rostoucí délkou. Nejdelší známý protein je Titin, který je tvořen 33000 aminokyselinami. [11] Kvůli vysoké maximální délce proteinů a množství proteinogenních aminokyselin je počet možných řetězců obrovský. Tento velký počet druhů proteinů jim umožňuje plnit řadu funkcí.

Mohou působit jako hormony, toxiny, enzymy transportéry, regulátory, protilátky, motory. Mohou plnit strukturní funkci a zprostředkovat interakci mezi buňkami.

1.2.1 Struktura proteinů

Každý protein je uspořádán do 3D struktury, která částečně určuje jeho funkci v daném biologickém procesu. Pochopení a popsání proteinové struktury je klíčové k tomu abychom pochopili jak proteiny fungují. Strukturu proteinu popisujeme na 3 úrovních, **1D**, **2D** a **3D**.

Primární Struktura

Primární strukturou se rozumí sekvence aminokyselin tvořících daný protein. Je to sekvence znaků o délce rovné počtu aminokyselin v proteinu, nad abecedou 23 znaků. 22 znaků znázorňuje aminokyseliny a 23. znak je X, který označuje neurčenou aminokyselinu. Přestože primární struktura nepopisuje tvar molekuly v prostoru, ale podle tak zvaného „Anfinsenova dogma"[12], jednoznačně určuje vyšší struktury proteinu. Avšak prostředí a události v něm jako je například interakce s ligandem, může skutečnou trojrozměrnou strukturu změnit.

Primární strukturu potřebujeme znát i k určení vyšších struktur, proto je primární struktura známá u největšího počtu proteinů. Pro představu ke dni 22.2.2023 bylo v databázi UniProt zaznamenáno 245 871 724 sekvencí [13].

Formátem popisujícím 1D strukturu je FASTA. V tomto formátu bylo přidáno k 23 znaků z výše zmíněné abecedy ještě několik znaků, které slouží k popsáni nepřesně určených částí sekvence. Více popsaný je tento formát v kapitole o datech.

Sekundární Struktura

Sekundární struktura proteinu popisuje lokální uspořádání sousedních aminokyselin a často se znázorňuje ve dvou rozměrech pomocí topologického diagramu Mezi nejčastější sekundární struktury patří $\alpha - helix$ a $\beta - sheet$,které můžeme vidět na obrázku 1.2. [14]

Terciární Struktura

Popisuje přesnou prostorovou koordinaci prvků sekvence a umístění všech funkčních skupin aminokyselin. Z terciární struktury jdou vypozorovat domény, záhyby a motivy.

Doména je oblast proteinu, která je stabilní, nezávislá na zbytku proteinu a zachovává svůj tvar. Mnoho proteinů se skládá z jedné a více domén, jedna doména se může vyskytovat ve více proteinech. Domény často tvoří funkční celky, ve kterých se nachází vazebná místa. Příkladem takovéto domény je vápník vázající doména v proteinu Calmodulin [15].

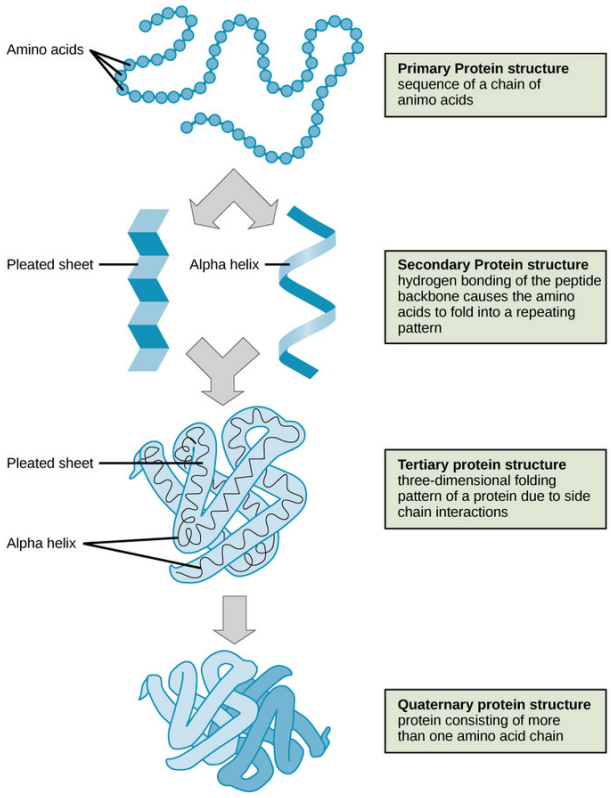
Většina proteinů není jen dlouhá molekula, ale je v prostoru několikrát přehnutá, v těchto záhybech můžeme také najít vazebná místa. Ještě menším zajímavým místy na proteinu jsou motivy které můžou právě tvořit záhyb. [16]

Jak záhyby, tak motivy a domény jsou místa, která jsou často spjata s nějakou funkcí proteinu a v průběhu evoluce zůstávají zachovány. Pokud se snažíme detekovat vazebná místa, pak je užitečné uvažovat existenci těchto míst. [16]

V Protein Data Bank, což je celosvětový archiv experimentálně určených 3D proteinových struktur. Ke dni 28.4.2023 bylo zaznamenáno 204 104 experimentálně určených struktur. [17]

Kvartérní struktura

Většina proteinů s molekulární hmotností $> 100kD$ se skládají z více jak jednoho polypeptidického řetězce to, jak jsou tyto řetězce rozmístěny popisuje kvartérní struktura. Pokud má protein více řetězců, pak se zpravidla označují velkými písmeny.



Obrázek 1.2: Struktura proteinu
cit. 2023-04-15, Dostupné z:https://med.libretexts.org/Bookshelves/Anatomy_and_Physiology/Anatomy_and_Physiology_%28Boundless%29/23%3A_Nutrition_and_Metabolism/23.4%3A_Proteins/23.4C%3A_Protein_Structure

1.2.2 Interakce mezi proteinem a ligandem

Ligandem v tomto kontextu rozumíme molekulu, která se váže na protein. Ligandy mohou být inhibitory, aktivátory, lipidová signální molekula a neurotransmiter. K proteinu se váže pomocí slabých vazeb (příkladem těchto vazeb jsou: vodíkové můstky, iontová vazba, Van der Waalsovy síly, hydrofobický efekt). Protein při těchto interakcích zastává roli receptoru, transportéru, nebo zprostředkovává nějakou biologickou funkci, jako je například kontrakce svalu.

Ne každý ligand se může navázat na dané vazebné místo. To jak je ligand pro dané místo vhodný rozlišuje vazebná specifičnost. Tuto vlastnost ovlivňuje jak tvar vazebného místa na proteinu, tak tvar molekuly představující ligand. Jelikož vazebná specifičnost je závislá i na 3D(terciární) struktuře molekuly, tak by nám její znalost v detekci protein-ligand vazebných míst mohla pomoct. Skutečně existují nástroje, které tuto vlastnost využívají, jako například P2Rank [18].

Ačkoliv tyto nástroje mají předpoklad k tomu být lepší, tak mít nástroj na předpověď ze sekvence se vyplatí, protože 1D struktura je známá u mnohonásobně více proteinů.

1.3 Formáty popisující protein

Tato sekce popisuje formáty určené k reprezentaci bílkovin v programu.

1.3.1 FASTA

Formát FASTA je textový formát používaný k reprezentaci proteinové sekvence (popisuje pouze 1D strukturu), nebo sekvenci nukleotidů. V této práci se sekvencí nukleotidů nepracujeme, proto popíšeme jen reprezentaci bílkovin.

Fasta nabízí dva typy komentářů. Prvním je symbol „>“ za kterým obvykle následuje identifikátor bílkoviny. Druhým je středník „;“, po kterém se vše do konce řádku ignoruje. Jelikož jediným používaným typem komentáře bylo „>“, tak se stalo praxí používat pouze „>“.

Ukázka definice záhlaví:

```
>sp|P12345|PROT_HUMAN Bílkovina ABC
```

Jak můžeme vidět na ukázce výše, tak pokud je více identifikátorů, tak se oddělují "|".

Poté následuje definice sekvence, kde každá aminokyselina je reprezentována jednopísmenným kódem stejným jako v tabulce 1.1.

Ukázka definice sekvence:

```
MELSLVLFLLCLPLALGASDPNIPADPNDE
```

Protože v nějakých případech není daná aminokyselina přesně určena, tak je těchto 22 znaků doplněno o následující znaky, které zastupují více aminokyselin naráz, nebo plní určitou funkci. Tyto znaky jsou v tabulce 1.3.

Znak	Význam	Možné aminokyseliny
B	kys. asparová,Asparagin	D,N
J	Leucin, Isoleucine	L,I
Z	Kyselina glutamová,Glutamin	E,Q
*	konec sekvence	
-	mezera nerozlišitelné délky	

Tabulka 1.3: Doplňující znaky ve FASTA[2]

V jedné složce může být zaznamenáno více sekvencí. Příkladem takové složky může být.

```
>sp|P12345|PROT_HUMAN protein ABC [Homo sapiens]
MELSLVLFLLCLPLALGASDPNIPADPNDE...
>sp|Q98765|PROT_MOUSE protein  XYZ [Mus musculus]
MDSLVELFLLCLPLALGASDQWERADFD...
```

1.3.2 PBD Formát

Tento formát je používaný pro reprezentaci 3D struktury proteinu. Pro každý atom zde jsou souřadnice, typ, číslo v pořadí ,typ aminokyseliny a řetězce. Podrobná dokumentaci je k nalezení na stránkách Protein Data Bank [17].

2. Informatický kontext

Po představení biologické části problémů, představíme informatické pozadí. V této práci budeme porovnávat metody strojového učení. Začneme tedy jeho definicí z knihy Machine Learning od Toma Mitchella. [19]

„*Cílem strojového učení je vytvořit program který se učí ze zkušenosti E vzhledem k určitému úkolu T a metriky P, přičemž se jeho výkon při konání úkolu T měřeného P zlepšuje s rostoucí E.*“

V našem problému, je úkolem T najít protein-ligand vazebná rezidua v sekvenci aminokyselin. V řeči strojového učení je to binární klasifikace, která rozřadí jednotlivé aminokyseliny na ty, která jsou součástí vazebného místa a na ty co ne. P je zvolená metrika v našem případě bude hlavní metrikou MCC (Matthew's correlation coefficient). Důvod proč jsme zvolili tuto metriku a její popis, je uveden v sekci 2.1. Všechny testované metody spadají pod kategorii metod strojového učení s učitelem. Naší znalostí E tedy jsou sekvence u kterých jsou již experimentálními metodami určená vazebná místa. Reziduím tvořícím vazebné místo přiřadíme **1**, ostatním **0**.

Vstupem programu je vektor čísel (atributů) reprezentujících reziduum a výstupem bude třída do jaké dané reziduum patří. To jak přiřadit ke každému reziduu konkrétní vektor je popsáno v metodologii v sekci 3.1. Jednomu takovému vektoru říkáme vzorek.

2.1 Výběr metriky

K měření výkonu modelu a k porovnání modelů mezi sebou potřebujeme metriku. Jelikož řešíme binární klasifikaci můžeme rozdělit výsledky klasifikace do 4 kategorii.

- **TP** - pravdivě pozitivní
- **TN** - pravdivě negativní
- **FP** - falešně pozitivní
- **FN** - falešně negativní

Nyní si představíme několik vzorců pro metriky.

- **Accuracy** - měří kolik případů bylo správně klasifikováno. Je dobrou metrikou pro vyvážené datasety.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

- **Precision** - měří kolik z detekovaných vazebných reziduí je klasifikováno správně.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- **Recall** - měří kolik procent vazebných reziduí bylo detekováno.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- **f1-score** - může být použita pro maximalizaci precision a recall zároveň.

$$\text{f1-score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2 \times \text{TP}}{2 \times \text{TP} + \text{FP} + \text{FN}}$$

- **MCC**(Matthews correlation coefficient) - stejně jako *f1-score* je vhodný pro maximalizaci precision a recall zaroveň. Oproti *f1-score* má dvě výhody. Není závislý na prohození tříd a uvažuje i případy, které jsou **TN**. Tím pádem *f1-score* může v některých případech dávat příliš optimistické výsledky na rozdíl od *MCC*.

  Obor hodnot je $h \in \langle -1,1 \rangle$, kde 1 odpovídá perfektní predikci a -1 opačné predikci.

$$\text{MCC} = \frac{\text{TP} \times \text{TN} - \text{FP} \times \text{FN}}{\sqrt{(\text{TP} + \text{FP}) \times (\text{TP} + \text{FN}) \times (\text{TN} + \text{FP}) \times (\text{TN} + \text{FN})}}$$

Jelikož vazebních míst, je jen malé procento, tak není vhodné použít accuracy. F1-score se sice často používá při nevyvážených datasetech ale občas může dát příliš optimistické výsledky. Z tohoto důvodu budeme používat právě MCC, které se ukazuje jako spolehlivější metrika pro nevybalancované datasety [20].

2.2 P2Rank

P2Rank je jeden z nejmodernějších nástrojů založeným na metodách strojového učení, který předpovídá výskyt protein-ligand vazebných míst z 3D struktury a který překonal mnoho doposud známých nástrojů. Z tohoto důvodu budeme srovnávat námi vytvořené modely právě s tímto nástrojem. Výhodou srovnání s P2Rank je, že jej lze použít jako zástupce metod využívajících znalostí 3D struktury proteinů. Základní metodou použitou v P2Rank je Random Forest klasifikátor, díky kterému je nejrychlejším takovýmto nástrojem[18].

Predikci P2Rank můžeme rozdělit do následujících kroků:

1. Vstupem je 3D struktura proteinu. Následně se vygeneruje množina pravidelně rozmístěných bodů na povrchu zadané struktury proteinu.

2. Vygeneruje se vektor atributů pro každý bod na povrchu 3D sekvence, v závislosti na jeho fyzicko-chemických vlastností. Poté se k atributům daného bodu přidají i atributy bodů z jeho okolí.

3. Predikce skóre ligandability těchto bodů pomocí RF klasifikátoru.

4. Filtrace bodů s nízkým skóre ligandability, a odstranění osamocených bodů. Jako vazebná místa byly klasifikovány pouze ty, které měli v okolí alespoň 3 body s vysokým skóre.

5. Ohodnocení predikovaných míst pomocí kumulativního skóre ligandability.

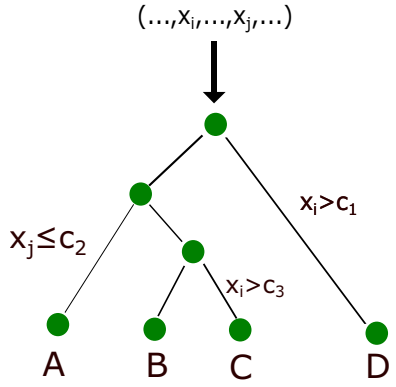
P2Rank nabízí možnost vlastního natrénování modelu. Jeho základní model byl natrénován na datasetu jménem CHEN11, který byl použit i pro trénování některých modelů v této práci. P2Rank je dostupný online z:
`https://github.com/cusbg/p2rank-framework`.[21]

2.3 Vybrané metody

Byly vybrány 3 metody pro porovnání a to **Random Forests**[22], **Gradient Boosting Decision Trees** [23] a **neuronova síť** [24].Metoda **Random Forests** byla vybrána, protože je použita v P2Rank [18]. Ačkoliv je P2Rank mnohem komplexnější nástroj než **Random Forest Classifier**, je tato metoda jeho významnou součástí a jeho úspěšnost nasvědčuje tomu že by mohla být vhodnou pro předpověď ze sekvence. Jako další vybranou metodou je **Gradient Boosting Decision Trees**, která byla vybrána kvůli její podobnosti s předchozí metodou. Má tedy také předpoklady k tomu být vhodnou metodou. Poslední zvolenou metodou je **neuronová síť**, která byla zvolena pro svou všestrannost. Tato metoda se prokazuje jako účinná při řešení mnoha různých problémů a jsou na ni založeny i další pokročilejší metody strojového učení, jako jsou například hluboké neuronové sítě.

2.3.1 Decision Tree Clasifier

Na tomto principu jsou založené následující dvě metody, proto první popíšeme tuto metodu. Tato metoda je formou strojového učení s učitelem. Základem je rozhodovací strom, kde listy reprezentují třídy, v našem případě máme pouze dvě. Hrany rozdělují jednotlivé vzorky do množin podle hodnot atributů daných vzorků. Množinám vzorků v listech je přiřazená třída, která má v daném listu nejvíce zástupců.



Obrázek 2.1: **Rozhodovací strom**

Na obrázku 2.1 jsou c_1, c_2, c_3 konstanty a A, B, C, D jsou množiny do kterých se jednotlivé vzorky rozdělují.

Konstrukce stromu

Strom začínáme stavět od kořene. Označme I_τ množinu vzorků patřící vrcholu τ. K rozdělování používáme **kritérium** c_τ, které říká jak moc uniformní je rozdělení vzorků v τ. Cílem je najít atributy a jejich hodnoty takové, že když rozdělíme vrchol τ na τ_L a τ_R, pak uniformnost nového rozdělení $c_{\tau_L} + c_{\tau_R}$ vrcholu τ , je menší než toho předchozího (uniformnost toho předchozího rozdělení je c_τ). Takovýchto rozdělení můžeme najít více a my se snažíme najít to, které sníží uniformnost co nejvíce. Jinými slovy to kdy

$$c_{\tau_L} + c_{\tau_R} - c_\tau$$

je nejmenší.

Při tomto můžeme dodržovat omezující podmínky, jako je maximální hloubka, minimální počet vzorků v listu nebo maximální počet listů.

Počítání kritéria

Nechť průměrná pravděpodobnost třídy k ve vrcholu τ je $p_\tau(k)$ a I_τ je množina vzorků náležící vrcholu τ. Pak definujme následující dvě kritéria.

- **Gini index** - měří jak často bude náhodně zvolený prvek nesprávně přiřazen třídě.

$$c_{gini}(\tau) = |I_\tau| \sum_k p_\tau(k)(1 - p_\tau(k))$$

- **Entropy** -měří míru entropie.

$$c_{entropy(\tau)} = |I_\tau| * H(p_\tau) = -|I_\tau| \sum_{\substack{k \\ p_\tau(k) \neq 0}} p_\tau(k) \log p_\tau(k)$$

2.3.2 Random forests

Tato metoda natrénuje množinu stromů. Při vyhodnocování nechá vyhodnotit vzorek na všech zároveň a poté hlasováním zvolí do jaké třídy bude zařazen. Je třeba ale zamezit trénování stejných stromů k tomu slouží **Bagging**. Bagging je trénovací algoritmus, kdy se z trénovacího datasetu opakovaně vybírá náhodná podmnožina s opakováním a na každé z těchto podmnožin se natrénuje rozhodovací strom. Pro zajištění aby spolu jednotlivé stromy nekorelovali se ještě v každém vrcholu stromu volí množina atributů. Jen tyto atributy se použijí jako kandidáti na rozdělení.Tato metoda je méně citlivá na šum v datech, protože se výsledek průměruje mezi mnoho stromů.

2.3.3 Gradient boosting decision trees

Tato metoda také trénuje množinu stromů ale trénování jednotlivých stromů je na sobě závislé. Stromy jsou trénovány sekvenčně, tak aby opravily chyby předchozích natrénovaných stromů. Popíšeme zde pouze binární klasifikaci pomocí GBDT.

Pokud označíme y_t jako funkci predikce t-tého stromu, pak predikce celé množiny je:

$$\sigma(y(x_i)) = \sigma(\sum_{t=1}^{T} y_t(x_i,w_t)),$$

kde w_t jsou parametry reprezentující t-tý strom, x_i je vzorek a σ je funkce, daná předpisem:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

.

Jelikož popisujeme pouze binární klasifikaci pak můžeme rozdělit třídy na pozitivní a negativní. Funkce y je logaritmus podílu mezi pozitivní a negativní třídou. Předpokládejme, že v listu, pod který spadá vzorek x_i, je A množina vzorků pozitivní třídy a B je množina vzorků negativní třídy. Pak když do funkce y dosadíme vzorek x_i ,tak $y_t(x_i,w_t) = log(\frac{|A|}{|B|})$

Vzorec predikce, můžeme chápat tak že funkce y přiřadí každému vzorku číslo říkající jak moc podle daného stromu patří do dané třídy. Funkce σ následně převede tyto čísla na pravděpodobnost.

Funkce určující chybu pro vzorek x_i definujeme takto:

$$l(t_i,y(x_i,w)) = -log[\sigma(y(x_i,w))^t(1-\sigma(y(x_i,w)))^{1-t}]$$

,

kde t_i je target vzorku i, tedy jeho skutečná třída.

Definice celkové chybové funkce v t-tém kroku je:

$$E^t(\omega_t, \omega_{1..t-1}) = \sum_i [l(t_i,y_{t-1}(x_i,\omega_{1..t-1}) + y_t(x_i,\omega_t))] + \frac{1}{2}\lambda||\omega_t||^2$$

.

kde

- ω_t jsou parametry t-tého stromu
- $\omega_{1..t-1}$ jsou parametry všech předchozích stromů
- $y_t(x_i,\omega_t)$ je funkce predikce posledního stromu
- $y_{t-1}(x_i,\omega_{1..t-1})$ je funkce predikce všech předchozích stromů
- t_i je skutečná třída vzorku i, $t = 0/1$
- l je chybová funkce pro jeden vzorek
- λ je síla $L^2 - regularizace$

Minimalizace ztrátové funkce probíhá v iteracích. V každé iteraci se vytvoří nový strom, pomocí rozdělovacího kritéria, které minimalizuje ztrátovou funkci. Tento nový strom se zařadí mezi předchozí stromy a stane se součástí predikční funkce celého lesa. Více detailní popis je zde. [25]

2.3.4 Dopředná neuronová síť

Dopředná neuronová sít, je poslední testovanou metodou. Jejím základem jsou neurony.

Neuron je funkce f

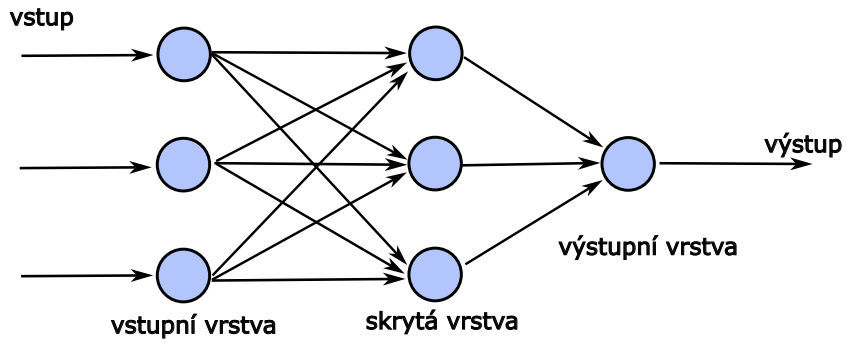
$$f(x) = g(\sum_{i=1}^{n} w_i * x_i + b)$$

,

kde

- x je vstupní vektor.
- $w_1 ... w_n$ jsou váhy.
- b je bias , může být považován jako váha w_{n+1} u které je $x_{n+1} = 1$
- g je aktivační funkce.

Dopřednou neuronovou síť, pak můžeme popsat jako orientovaný acyklický graf, kde hrany reprezentují váhy a vrcholy neurony. Takovouto sít dělíme na vrstvy, kde první vrstva je vstupní, poté následuje několik skrytých vrstev až k poslední vrstvě, která je výstupní. Obecná architektura sítě je zobrazena na obrázku 2.2.



Obrázek 2.2: Dopředná neuronová síť
cit. 2023-04-15, Dostupné z:
https://cs.m.wikipedia.org/wiki/Soubor:MultiLayerNeuralNetworkBigger_english.png

Výpočet sítě probíhá sekvenčně, kdy výstupy z předchozí vrstvy se stanou vstupy pro následující vrstvu a v každém neuronu se aplikuje jeho aktivační funkce. Jako aktivační funkce se používá několik funkcí. Většina těchto aktivačních funkcí je nelineárních. Právě nelinearita aktivačních funkcí umožňuje řešit netriviální problémy, ke kterým je třeba aby síť aproximovala nelineární funkci. Příklady aktivačních funkcí jsou:

- **Rectified Linear Unit (ReLU)**-typicky používaná funkce ve skrytých vrstvách.

$$ReLU(x) = max(0,x)$$

- **Hardswish** - tato funkce připomíná ReLU, při testování byla zvolena jako aktivační funkce ve skrytých vrstvách.

$$\begin{cases} 0 & x \leq -3 \\ x & x \geq 3 \\ x * (x + 3/6) & jinak \end{cases}$$

- **Sigmoid** - používaná pro binární klasifikaci ve výstupní vrstvě. Slouží pro převedení čísel na pravděpodobnosti.

$$\sigma(x) = \frac{1}{1 + e^{-z}}$$

Při testování bylo k dispozici více funkcí jejich seznam je k nalezení na stránkách dokumentace ke knihovně PyTorch[26], seznam vyzkoušených aktivačních funkcí je v sekci 3.4. K tomu aby byl model použitelný, musíme nastavit váhy. Procesu nastavování vah se říká učení. Jsou to právě váhy společně s aktivačními funkcemi, které dovolují modelovat komplexní vztahy mezi vstupem a výstupem.

Ztrátová funkce a zpětná propagace

Váhy se model učí iterativně. Postupně se upravují, tak aby byla minimalizovaná ztrátová funkce. Příkladem používaných ztrátových funkcí je:

- **Cross-Entropy loss**

$$CEL(t,y) = -\sum_{i=1}^{m} t_i log(y_i)$$

,

kde m je počet případů

t_i je skutečná třída vzorku i, $t_i = 0/1$.

y_i je předpověděná pravděpodobnost třídy vzorku i $y_i \in \langle 0,1 \rangle$.

Používáme ji jako ztrátovou funkci v našich modelech.

- **Mean squared error (MSE)**

$$MSEL(t,y) = -\sum_{i=1}^{m} \frac{1}{m}(t_i - y_i)^2)$$

.

Pokud chceme přiřadit váhu jednotlivým třídám, tak můžeme použít vážené verze těchto funkcí, nebo můžeme v poměru podle váhy třídy vybírat vzorky s opakováním. Pro model to znamená, že bude lépe natrénován na třídu s větší vahou protože častěji bude dostávat vzorky patřící dané třídě. Pro naše účely jsme zvolili způsob, kdy vybíráme vzorky s opakováním.

Protože neuronová síť není lineární, tak ztrátová funkce není konvexní a je těžké ji minimalizovat. Z tohoto důvodu se používá pro iterativní algoritmus zvaný gradient-descent [27], což je optimalizační algoritmus, který iterativně hledá

lokální minimum diferencovatelné funkce. Jelikož je síť dopředná pak výpočet gradientu je jednoduchý. Stačí uplatnit řetízkové pravidlo derivací v opačném směru ke směru výpočtu. Tomuto postupu se říká zpětná propagace chyby [28]. Obecný předpis pro aktualizaci vah v iteraci i je:

$$w_i \leftarrow w_{i-1} - \alpha \nabla E(w_{i-1}),$$

kde α je konstanta zvaná learning rate.
∇E je gradient ztrátové funkce vzhledem k vahám.

3. Metodologie

3.1 Reprezentace reziduí

Při předpovídání, zda určité reziduum v určité sekvenci je součástí vazebného místa, potřebujeme nějak toto reziduum reprezentovat. Byly zvoleny dva typy reprezentace, které budeme mezi sebou porovnávat. První typ využívá vlastností dané aminokyseliny a druhý vytvoří reprezentaci pomocí jazykových modelů. Na protein v tomto kontextu pohlížíme jako na řetězec reziduí. Avšak protein může být složen z několika takových řetězců. Pokud je tomu tak, tak jej rozdělíme na jednotlivé řetězce, které pojmenujeme velkými písmeny.

3.1.1 Vlastnosti aminokyselin

V tomto případu využíváme vlastností aminokyselin, které jsou uvedeny v biologické části. Proto abychom zahrnuli do reprezentace i strukturu proteinu, tak budeme uvažovat i okolí daného rezidua v sekvenci. Toto okolí je reprezentováno jeho poloměrem r. Tedy jedno reziduum v sekvenci popíšeme pomocí $2r + 1$ sousedních reziduí a jejich vlastností. Příkladem pokud chceme popsat v pořadí n reziduum ze sekvence

$$L_1...P_{n-r}...L_{n-1}A_nL_{n+1}...G_{n+r}A...E,$$

pak okolí je

$$[P_{n-r}...L_{n-1}A_nL_{n+1}...G_{n+r}].$$

Vektor reprezentující dané reziduum získáme substitucí jednotlivých reziduí vektory popisující vlastnosti dané aminokyseliny, z které reziduum vzniklo. Tomuto vektoru říkáme vzorek.

Tabulka vlastností 3.1 je použita pro převod vektoru reziduí na vektor popisující dané reziduum a jeho okolí. Vychází z tabulky 1.2. Tato tabulka měla být původně doplněna o doplňující znaky používané ve formátu FASTA viz tabulka 1.3, avšak při prozkoumání používaných datasetů tyto znaky nebyly nalezeny a tudíž je nemá smysl reprezentovat.

Chybějící hodnoty

V tambulce 3.1 u symbolů *U,O,X* jsou určité atributy neznámé, nebo nemohou být přesně určeny. Chybějící hodnoty můžou mít negativní vliv na predikci. Avšak tyto symboly se v sekvencí vyskytují ojediněle, proto předpokládáme, že tento negativní vliv nebude pozorovatelný. Příkladem v největším použitém datasetu **HOLO4K** s 4543 sekvencemi se vyskytují pouze 2 symboly *U,O* a 37 symbolů *X*. Pro reprezentaci neznámých kategorických atributů (kód, třída, polarita, náboj) je zaveden nový symbol „—“. Za neznámé nekategorické atributy (hydropatický index, hmotnost, četnost) je dosazena 0.

Kódování atributů

První čtyři atributy (kód, třída, polarita, náboj) jsou kategorické. Pro tyto typy atributů budeme potřebovat zvolit jejich kódování. Jsou dva způsoby, jakými

Kód	Třída	Polarita	Náboj	Hydr.ind.	Hmotnost	Četnost
A	Aliphatic	Nonpolar	Neutral	1.8	89.094	8.76
R	Fixed cation	Basic polar	Positive	-4.5	174.203	5.78
N	Amide	Polar	Neutral	-3.5	132.119	3.93
D	Anion	Bronsted base	Negative	-3.5	133.104	5.49
C	Thiol	Bronsted acid	Neutral	2.5	121.154	1.38
E	Anion	Bronsted base	Negative	-3.5	147.131	6.32
Q	Amide	Polar	Neutral	-3.5	146.146	3.9
G	Aliphatic	Nonpolar	Neutral	-0.4	75.067	7.03
H	Aromatic cation	Bronsted acid and base	Neutral	-3.2	155	2.26
I	Aliphatic	Nonpolar	Neutral	4.5	131.175	5.49
L	Aliphatic	Nonpolar	Neutral	3.8	131.175	9.68
K	Cation	Bronsted acid	Positive	-3.9	146.189	5.19
M	Thioether	Nonpolar	Neutral	1.9	149.208	2.32
F	Aromatic	Nonpolar	Neutral	2.8	165.192	3.87
P	Cyclic	Nonpolar	Neutral	-1.6	115.132	5.02
S	Hydroxylic	Polar	Neutral	-0.8	105.093	7.14
T	Hydroxylic	Polar	Neutral	-0.7	119.119	5.53
W	Aromatic	Nonpolar	Neutral	-0.9	204.228	1.25
Y	Aromatic	Bronsted acid	Neutral	-1.3	181.191	2.91
V	Aliphatic	Nonpolar	Neutral	4.2	117.148	6.73
U	Amide	—	—	0	168.064	0
O	Amide	—	—	0	255.313	0
X	—	—	—	0	0	0

Tabulka 3.1: Vlastnosti Aminokyselin pro získání atributů

můžeme tyto atributy zakódovat. Prví je ordinální, které každému atributu přiřadí jedno číslo podle kategorie do které patří. Druhým způsobem je one-hot kódování, které reprezentuje každý atribut vektorem, jehož velikost je stejná jako počet kategorií. V tomto vektoru jsou samé nuly až na jednu jedničku na místě kategorie do které vzorek patří .

Není jasné, která z těchto reprezentací bude lepší. Očekáváme, že pro metody založených na rozhodovacích stromech bude lepší ordinální reprezentace. Protože tyto metody rozdělují vzorky na základě jedné hodnoty a u one-hot kódování je tento atribut reprezentován více hodnotami. Předpokládáme že algoritmus z tohoto důvodu bude mít tendenci tyto hodnoty ignorovat, protože rozdělení v tomto případu oddělí pouze malou část vzorků a tím sníží celkovou uniformnost stromu jen o málo. Na druhou stranu očekáváme, že one-hot kódování bude vhodnější pro neuronovou síť, protože v tomto případu hodnota u atributu neodpovídá důležitosti atributu. Oba přístupy jsou porovnané v sekci 4.1.2.

3.1.2 Jazykové modely (embeddingy)

K získání reprezentace jednotlivých reziduí můžeme také využít technik ze *zpracování přirozeného jazyka* [29]. V tomto kontextu představují jednotlivé rezidua slova a sekvence větu. Můžeme tedy na hledání reprezentace daného rezidua v sekvenci nahlížet, jako na hledání reprezentace slova ve větě. Tedy můžeme využít technik a znalostí, používaných pravě při *zpracování přirozeného jazyka*. V posledních letech přispěli k rozvoji tohoto oboru jazykové modely, díky své schopnosti reprezentovat slova a věty způsobem, který dovoluje zachytit vztahy mezi nimi.

Jazykový model je druh hluboké neuronové sítě natrénovaný na rozsáhlém kor-

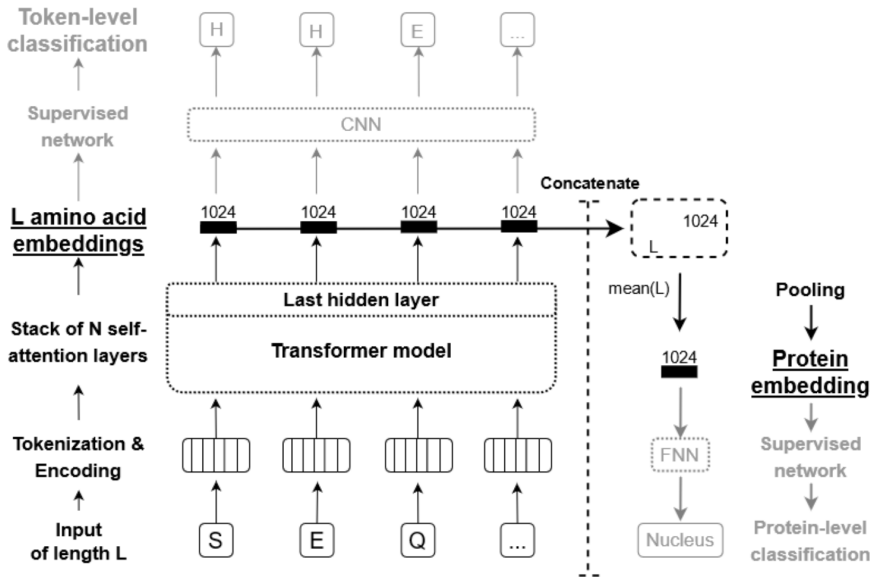
pusu[1], za cílem naučení vektorové reprezentace jednotlivých slov. Těmto vektorům říkáme **embeddingy**. Trénování takovéhoto modelu může probíhat formou, kdy vstupem modelu je věta s zamaskovanými slovy, které má model za úkol předpovědět. Tímto způsobem se model naučí vytvářet embeddingy reprezentující kontext slov ve větě. Samotné embedingy poté mohou být získány z poslední skryté vrstvy modelu jako její výstupy. Jelikož existují rozsáhlé databáze proteinových sekvencí tak je možné převést tento koncept i na proteiny. Je známých několik technik jako například ProtVec [30], SeqVec[31] nebo ProtTrans[32] [33].

Konkrétně v projektu ProtTrans byly vytvořeny embeddery jako ProtTransBERT, ProtTransXLNet nebo ProtTransT5, které dokážou vygenerovat embeddingy pro danou sekvenci. V této práci budeme používat právě tyto tři modely pro generování embeddingů. Tyto modely byly natrénovány na datasetech BFD, UniRef50 a UniRef100.[34]. Jsou součástí knihovny *Bio Embeddings* [35]. Jejich porovnání můžeme vidět v tabulce 3.2.

Embeddingy	**ProtBert**	**ProtXLNet**	**ProtT5**
Trenovací datasety	BFD	UniRef100	BFD, UniRef50
GPU pamět(GB)	2.8	2.7	5.9
Dimenze výstupu	1024	1024	1024

Tabulka 3.2: Porovnání modelů použitých pro získání embeddingů.

ProtTransBERT a ProtTransXLNet se od sebe liší svou architekturou. Zatímco třetí model ProtTransT5 se od těchto modelů odlišuje také svou složitější strukturou, která se projevila na jeho velikosti. Obecnou architekturu můžeme vidět na obrázku 3.1.



Obrázek 3.1: Obecná architektura ProtTrans modelu.
cit. 2023-04-15, Získáno z ProtTrans: Towards Cracking the Language of Life's Code Through Self-Supervised Learning [34]

[1]Souboru textů v elektronické formě ve formátu umožňujícím vyhledávaní jazykových jevů.

Embeddingy se počítají z celé sekvence ve formátu FASTA. Výstupem tedy je 2D matice reálných čísel o rozměrech $N \times D$, kde N je délka sekvence a D dimenze embeddingu. Jedno rezidum je tedy charakterizováno vektorem o délce D, kterému říkáme vzorek. Jelikož se embeddingy počítají z celé sekvence, pak nesou i informaci o okolí daného rezidua v sekvenci.

3.2 Datasety a jejich příprava

V této práci používáme datasety ze dvou zdrojů, které se liší přístupem k označení vazebních míst. Tím prvním jsou datasety používané k trénování a testování P2Rank, o kterém je sekce 2.2. Tato data jsou volně dostupná na gitovém repozitáři, uvedeném na stránkach P2Rank a hrála roli při trénování jeho základního modelu. [36]. Najdeme zde několik druhů datasetů, v této práci zmíníme tyto:

CHEN11[37] - byl použit k natrénování základního modelu v P2Rank. Budeme jej používat také jako trénovací množinu, a to abychom mohli porovnat jednotlivé metody s P2Rank.

COACH420[38] - budeme ho používat jako testovací dataset, protože byl použit i jako testovací dataset u P2Rank, a je disjunktní s *CHEN11*.

HOLO4K[39] - je největší dataset obsahující 4543 proteinů a je disjunktní s *CHEN11*. Byl použit jako testovací množina při testování základního modelu P2Rank.

Bližší informace jsou v tabulce 3.3.

Dataset	Sekvencí	V (num) / N (num)	V (%)
CHEN11	251	4 779 / 55 623	7.9
COACH420	420	5 781 / 108 885	5.0
HOLO4K	4543	59 929 / 977 835	5.8

Tabulka 3.3: **Porovnání datasetů z P2Rank**,**numV* je počet vazebných aminokyselin, *numN* je počet nevazebných aminokyselin a *V (%)* je procentuální podíl vazebných aminokyselin.

Druhým zdrojem dat jsou datasety rozdělené podle typu ligandů od D.J. Yu [40] . Je zde dvanáct datasetů, přičemž každý je zaměřený na jiný ligand. V těchto datasetech jsou označena pouze místa vázající daný ligand. Nachází se zde 5 datasetů pro 5 nukleotidů (AMP, ADP, ATP, GTP, GDP), 5 iontů (CA, MG, MN, FE, ZN), DNA a HEME. Každý z těchto datasetů už má předem určené rozdělení na trénovací a testovací část. My budeme toto rozdělení respektovat. Bližší informace jsou v tabulce 3.4

Používáme tedy datasety ze dvou zdrojů, jejich shrnutí a důvod použití je v tabulce 3.5.

Ligand	Trénovací dataset			Testovací dataset		
	Sekvencí	NumV/NumN	V (%)	Sekvencí	NumV/NumN	V (%)
ATP	221	3021/72334	4.0	50	647,16639	3.7
ADP	296	3833/98740	3.7	47	686,20327	3.2
AMP	145	1603/44401	3.4	33	392,10355	3.6
GDP	82	1101/26244	4.0	14	194,4180	4.4
GTP	54	745/21205	3.9	7	89,1868	4.5
Ca^{2+}	965	4914/287801	1.6	165	785,53779	1.4
Zn^{2+}	1168	4705/315235	1.5	176	744,47851	1.5
Mg^{2+}	1138	3860/350716	1.1	217	852,72002	1.1
Mn^{2+}	335	1496/112312	1.3	58	237,17484	1.3
Fe^{3+}	173	818/50453	1.6	26	120,9092	1.3
DNA	335	6461/71320	8.3	52	973,16225	5.6
HEME	206	4380/49768	8.1	27	580,8630	6.2

Tabulka 3.4: **Porovnání datasetů rozdělených podle ligandů**,*$NumV$ je počet vazebných aminokyselin, *NumN* je počet nevazebných aminokyselin a *V (%)* je procentuální podíl vazebných aminokyselin.

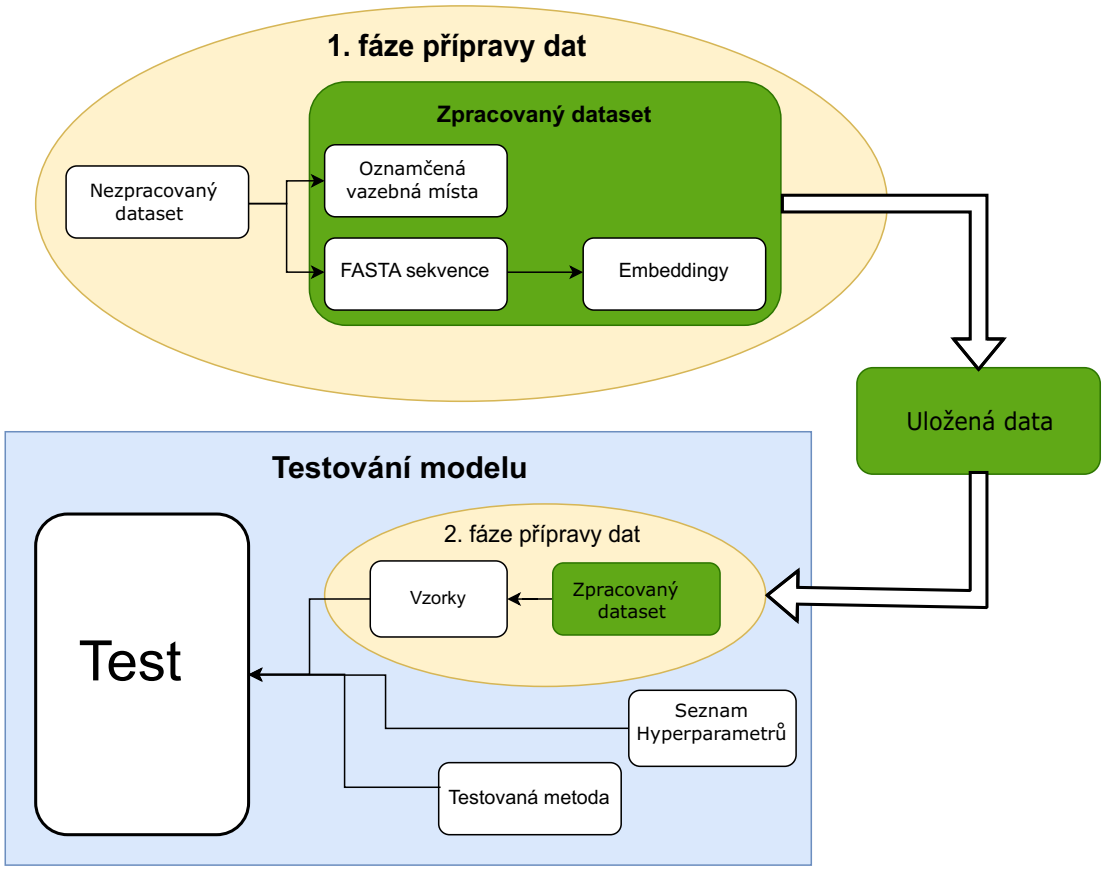
Důvody použití datasetů	
Nerozlišené datasety z P2Rank	**Datasety pro jednotlivé ligandy**
- možnost porovnat účinnost metod s P2Rank a tím pádem porovnat metody využívající predikci znalosti 1D struktury s metodou využívající znalosti 3D struktury - ukázat vliv reprezentace reziduí na účinnost zkoumaných metod -ukázat účinnost metod na těchto datasetech	- ukázat účinnost zkoumaných metod na datech rozdělených podle ligandů - možnost porovnat účinnost zkoumaných metod s jinými metodami používající tyto datasety
Rozdíly	
- procentuálně více vazebných míst - není pevně daná testovací množina, my jsme určili (COACH420)	-procentuálně méně vazebných míst -pevně daná testovací množina

Tabulka 3.5: Porovnání datasetů z odlišných zdrojů.

3.2.1 Příprava datasetů

Cílem této práce není jen porovnat metody strojového učení, ale i způsoby reprezentace reziduí. K reprezentaci reziduí používáme embeddingy, nebo znalost vlastností aminokyselin a určitého okolí daného rezidua. Konkrétní způsob reprezentace záleží na zvoleném testu. Abychom nemuseli vygenerovat a uložit všechny možné způsoby reprezentace reziduí pro všechny datasety, tak jsme rozdělili přípravu dat do dvou fází. První fáze proběhla celkově jednou. V této fázi se pro každý dataset a sekvenci v něm, připravila sekvence ve FASTA formátu,

reprezentace vazebných míst v této sekvenci a popis sekvence pomocí embeddingů. Embeddingy popisují sekvenci pomocí 2D pole o rozměrech $N \times D$, kde N je délka sekvence a D délka jednoho embeddingu popisujícího jeden prvek sekvence. Druhá fáze probíhá až při samotném testování konkrétního modelu predikujícího vazebná místa. V této druhé fázi se připraví z dat vytvořených v první fázi konkrétní vzorky pro daný test. Obě fáze jsou znázorněny v obrázku 3.2.



Obrázek 3.2: Schéma přípravy dat a testování

Na obrázku 3.2 je vyznačena první fáze přípravy dat, která probíhá před testováním modelů. Při testování se provádí test, během kterého se při zvolené reprezentaci dat a metody strojového učení natrénuje model predikující vazebná místa. Při testování modelu, jehož schéma je na obrázku v modré části se používají pouze výsledky z 1. fáze přípravy dat ale samotná 1.fáze už se při každém testu neprovádí.

Zpracování datasetů rozdělených podle ligandů v 1. fázi přípravy dat

Tyto datasety už byly ve formátu fasta a měli i označená vazebná místa. Příklad záznamu jedné sekvence je zde:

```
3QY0;A;BS01;GDP;N14 A15 G16 K17 T18 T19;FQGHMLFISATNTNAGKTTCARLLAQ...
```

Abychom byly schopni rozpoznat jednotlivé sekvence od sebe, jsou pro nás potřebné tyto údaje:

- `3QY0` je kód proteinu ze kterého sekvence pochází,
- `A` je jméno řetězce,
- `GDP` je zkratka ligandu, který se váže na označená místa v této sekvenci,
- `N14 A15 G16 K17 T18 T19` jsou označená vazebná místa, kde číslo je index aminokyseliny v sekvenci a písmeno je kód pro aminokyselinu na tomto indexu, který slouží pro kontrolu správně určeného vazebného místa,
- `FQGHMLFISATNTNAGKTTCARLLAQ...` je sekvence zapsaná ve formátu FASTA.

Jediné co zbývalo udělat v první fázi přípravy dat, je vygenerovat embeddingy. Tento postup je stejný, jak u datasetů pro ligandy, tak u datasetů získaných pomocí P2Rank, proto je popsaný ve společné sekci 3.2.1.

Zpracování datasetů pomocí P2Rank v 1. fázi přípravy dat

Datasety **HOLO4K**, **COACH420** a **CHEN11**, obsahují pdb složky, které popisují 3D strukturu sekvence aminokyselin. My potřebujeme pouze 1D reprezentaci sekvence ve formátu FASTA a potřebujeme označit vazebná místa. Pro získání této reprezentace a následně i označení vazebných míst, byl využit P2Rank. Označení vazebních míst se zde liší od způsobu označení vazebních míst v datasetech rozdělených podle ligandů. Ke každému datasetu existuje *.csv* soubor, který má v prvním sloupci jména sekvencí a ve druhém sekvence s vyznačenými vazebními místy. Příkladem jednoho záznamu je:

`a.029.003.001_1rx0a_A,0 0 0 1 0 0 0 1 0 0 1 1 0 0 1...`

Kde

- `a.029.003.001_1rx0a_A` je jméno sekvence, přidáním přípony `.fasta` z něj dostaneme jméno složky s sekvencí ve formátu FASTA,
- `0 0 0 1 0 0 0 1 0 0 1 1 0 0 1..` je sekvence označující vazebná místa, kde každá 0 a 1 odpovídají jedné aminokyselině v sekvenci.

Pokud tedy zarovnáme sekvenci nul a jedniček s sekvencí ve FASTA formátu, tak můžeme vidět, že světlezeleně vyznačené amimokyseliny, jsou ty na kterých se nachází vazebné místo.

```
0001000100110010000000000000000000000000010110001001001000000000000...
GQGFLIAVRGLNGGRINIASCSLGAAHASVILTRDHLNVRKQFGEPLASNQYLQFTLADMATR...
```

Během této fáze přípravy dat(1.fáze) proběhla i kontrola, vygenerovaných sekvencích ve FASTA formátu a sekvencí s vyznačenými vazebnými místy. Tuto kontrolu jsme provedli abychom měli jistotu, ža získaná data jsou správná. Během kontroly bylo kontrolováno:

- *Sekvence ve FASTA formátu je správně.* Data v datasetech které zpracováváme jsou uložena ve složkách v PDB formátu. Tento formát popisuje 3D strukturu proteinu, ale je možné z něj získat i popis proteinu pomocí FASTA formátu. Sekvence ve formátu FASTA získaná z PDB se následně porovnává se sekvencí ve formátu FASTA ale získanou pomocí P2Rank.

- *Délka sekvence s vyznačenými vazebními místy odpovídá délce sekvence ve formátu FASTA.* Tato kontrola slouží k tomu, aby bylo zaručeno, že u každého rezidua je určeno, zda je součástí vazebného místa, nebo ne. Zároveň slouží i jako kontrola toho, že sekvence s vyznačenými vazebnými místy jsou správně přiřazeny k sekvencím ve FASTA formátu, neboť délky dvou různých sekvencí se liší.

Při provádění kontroly, byl nalezen v několika sekvencích znak „?“, který byl nahrazen znakem „X“ pro neznámou aminokyselinu. Dále u 27 sekvencí v datasetu **HOLO4K** se u sekvence získané z PDB lišila první aminokyselina. Jelikož se tato chyba vyskytuje pouze na prvním místě, a může se týkat pouze 27 vzorků(celkový počet vzorků je v řádu miliónů), tak jsme se rozhodli tuto chybu nebrat v potaz.

Generování embeddingů

K reprezentaci pomocí jazykových modelů bylo třeba vygenerovat embeddingy. K jejich generování bylo využito projektu na GitHubu od H. Gamouh [41]. Jelikož je generování embeddingů časově a paměťově náročné, tak jejich výpočet probíhal na KSI klastru[42], s čímž počítá i z míněný projekt od H. Gamouh.

Pro usnadnění generování embeddingů, jsme pro každý dataset vytvořili textový soubor, který obsahuje všechny sekvence z daného datasetu ve formátu FASTA. Záznam jedné sekvence v tomto souboru začíná komentářem popisujícím danou sekvenci, a poté následuje samotná sekvence ve formátu FASTA. Příklad jednoho záznamu je:

```
>2YGP A CA
TGSLYLWIDAHQARVLIGFEEDILIVSEGKMAPFTHDFRKAQQRWPAIPVN...
```

Kde z komentáře můžeme vyčíst, že sekvence je z proteinu *2YGP*, popisuje v něm řetězec *A* a je součástí datasetu zaměřeného na ligand *CA*(vápník).

Jednotlivé sekvence jsou v datasetech uloženy každá v jednom souboru s příponou *.fasta*. Avšak projekt od H. Gamouh počítá s uložením všech sekvencí v jednom souboru, proto jsme tyto soubory vytvořili.

Pokud máme sekvenci o délce N reziduí pak vygenerované embeddingy k této sekvenci mají tvar matice, desetinných čísel o rozměrech $N \times D$, kde D je délka jednoho embedingu. Při generování může dojít k chybě, která dosadí do matice hodnotu *nan*. Takto vzniknuté chyby byly odstraněny znovu vygenerováním embeddingů k dané sekvenci.

2.fáze přípravy dat

Zatímco 1.fáze přípravy dat je třeba provést pouze jednou, tak druhá se provádí až při spuštění testů. Je tomu tak, protože způsob reprezentace reziduí se může test od testu lišit. Avšak pro případ, kdy se data nemusí generovat při každém spuštění testu je zde možnost ukládat vygenerovaná data a v případě potřeby použít již tyto uložená data a přeskočit tak 2. fázi generování dat. Ale tyto data můžou snadno dosáhnout velikosti gygabytů. Vygenerovaným datům v 2. fázi říkáme **vzorky**. Pokud k reprezentaci reziduí používáme jazykové modely, tak vzorky odpovídají embeddingům. Jeden vzorek je potom vektor s 1024

hodnotami. Pokud používáme reprezentaci pomocí vlastností aminokyselin, tak velikost jednoho vzorku závisí na velikosti zvoleného okolí a použitých vlastnostech aminokyselin viz sekce 3.1.1

V této fázi se také **odstraňují ze vzorků duplicity**. Při reprezentování reziduí pomocí embedddingů stačí zkontrolovat zda jsou všechny sekvence jiné. Embeddingy se generují z celé sekvence, a ze dvou rozdílných sekvencí se vygenerují odlišné embeddingy. Větší šance na výskyt duplicit u vzorků nastává při zvolení reprezentace reziduí pomocí vlastností aminokyselin. V tomto případě záleží podoba vzorku pouze na zvoleném okolí aminokyseliny. Pokud jako okolí považujeme 5 reziduí na každou stranu, tak v sekvenci

`TGSLYLWIDAHQARVLIGFEEDILIVSLYLWIDAHQADFRKAQQRWPAIPVN`

jsou vyznačena dvě shodná místa která budou mít stejný vzorek. v tomto případě by vyznačená část sekvence `SLYLWIDAHQA` popisovala rezidum I. Pokud by ale na popisovaném rezidu bylo v prvním případu vazebné místo a v druhém ne, tak zmíněné dva vzorky za duplicitní nepovažujeme.

3.3 Testovací fáze

Po zpracování datasetů je možné přejít k testování samotných modelů. V testovací fázi se provádí testy, jejichž výsledky slouží k porovnání jednotlivých přístupů k predikci vazebných míst. Během jednoho testu vytvoříme model na trénovací množině vzorků a pomocí něj následně predikujeme vazebná místa na testovací množině vzorků.

3.3.1 Parametry Testu

K vytvoření jednoho testu potřebujeme určit následující parametry.

- **Způsob reprezentace reziduí** – rezidua můžeme reprezentovat pomocí vastností aminokyselin, nebo pomocí embeddingů. Pokud zvolíme k reprezentaci embeddingy, pak potřebujeme zvolit druh vygenerovaných embeddingů, který záleží na zvoleném modelu generujícím embeddingy. Pokud zvolíme k reprezentaci vlastnosti aminokyselin, pak potřebujeme zvolit velikost okolí a konkrétní vlastnosti používané k reprezentaci reziduí ve zvoleném okolí.

- **Metodu strojového učení** – potřebujeme zvolit jakou metodu vybereme k vytvoření modelu pro předpověď vazebných míst.

- **Kandidáty na hyperparametry** – k vytvoření modelu potřebujeme nastavit hyperparametry. Tyto hyperparametry závisí na zvolené metodě a jejich nastavení, ovlivňuje učinost predikce modelu vytvořeného podle zvolené metody. Kandidáty pro daný hyperparametr reprezentujeme seznamem hodnot, kterýma může daný hyperparametr nabývat. Ze seznamu se zvolí nejvhodnější hodnota pro daný hyperparametr.

Volíme z následujících reprezentací reziduí:

- **embeddingy** - Při zvolení této reprezentace, volíme mezi jedním z následujících modelů:
  - ProtBert
  - ProtXLNet
  - ProtT5
- **Vlastnosti aminokyselin** - Při zvolení této reprezentace musíme určit oba následující parametry:
  - Poloměr (volíme číslo r, kde $r \in \mathbb{N}$)
  - Vlastnosti (volíme podmnožinu z následujících vlastností: kód aminokyseliny, třída, polarita, náboj, hydropatický index, hmotnost, četnost)

Při volbě konkrétní metody strojového učení volíme z následujících metod:

- Random Forests
- Gradient Boosting Decision Tree
- Neuronová síť

Nastavované hyperparametry závisí na zvolené metodě. Tomu jaké konkrétní hyperparametry a jejich hodnoty nastavujeme je věnována sekce 3.4

3.3.2 Trénovací, validační, testovací rozdělení a způsob testování modelů

Z dat předpřipravených při první fázi přípravy dat, připravíme vzorky. Tyto vzorky následně dělíme na dvě množiny trénovací a testovací množinu. Rozdělení na testovací a trénovací množinu závisí na vybraných datasetech pro testování.

- **Testovací množina** je předem určená. U datasetů z P2Rank je to *COACH420* a u datasetů rozdělených podle ligandů, jsou tyto množiny už dány předem určeným rozdělením, popsaným v tabulce 3.4.
- **Trénovací množina** je množina vzorků, kde probíhá trénování modelů a hledání optimálních hyperparametrů. Hledání těchto parametrů provádíme pomocí 5-fold crosvalidace, při které dělíme tuto množinu pětkrát na **trénovací** a **validační** část.

Postup testování probíhá v následujících krocích:

1. Určení možných hodnot hyperparametrů, určení metody strojového učení a určení reprezentace reziduí.
2. Získání vzorků ze vstupních dat a jejich rozdělení na testovací, trénovací a validační množinu.
3. Hledání vhodných hyperparametrů na trénovací množině.
4. Natrénování modelu na celé trénovací množině s nejlepšími hyperparametry.
5. Otestování modelu na testovací množině

3.3.3 Hledání vhodných hyperparametrů na trénovací množině.

Při definování testu byly nadefinovány i hyperparametry a seznamy hodnot, které můžeme za ně dosadit. Každou možnou kombinaci z definovaných hodnot hyperparametrů budeme testovat metodou, které se anglicky říká **5-fold cross validation**. Rozdělíme trénovací množinu vždy na 5 stejných částí (tyto části jsou u každé kombinace parametrů stejné). Postupně označíme každou z nich jako validační a na zbylých čtyřech natrénujeme model s danými hyperparametry. Nakonec natrénovaný model otestujeme na validační množině zvolenou metrikou (MCC). Získáme tak 5 hodnot, jejichž průměrem měříme účinnost dané kombinace hodnot hyperparametrů. Nejlepší kombinace hodnot je poté předána do dalšího kroku [43].

3.3.4 Výstup testu

Testem v tomto kontextu rozumíme celý proces otestování jednoho modelu. Výstupem jednoho testu jsou výsledky na testovací množině, kde jako hlavní metrika byl určen *MCC*. Kromě něj se ale měří i *precission*, *recall*, a *accuracy*. Tyto výsledky pochází pouze z jednoho natrénování modelu. Avšak pro porovnání modelů využíváme i výsledků z cross-validace prováděné při hledáním optimálních parametrů. V této fázi se provedlo 5 natrénování modelu na trénovací množině, pro každou kombinaci hyperparametrů, tudíž i pro tu nejvhodnější. Právě výsledky z cross-validace pro nejvhodnější kombinaci hyperparametrů také používáme pro porovnání jednotlivých modelů. Měřené metriky jsou stejné jako ty použité na testovací množinu.

3.4 Hledání vhodných hyperparametrů

Při vytváření testu musíme definovat možné hodnoty pro nastavované hyperparametry. Podle zvolené metody nastavujeme 6 až 9 hyperparametrů, přičemž každý hyperparametr může nabývat jednotek, desítek i stovek hodnot. To znamená, že není možné vyzkoušet, všechny možné kombinace. Proto jsme rozdělili hledání optimálních parametrů do dvou fází. V první fázi hledání hyperparametrů bylo třeba určit hodnoty pro dosazení do cross validace. Tyto hodnoty byly odhadnuty empiricky, vyzkoušením náhodných kombinací hodnot. Rozsahy zkoušených parametrů jsou uvedeny v této sekci v tabulkách 3.6, 3.7, 3.8. Jeden test si v této fázi můžeme představit tak, že každému hyperparametru byla náhodně vybrána jedna, nebo dvě hodnoty z intervalu uvedeném v příslušné tabulce. Na základě těchto výsledků se určily hodnoty dosazené do 2. fáze hledání hyperparametrů. Ve 2. fázi se pro každý hyperparametr vybralo několik hodnot, z pravidla více než v 1. fázi, které byly také dosazeny do cross-validace. Tato fáze se 2-3 zopakovala a nejlepší parametry byly použity k natrénování konečného modelu.

Testy v obou částech probíhají stejným způsobem. V první fázi se dosazují náhodné hodnoty hyperparametrů z většího rozsahu hodnot a s méně hodnotami pro jednotlivé . Zatímco v druhé fázi jsou zkoušené hodnoty odhadnuty na základě výsledků z první části. Dosazují se hodnoty z menšího rozsahu a s více

hodnotami pro jednotlivé parametry. Následně jsou výsledky těchto testů použity pro srovnání jednotlivých modelů.

Jméno hyperparametru	**Rozsah hodnot**
min. počet vzorků v listu	2-40
min. počet vzorků pro rozdělení listu	5-60
max. hloubka stromu	6-22
váha méně zastoupené třídy	6- 21
počet stromů	100 - 200
max. počet atributů	sqrt,log2,None

Tabulka 3.6: Zkoušené parametry–**Random Forest Classifier**
váha méně zastoupené třídy - Jelikož jsou datasety nevyvážené, tak se vzorkům s méně zastoupené třídy dává větší váha.
max. počet atributů - Mezi kolika atributy algotmus volí při rozdělování listu(None...bere v potaz všechny).

Jméno hyperparametru	**Rozsah hodnot**
min. počet vzorků v listu	6-30
max. listů	25-40
max. hloubka stromu	5-15
váha méně zastoupené třídy	5- 17
počet iterací	100 - 200
l2-regularizace	1 - 0.0003
ztrátová funkce	BCEloss

Tabulka 3.7: Zkoušené parametry–**Gradien Boosting D.T.**
BCEloss*(Binary Crosss-Entropy loss) je ztrátová funkce popsaná v sekci 2.3.4 o dopředné neuronové síti*

Jméno hyperparametru	**Rozsah hodnot**
velikost batche	100,1024
learning rate	1e-3 - 1e-5
počet neuronů	50-500
počet skrytých vrstev	1-3
learning rate	0.0002
trpělivost	5-9
tolerance	1e-08,1e-07
počet iterací	100
l2-regularizece	5e-3 - 8e-7
váha méně zastoupené třídy	5- 15
ztrátová funkce	BCEloss

Tabulka 3.8: Zkoušené parametry–**neuronová síť**
velikost batche - Nakonec byla u všech modelů zvolena 1024, a to aby v každém batchi bylo dostatek vazebných vzorků.
trpělivost - Pokud se tento počet iterací nezlepší výsledky loss funkce, tak se trénování zastaví.

3.5 Kód a použité knihovny

Jedním z cílů této práce bylo implementovat nástroj pro otestování metod strojového učení. V této sekci popisujem jaký byl třeba vytvořit kód pro implementaci tohoto nástroje a jeho strukturu.

3.5.1 Knihovny

Pro hledání hyperparametrů, vytvoření modelů využívajících metod *Gradient Boosting* a *Random Forests* jsme použili knihovnu Scikit-learn[44]. Protože mimo stálých aktualizací nabízí i možnost vytvořit modely založené na rozhodovacích stromech a potřebné funkce k otestování algoritmů strojového učení a hledání hyperparametrů. K vytvoření modelů s *neuronovou sítí* jsme ale využily knihovny PyTorch[24], protože nabízí větší volnost při tvoření modelu a širší variabilitu při tvoření modelu. Kompletní seznam použitých knihoven je uveden v GitHubovém repozitáři s kódem.

3.5.2 Kód a jeho dostupnost

Kód je dostupný na GitHub repozitáři spolu s předpřipravenými daty. Jelikož vygenerované embeddingy zabírají jednotky až desítky gigabajtů, tak byly vygenerovány pouze pro datasety *CHEN11*, *COACH420* a datasety pro jednotlivé ligandy *AMP*, *GTP*, *GDP*. Postup, jak provést předpřípravu dat je uveden v *README.md* na zmíněném repozitáři, stejně jako postup k použití hlavního programu. Adresa repozitáře je:
`https://github.com/ProkopDivin/PBSPrediction`.

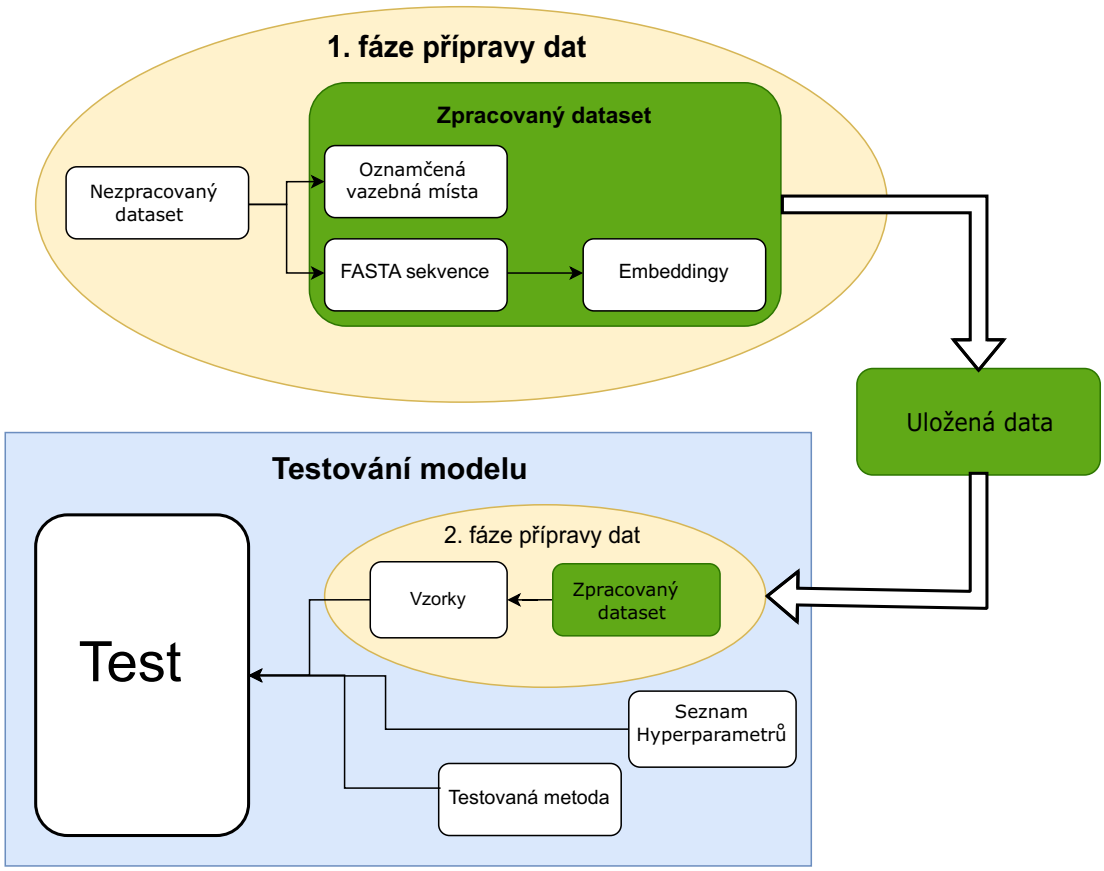
Kód můžeme rozdělit do dvou částí, které jsou znázorněny v obrázku 3.3. V první části předpřipravíme data a poté už tento kód nepoužíváme. Druhá část tvoří už zmíněný nástroj pro testování metod, který má za úkol podle zadaných parametrů vytvořit a provést test.

3.5.3 První fáze přípravy dat

K předpřípravě datasetů, byl napsán skript ***prepareData.sh***, který předpřipraví datasety pomocí nástroje P2Rank. Tato předpříprava se netýká datasetů rozdělených podle ligandů, protože ty už jsou ve vhodné formě. Vše co se v této fázi děje už bylo popsáno v sekci 3.2.1 o přípravě datasetů. Poslední věcí, která se v tomto kroku provádí je generování embeddingů, které je také popsané v sekci 3.2.1, a ke kterému jsme využili projektu na GitHubu od H. Gamouh [41]. Stručně můžeme říct, že po skončení této fáze máme připravené data, které se používají jako vstup při testování modelů. Jelikož se jedná pouze o spuštění příkazů z P2Rank a pak následnou kontrolu a vhodné uložení výsledků, tak tento kód zde už nebudeme více popisovat.

3.5.4 Testování modelů

Proto abychom mohli otestovat různé přístupy, tak jsme vytvořili program pro testování modelů, který odpovídá části obrázku 3.3 s modrým pozadím. Tento



Obrázek 3.3: Schéma přípravy dat a testování

nástroj má za úkol

- připravit vzorky podle zvolené reprezentace dat,
- vytvořit model podle zvolené metody strojového učení,
- natrénovat tento model a otestovat ho. Tato část zahrnuje natrénování modelu na trénovacích vzorcích, určení optimálních hodnot hyperparametrů a otestovaní modelu na testovacích vzorcích.

Podrobněji celý princip testování modelu byl popsán v sekci 3.3, proto zde popisujeme tento proces z programátorského hlediska. Test se spouští pomocí skriptu ***main.py***, kterému je třeba zadat parametry popisující daný test, nebo číslo testu které se má provést. Protože hledání optimálních parametrů může zabrat netriviální čas a většinou se provádí více testů naráz, tak jednotlivé testy běžely na KSI klastru[42]. Z tohoto důvodu existuje možnost spustit test pomocí jeho čísla. Rozhraní klastru nám dovoluje naráz spustit několikrát skript ***main.py*** vždy s jeho pořadovým číslem jako parametrem. Toto se hodí při spouštění více testů naráz, kdy pořadové číslo používáme jako číslo testu.

Výstup programu se ukládá do složky, kde můžeme najít výsledky rozdělené do podsložek podle metody a použitého datasetu. V těchto složkách je také podrobný záznam výsledků testů. Pro kompaktnější a stručnější shrnutí výsledků

do jednoho souboru, je zde skript ***summaryResults.py***, který shrne všechny výsledky do jedné tabulky ve formátu *csv*.

program můsí plnit tři úkoly:

- Přípravu vzorků.
- Přípravu testu.
- Testování modelu.

Příprava vzorků

Jak víme ze sekce o datech 3.2, tak máme dva druhy datasetů. Datasety pro jednotlivé ligandy a datasety používané při testování P2Rank. Tyto datasety se liší způsobem reprezentace vazebných míst. A přístupem k jejich zpracování. Datasety použité při testování P2Rank byly původně přizpůsobeny k metodám využívající 3D strukturu proteinu, proto vyžadují více přípravy než datasety pro jednotlivé ligandy. Naproti tomu datasety pro jednotlivé ligandy byly již ve formě vhodné pro naše účely. Navíc abychom zachovali možnost porovnat naše výsledky s výsledky někoho, kdo používá stejné datasety tak je chceme ponechat nezměněné. Kvůli odlišnému přístupu jsme se rozhodli zpracovávat každý z druhů datasetů zvlášť. Část programu starající se o přípravu dat je v souborech:

- **datafiles.py** - obsahuje pouze třídu s cestami k souborům, které program používá aby všechny cesty byly na jednom místě.
- **nonkontexFeatures.csv** - tabulka s nekontextovými vlastnostmi aminokyselin stejná jako tabulka 3.1.
- **aminoacidFeatures.py** - obsahuje třídu, která z tabulky s nekontextovými vlastnostmi(nonkontexFeatures.csv), vytvoří tabulku pro převod jednotlivých reziduí v sekvenci na jejich reprezentující vektory.
- **preprocesingDataset.py** - obsahuje kód který vytvoří ve 2. fázi přípravy dat vzorky pro datasety společné s P2Rank, které nejsou rozděleny podle ligandů.
- **preprocesingLigands.py** - obsahuje kód který vytvoří ve 2. fázi přípravy dat vzorky pro datasety rozdělené podle ligandů.

Příprava testu

Tato část programu obstarává přípravu modelu a jeho nastavení pro trénování. Také jsou její součástí funkce automaticky generující parametry testů pro hromadné testování pomocí klastru. Část programu obstarávající tyto funkce je v souborech:

- **testerUtils.py** - nachází se zde třída která je předkem pro třídy zařizující přípravu testů. Má v sobě funkce společné pro obě třídy zařizující přípravu testů.

- **featureTeater.py** - obsahuje kód který vytvoří model a připraví pro něj data. Stará se o modely využívající k reprezentaci vlastnosti aminokyselin.
- **embeddingsTester.py** obsahuje kód který vytvoří model a připraví pro něj data. Stará se o modely využívající k reprezentaci embeddingy.
- **main.py** - zde začíná program a nachází se funkce *main*.
- **estimators.py** - obsahuje třídu obalující neuronovou síť vytvořenou pomocí knihovny *Pytorch*. Modely potřebujeme zaobalit touto třídou abychom mohli použít stejné funkce k hledání optimálních parametrů jako pro modely vytvořené pomocí ostatních metod.
- **grids.py** - obsahuje třídy se slovníky obsahující seznamy hodnot zkoušených hyperparametrů pro dané metody. Tyto hodnoty v seznamech nejsou jediné zkoušené hodnoty ale jsou zde aby bylo možné vygenerovat poslední provedené testy. Jméno je odvozeno od objektu provádějícího optimalizaci hyperparametrů(GridSearchCV).
- **statistics.py** - obsahuje třídu určenou pro uchování parametrů testu, aby byly známé při vyhodnocování testu a také třídu, která vyhodnotí model určenými metrikami.
- **tests.py** - kód s funkcemi generující parametry testů. Při spouštění testů na klastru, bylo vždycky spuštěno několik testů naráz. Proto jsou zde funkce pro generování parametrů které definují test, aby nemuseli být parametry zadávány ručně.

Testování modelu

Tato část programu má za úkol natrénovat a otestovat model. Také se zde nachází kód shrnující výsledky testů do jedné *csv* tabulky.

- **tunning.py** - obsahuje kód, kde probíhá trénování vytvořeného modelu, hledání optimálních hyperparametrů a otestování natrénovaného modelu.
- **summaryResults.py** - shrne vyprodukované výsledky několika testů do jedné tabulky. Muže být také spuštěn sám o sobě s parametrem *-r* za kterým následuje cesta ke složce s výsledky.

4. Výsledky a diskuze

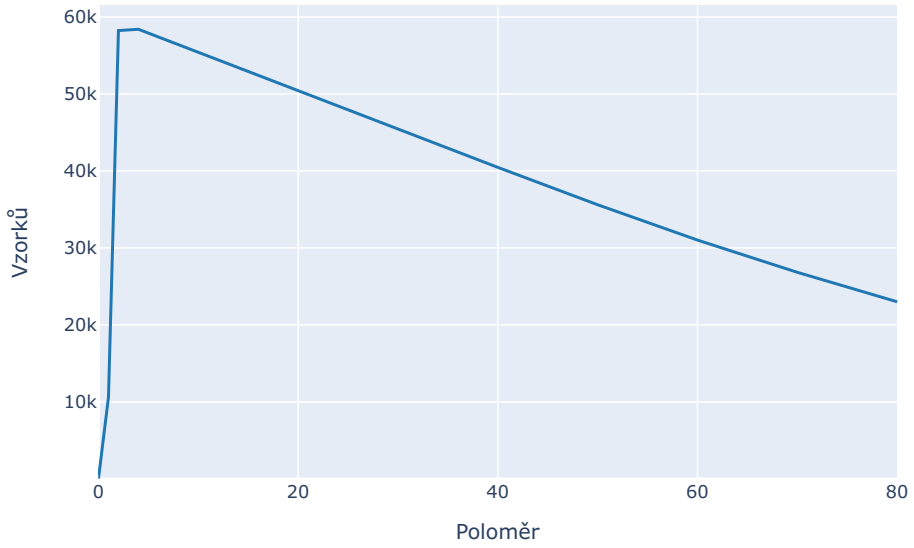
V této kapitole představujeme výsledky naší práce. Nejprve zde představíme výsledky modelů používajících k zakódování reziduí vlastnosti aminokyselin. Poté představíme modely používající embeddingy a následně tyto dva přístupy srovnáme. Nadále vyhodnotíme jaká metoda strojového učení je nejvhodnější pro náš úkol. Jako poslední srovnáme výsledky s P2Rank, což je nástroj, který používáme jako zástupce moderních nástrojů používajících 3D strukturu proteinu.

4.1 Kódování pomocí vlastností aminokyselin

V této sekci používáme ke kódování reziduí jejich vlastnosti ze sekce 1.1.1. Jako trénovací dataset byl použit dataset *CHEN11* a jako testovací *COACH420*. Tyto datasety byly získány pomocí nástroje P2Rank. Dataset *CHEN11* byl použit k natrénování defaultního modelu v P2Rank, zatímco *COACH420* byl použit k jako testovací množina. Takovéto rozvržení datasetů budeme používat i v této sekci. Výsledky pokusů v této sekci jsou shrnuty v poněkud rozsáhlých tabulkách. Kvůli usnadnění čtení dat z nich jsme se rozhodli podbarvit každý druhý řádek v tabulkách této sekce.

4.1.1 Vliv velikosti okolí

K tomu abychom mohli zkoumat vliv na to, jak vhodné jsou vlastnosti aminokyselin pro reprezentaci reziduí, musíme napřed určit poloměr okolí, které budeme brát v potaz. Větší okolí znamená více informací o kontextu zkoumaného rezidua, ale také nám zabraňuje zkoumat rezidua na začátku a konci sekvence. Jelikož tyto krajní rezidua nezkoumáme, tak s rostoucím poloměrem bude ubývat vzorků, které můžeme použít pro trénování. Naproti tomu i malé okolí znamená méně vzorků, to protože z dat před trénováním odstraňujeme duplicity. Ačkoliv testovací a trénovací množina nemá více kopií stejného proteinu, tak při zvolení této reprezentace můžeme získat duplicitní vzorky ze shodných částí proteinů. Jsou dva druhy duplicit, které odstraňujeme. První druh duplicit vzniká ze shodných částí proteinů ve stejném datasetu. Z takto vzniklých duplicit ponecháme vždy pouze jednoho reprezentanta. Druhý druh vzniká ze shodných částí proteinů z nichž, jedna je v trénovacím datasetu a druhá je v testovacím datasetu. U tohoto druhu odstraňujeme duplicity vzniklé v testovacím datasetu. V grafu na obrázku 4.1 můžeme vidět kolik vzorků máme k dispozici v závislosti na poloměru v datasetu *CHEN11*. Nejmenší počet vzorků máme k dispozici při poloměru 0, kdy je jeden vzorek určen pouze vlastnostmi jedné aminokyseliny. Takových to vzorků je k dispozici 40, což odpovídá tomu, že v datasetu jsou proteiny tvořené 20 druhy aminokyselin a ke každé aminokyselině máme jeden vzorek označený jako vazebný a jeden označený jako nevazebný. Konkrétní hodnoty z grafu na obrázku 4.1 jsou uvedeny v druhém sloupci tabulky 4.1, v třetím sloupci jsou počty vzorků v testovací množině s odstraněnými duplicitami prvního druhu a v posledním sloupci je reálný počet testovacích vzorků po odstranění duplicit prvního a druhého druhu. Z grafu na obrázku 4.1 je patrné, že s rostoucím poloměrem přicházíme poměrně rychle o příležitost predikovat vazebnost reziduí na krajích



Obrázek 4.1: Počet vzorků v závislosti na poloměru okolí (CHEN11)
$r = 0$ odpovídá samotné aminokyselině, zatímco $r = 80$ odpovídá úseku dlouhém 161 aminokyselin.

sekvence. Z tohoto důvodu nezkoumáme poloměr větší než 80 a zaměřujeme se spíš na nižší hodnoty tohoto parametru. Oproti tomu poloměr s hodnotou 0 a 1 také nebudeme zkoumat, protože při těchto hodnotách máme málo testovacích vzorků a výsledky by mohly být tímto zkreslené.

Radius	**CHEN11**	**COACH420**	**COACH420 (bez duplicit)***
0	40	40	0
1	10601	11052	1998
2	58231	103513	95432
4	58402	106879	102989
6	57402	105534	101843
7	56902	104837	101229
8	56402	104130	100596
9	55902	103417	99956
10	55402	102695	99301
12	54402	101229	97962
14	53402	99763	96622
16	52402	98293	95266
18	51402	96816	93901
21	49902	94594	91840
25	47902	91606	89065
30	45404	87846	85551
35	42917	84085	81999
40	40464	80323	78412
50	35609	72801	71180
60	31021	65451	64094
70	26832	58395	57285
80	23006	51645	50746

Tabulka 4.1: Počet vzorků v závislosti na velikosti poloměru okolí
*Reálný počet testovacích vzorků, které máme k dispozici, je roven počtu unikátních vzorků v COACH420, které nejsou v CHEN11.

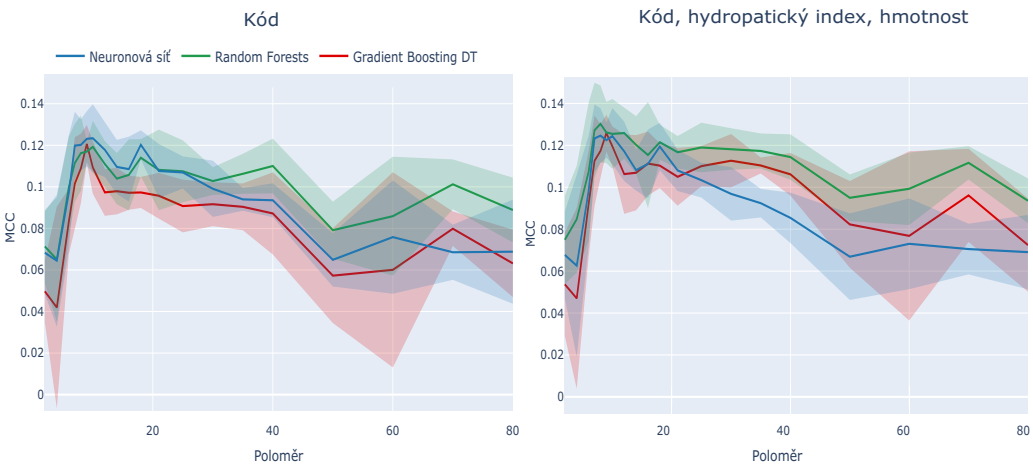
Hledání hyperparametrů

I volba poloměru okolí může ovlivnit optimální hodnoty hyperparametrů. Poloměr samotný jsme tedy v tomto případě považovali také za hyperparametr. Proto abychom našli vhodné hodnoty ke každému hyperparametru, jsme použili postup popsaný v sekci 3.4, kde nejprve empiricky odhadneme možné hodnoty pro každý parametr a poté následně pomocí cross-validace vybereme nejlepší kombinaci hodnot hyperparametrů.

Vliv velikosti okolí na předpověď vazebných reziduí

Abychom mohli pozorovat, jak se budou modely chovat vzhledem k měnícímu se poloměru, tak jsme vybrali dva typy reprezentace reziduí. V prvním typu reprezentujeme rezidua pomocí jejich kódu převedeného do *one-hot* reprezentace, kterou bereme jako nejzákladnější a nejpřímočařejší způsob reprezentace. V druhém typu jsme k reprezentaci reziduí použili vektor s hodnotami pro kód ve *one-hot* reprezentaci, hydropatický index, a hmotnost. Tyto vlastnosti byly vybrány, protože při úvodních testech výsledky naznačovaly, že by mohly být lepším kandidátem na reprezentaci než samotný kód. Avšak o to, jak dobře lze jednotlivými vlastnostmi popsat dané reziduum se zabýváme až po určení poloměru v sekci 4.1.2.

Na obrázku 4.2 můžeme vidět grafy průměrných výsledků na validační množině spolu s jejich standardní odchylkou. V grafech na obrázku 4.3 jsou znázorněny výsledky na testovací množině.



Obrázek 4.2: (MCC)Výsledky cross-validace se standardní odchylkou při změně poloměru okolí.

Hodnoty pro jednotlivé grafy se nacházejí v tabulkách 4.3 a 4.4. Z těchto výsledků můžeme vypozorovat trend, kdy po překročení hodnoty r v intervalu $\langle 6{,}14 \rangle$ se s rostoucím poloměrem zmenšuje hodnota MCC. Tento jev vysvětlujeme úbytkem trénovacích vzorků. Interval $\langle 6{,}14 \rangle$ jsme vybrali protože se v něm nachází většina největších hodnot MCC(na testovací množině) z provedených testů jak můžeme vidět tabulce 4.2. Oproti tomu při malých hodnotách poloměru MCC s rostoucím poloměrem stoupá, což vypovídá o tom, že zvětšování okolí má smysl.

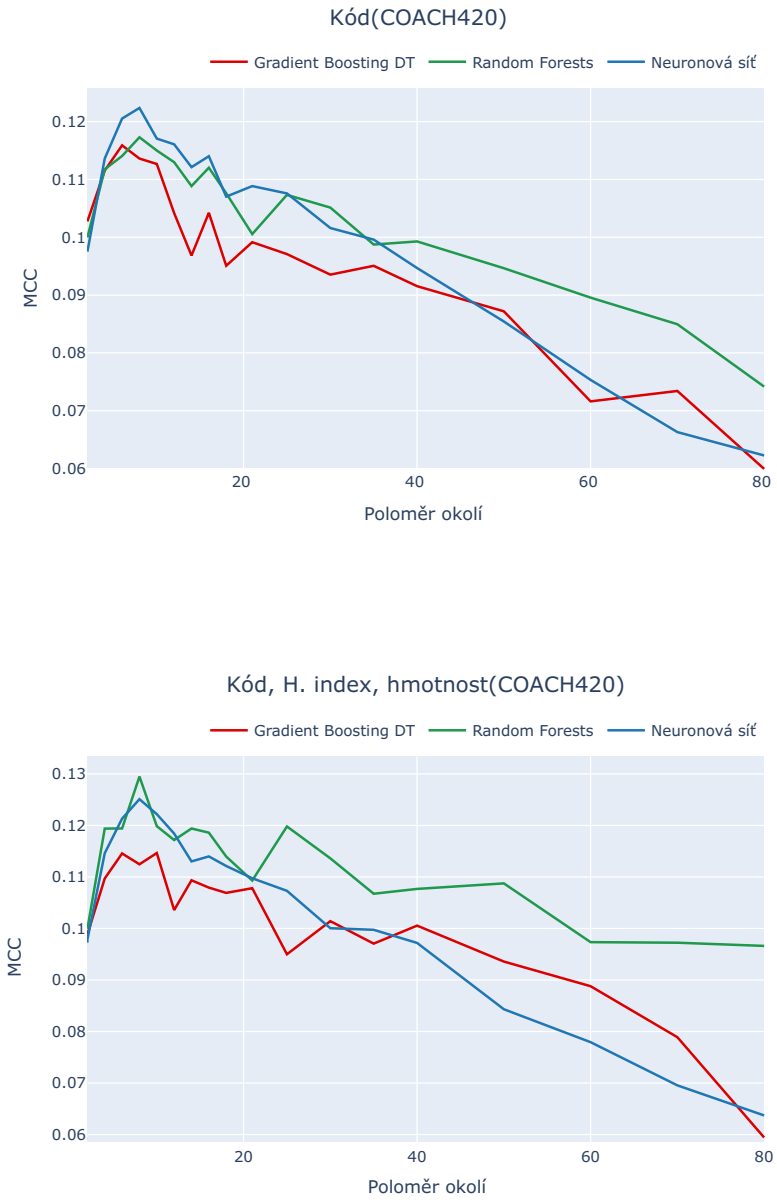
Bohužel tento benefit časem překoná negativní vliv úbytku dostupných vzorků. To nás dovedlo k výběru poloměru $r = 10$, pro testování reprezentace reziduí pomocí jednotlivých vlastností.

Test	**Počet 5 největších hodnot MCC,** $r \in [6, 8, 10, 12, 14]$		
	RF	GB	NN
Kód	4	4	4
Kód+H.index+Hmotnost	4	3	4

Tabulka 4.2: Počet nejlepších výsledků MCC na COACH420

Poloměr	**RF MCC**		**GB MCC**		**NN MCC**	
	Validace	Test	Validace	Test	Validace	Test
2	0.071 (±0.018)	0.100	0.050 (±0.015)	0.103	0.068 (±0.020)	0.098
4	0.065 (±0.030)	0.112	0.042 (±0.049)	0.112	0.064 (±0.032)	0.114
6	0.100 (±0.024)	0.114	0.083 (±0.016)	0.116	0.097 (±0.028)	0.121
8	0.116 (±0.018)	0.117	0.109 (±0.017)	0.114	0.120 (±0.012)	0.122
10	0.119 (±0.013)	0.115	0.109 (±0.013)	0.113	0.123 (±0.016)	0.117
12	0.111 (±0.011)	0.113	0.097 (±0.011)	0.104	0.118 (±0.013)	0.116
14	0.104 (±0.012)	0.109	0.098 (±0.011)	0.097	0.110 (±0.013)	0.112
16	0.106 (±0.017)	0.112	0.097 (±0.008)	0.104	0.109 (±0.016)	0.114
18	0.114 (±0.009)	0.108	0.097 (±0.007)	0.095	0.120 (±0.007)	0.109
21	0.108 (±0.019)	0.101	0.096 (±0.011)	0.099	0.108 (±0.013)	0.109
25	0.107 (±0.015)	0.107	0.091 (±0.013)	0.097	0.107 (±0.008)	0.108
30	0.103 (±0.007)	0.105	0.092 (±0.011)	0.094	0.099 (±0.014)	0.103
35	0.106 (±0.010)	0.099	0.090 (±0.011)	0.095	0.094 (±0.005)	0.101
40	0.110 (±0.013)	0.099	0.087 (±0.020)	0.092	0.094 (±0.008)	0.095
50	0.079 (±0.014)	0.095	0.057 (±0.023)	0.087	0.065 (±0.013)	0.085
60	0.086 (±0.029)	0.090	0.060 (±0.047)	0.072	0.076 (±0.027)	0.075
70	0.101 (±0.012)	0.085	0.080 (±0.008)	0.073	0.069 (±0.013)	0.066
80	0.089 (±0.016)	0.074	0.063 (±0.016)	0.060	0.069 (±0.025)	0.062

Tabulka 4.3: MCC v závislosti na poloměru (kód)



Obrázek 4.3: (MCC)Výsledky na testovací množině při změně poloměru okolí.

Poloměr	RF MCC		GB MCC		NN MCC	
	Validace	Test	Validace	Test	Validace	Test
2	0.075 (±0.021)	0.100	0.054 (±0.025)	0.099	0.068 (±0.021)	0.098
4	0.085 (±0.026)	0.119	0.047 (±0.043)	0.110	0.063 (±0.043)	0.115
6	0.107 (±0.033)	0.119	0.092 (±0.025)	0.115	0.098 (±0.026)	0.121
8	0.130 (±0.018)	0.129	0.117 (±0.013)	0.112	0.125 (±0.013)	0.125
10	0.125 (±0.017)	0.120	0.120 (±0.009)	0.115	0.124 (±0.013)	0.122
12	0.126 (±0.012)	0.117	0.106 (±0.019)	0.104	0.117 (±0.014)	0.118
14	0.120 (±0.014)	0.119	0.107 (±0.018)	0.109	0.108 (±0.010)	0.113
16	0.115 (±0.025)	0.119	0.111 (±0.015)	0.108	0.111 (±0.016)	0.114
18	0.121 (±0.008)	0.114	0.110 (±0.011)	0.107	0.119 (±0.011)	0.114
21	0.117 (±0.008)	0.109	0.105 (±0.014)	0.108	0.108 (±0.010)	0.110
25	0.119 (±0.012)	0.120	0.110 (±0.010)	0.095	0.103 (±0.008)	0.107
30	0.118 (±0.010)	0.114	0.113 (±0.013)	0.101	0.097 (±0.013)	0.100
35	0.117 (±0.008)	0.107	0.110 (±0.004)	0.097	0.092 (±0.007)	0.100
40	0.114 (±0.011)	0.108	0.106 (±0.010)	0.101	0.085 (±0.012)	0.098
50	0.095 (±0.011)	0.109	0.082 (±0.021)	0.094	0.067 (±0.021)	0.084
60	0.099 (±0.017)	0.097	0.077 (±0.040)	0.089	0.073 (±0.022)	0.078
70	0.112 (±0.008)	0.097	0.096 (±0.022)	0.079	0.071 (±0.012)	0.072
80	0.094 (±0.011)	0.097	0.072 (±0.022)	0.059	0.069 (±0.018)	0.064

Tabulka 4.4: MCC v závislosti na poloměru (kód, hydropatický ind., hmotnost)

4.1.2 Užitečnost kódování pomocí vlastností aminokyselin

Teď se budeme věnovat otázce, jaké vlastnosti aminokyselin jsou vhodné pro reprezentaci jednotlivých reziduí. Pro tyto testy jsme určili pevný poloměr okolí $r = 10$, jak je popsané v sekci 4.1.1. Nejprve jsme provedli testy, kdy každé reziduum bylo reprezentováno pomocí jedné vlastnosti aminokyselin. Tyto testy provádíme abychom získali představu o tom, jaké vlastnosti je dobré použít pro reprezentaci. Jeden vzorek je při zvoleném poloměru buď vektor 21 hodnot, pokud kategorické proměnné reprezentujeme ordinálně, nebo vektor vzniklý spojením 21 vektorů z nichž každý představuje one-hot reprezentaci kategorického atributu. Výsledné *MCC* a to jak na trénovací, tak testovací množině je uvedeno v tabulce 4.5. V tabulce 4.6 jsou zaznamenány výsledky pro *preccision*, *recall* a *accuracy*.

Vlastnost	Trénovací (5-fold CV)			Testovací COACH420		
	RF MCC	GB MCC	NN MCC	RF MCC	GB MCC	NN MCC
Kód	0.079 (±0.011)	0.100 (±0.013)	0.013 (±0.008)	0.083	0.090	0.011
Kód (o)	**0.119 (±0.013)**	**0.112 (±0.009)**	**0.124 (±0.014)**	0.115	0.110	0.117
Třída	0.088 (±0.010)	0.091 (±0.018)	0.045 (±0.008)	0.071	0.072	0.032
Třída (o)	0.106 (±0.009)	0.097 (±0.011)	0.093 (±0.019)	0.093	0.082	0.093
Polarita	0.071 (±0.011)	0.071 (±0.008)	0.051 (±0.011)	0.061	0.059	0.031
Polarita (o)	0.070 (±0.007)	0.068 (±0.007)	0.078 (±0.013)	0.077	0.067	0.063
Náboj	0.100 (±0.019)	0.090 (±0.011)	0.089 (±0.032)	0.114	0.103	0.114
Náboj (o)	0.106 (±0.019)	0.095 (±0.018)	0.110 (±0.022)	**0.119**	0.110	**0.126**
Hdr. Index	0.105 (±0.014)	0.098 (±0.016)	0.077 (±0.012)	0.109	**0.112**	0.071
Hmotnost	0.106 (±0.020)	0.104 (±0.013)	0.080 (±0.012)	0.094	0.094	0.070
Četnost	0.095 (±0.011)	0.100 (±0.009)	0.060 (±0.011)	0.079	0.090	0.033

Tabulka 4.5: Porovnání MCC, atributy získané pomocí jednotlivých vlastností (o)-označuje kategorickou vlastnost, která byla převedena do *one-hot* reprezentace. **Tučně** vyznačené hodnoty jsou největší hodnoty v daném sloupci.

Vlastnost	Random Forests			Gradient Boosting			Neuronová síť		
	Accuracy	**Precision**	**Recall**	**Accuracy**	**Precision**	**Recall**	**Accuracy**	**Precision**	**Recall**
Kód	0.838	0.258	0.099	0.828	0.285	0.100	0.845	0.132	0.059
Kód (o)	0.852	0.284	0.116	0.803	0.363	0.105	0.853	0.285	0.120
Třída	0.749	0.365	0.081	0.817	0.271	0.090	0.761	0.270	0.066
Třída (o)	0.654	**0.537**	0.081	0.670	0.492	0.079	0.709	**0.467**	0.085
Polarita	0.795	0.283	0.082	0.653	0.461	0.071	0.740	0.295	0.065
Polarita (o)	0.743	0.387	0.083	0.820	0.257	0.087	0.825	0.243	0.086
Náboj	0.823	0.321	0.122	0.839	0.275	0.121	**0.878**	0.217	**0.146**
Náboj (o)	0.838	0.302	**0.129**	0.838	0.287	**0.125**	0.866	0.259	**0.146**
H.index	**0.862**	0.259	0.121	**0.866**	0.257	0.124	0.718	0.405	0.078
Hmotnost	0.835	0.281	0.104	0.690	**0.494**	0.084	0.698	**0.430**	0.077
Četnost	0.788	0.327	0.089	0.810	0.315	0.097	0.755	0.280	0.066

Tabulka 4.6: Porovnání ostatních metrik, atributy získané pomocí jednotlivých vlastností
(o)-označuje kategorickou vlastnost, která byla převedene do *one-hot* reprezentace. **Tučně** vyznačené hodnoty jsou největší hodnoty v daném sloupci.

Kombinace vlastností

Po vyzkoušení jednotlivých vlastností pro reprezentaci reziduí se nám nabízí otázka, jestli jejich kombinace, může zlepšit účinnost natrénovaných modelů? Mohlo by se zdát, že při kombinacích různých vlastností, bude mít model přesnější informace o okolí daného rezidua tudíž bude i lepší v predikci. Na druhou stranu při reprezentaci pomocí více vlastností nemusí model tyto informace využít, ani nevíme, jestli jsou tyto informace pro predikci vazebných reziduí relevantní. Zároveň Vyzkoušet všechny možné kombinace všech 7 vlastností, kterých je $2^7 - 1 = 127$, není v našich silách. Proto jsme zvolili následující kombinace vlastností, podle výsledků testů v tabulce 4.5:

- **kód, hydropatický index, hmotnost** - V této kombinaci byly vybrány dvě vlastnosti s nekategorickými hodnotami, které měly nejlepší výsledky. Následně k nim by přidán kód, jakožto nejzákladnější způsob reprezentace.
- **kód, náboj** - V této kombinaci byly vybrány dvě vlastnosti s kategorickými hodnotami, které měly nejlepší výsledky.
- **všechny** - Všech 7 vlastností, které jsme zde zkoumali.

Ke každé výše uvedené kombinaci jsme testovali dvě varianty. První varianta měla všechny vlastnosti s kategorickými hodnotami zakódované ordinálně a druhá pomocí *one-hot* reprezentace. Výsledky na testovací množině jsou v tabulce 4.7. Výsledky z cross-validace jsou v tabulce 4.8 a v tabulce 4.9 jsou výsledky ostatních metrik.

4.1.3 Závěr-kódování pomocí vlastností aminokyselin

Postupně jsme vyzkoušeli 17 typů reprezentace reziduí, z nichž jsme u 7 použili *one-hot* kódování k zakódování kategorických vlastností. Vždy když bylo použito *one-hot* kódování tak byla vyzkoušena i varianta, kde byly kategorické proměnné zakódovány pouze ordinálně. Pokud porovnáme vždy variantu s *one-hot* reprezentací a bez ní, pak pro všechny metody na testovací množině byla varianta s *one-hot* kódováním lepší. Konkrétně průměrný MCC z těchto 7 vlastností je pro metodu *Random Forests* 0.105 pro variantu s *one-hot* kódováním a 0.092

Vlastnost	Testovací množina(COACH420)		
	RF MCC	GB MCC	NN MCC
Kód	0.083	0.090	0.011
Kód (o)	0.113	0.110	0.117
kód+H.index+hmotnost	0.110	0.106	0.086
Kód+H.index+hmotnost(o)	0.116	0.112	0.120
Kód+náboj	0.096	0.097	0.061
Kód+náboj(o)	0.110	0.106	0.119
Všechny	0.108	0.110	0.101
Všechny(o)	0.110	0.114	0.115

Tabulka 4.7: Porovnání testovací MCC, kombinace vlastností
(o)-označuje řádek, kde byly všechny kategorické vlastnosti převedeny do *one-hot* reprezentace

Vlastnost	Trénovací množina(5-fold CV)		
	RF MCC	GB MCC	NN MCC
Kód	0.079 (±0.011)	0.100 (±0.013)	0.013(±0.008)
Kód (o)	0.116 (±0.013)	0.112 (±0.009)	0.124(±0.014)
Kód+H.index+hmotnost	0.118(±0.013)	0.117(±0.019)	0.101(±0.016)
Kód+H.index+hmotnost(o)	0.125(±0.017)	0.120(±0.009)	0.124(±0.012)
Kód+náboj	0.081(±0.021)	0.092(±0.023)	0.052(±0.010)
Kód+náboj(o)	0.113(±0.011)	0.114(±0.005)	0.124(±0.014)
Všechny	0.120(±0.016)	0.122(±0.014)	0.119(±0.012)
Všechny(o)	0.125(±0.013)	0.123(±0.009)	0.119(±0.010)

Tabulka 4.8: Porovnání trénovací MCC, kombinace vlastností
(o)-označuje řádek, kde byly všechny kategorické vlastnosti převedeny do *one-hot* reprezentace

Vlastnost	Random Forests			Gradient Boosting			Neuronová síť		
	Accuracy	Precision	Recall	Accuracy	Precision	Recall	Accuracy	Precision	Recall
Kód	0.838	0.258	0.099	0.828	0.285	0.100	0.845	0.132	0.059
Kód (o)	0.795	0.382	0.104	0.803	0.363	0.105	0.853	0.285	0.120
Kód+H.index+hmotnost	0.833	0.312	0.111	0.700	0.507	0.089	0.871	0.206	0.111
Kód+H.index+hmotnost(o)	0.731	0.486	0.096	0.842	0.300	0.115	0.829	0.339	0.115
Kód+náboj	0.732	0.441	0.088	0.899	0.169	0.133	0.740	0.357	0.076
Kód+náboj(o)	0.778	0.401	0.100	0.806	0.349	0.103	0.792	0.398	0.106
Všechny	0.856	0.269	0.118	0.801	0.365	0.104	0.849	0.270	0.112
Všechny (o)	0.771	0.498	0.091	0.8171	0.357	0.109	0.803	0.428	0.102

Tabulka 4.9: Porovnání ostatních metrik, atributy získané pomocí vlastností

pro variantu bez *one-hot* kódování, pro metodu *Gradient Boosting* je to 0.100 a 0.091 a pro metodu používající *neuronovou síť* je to 0.108 a 0.062. V sekci 3.1.1 o vlastnostech aminokyselin jsme předpokládali, že *one-hot* kódování bude vhodné pouze pro *neuronovou síť*, avšak výsledky v tabulkách 4.5 a 4.7 ukazují, že je vhodné pro všechny tři metody.

V tabulkách 4.7 a 4.8, můžeme vidět porovnání modelů, kde k reprezentaci reziduí byly použity kombinace vlastností aminokyselin. Z těchto tabulek je vidět, že žádná z kombinací vlastností s *one-hot* kódováním nepřináší výrazně lepší výsledky než ty ostatní. Nejlepší výsledky mají dva modely, model kde byly použity všechny vlastnosti a model kde byl použit kód, hydropatický index a hmotnost. Jako hlavní model se, kterým budeme porovnávat ostatní výsledky, budeme brát

model používající kód, hydropatický index a hmotnost, protože tento model překonal všechny ostatní modely na testovací množině. Tento model je porovnán v tabulce 4.10 s modelem používajícím kód aminokyseliny ve *one-hot* kódování, jakožto nejzákladnějším typem reprezentace. Všechny tyto modely byly trénovány na množině CHEN11 a otestovány na množině COACH420.

Metoda	**MCC**	**Accuracy**	**Precision**	**Recall**
RF[1]	0.113	0.838	0.258	0.099
GB[1]	0.110	0.828	0.285	0.100
NN[1]	0.117	0.845	0.132	0.120
RF[2]	0.116	0.731	0.486	0.096
GB[2]	0.112	0.842	0.300	0.115
NN[2]	0.120	0.829	0.339	0.115

Tabulka 4.10: Porovnání modelů
[1]-Pro reprezentaci reziduí byl použit Kód ve *one-hot* kódování. [2]-Pro reprezentaci reziduí byl použit kód ve *one-hot* kódování, Hydropatický index, hmotnost.

4.2 Kódování pomocí embeddingů

V této sekci používáme ke kódování reziduí embeddingy z sekce 3.1.2. Embeddingy samy o sobě nesou informaci o okolí daného rezidua, proto nebereme v potaz sousedící rezidua v sekvenci, jak tomu bylo při reprezentování reziduí pomocí vlastností aminokyselin. Jeden vzorek je v tomto případě tvořen pouze embeddingem, který mu byl počítán. Od tohoto typu reprezentace očekáváme, že bude lepší než reprezentace pomocí vlastností aminokyselin. Jak jsme ukázali v sekci 3.1.2, tak problém hledání reprezentace reziduí v sekvenci se dá připodobnit k hledání reprezentace slova ve větě. Jelikož se v tomto úkolu embeddingy osvědčili, tak lze předpokládat, že budou vhodné i pro reprezentaci reziduí v sekvenci. My využíváme embeddingy vygenerované modely(V závorkách jsou námi používané zkratky): *prottrans_xlnet_uniref100*(ProtXlnet), *prottrans_bert_bfd*(ProtBert) a *prottrans_t5_uniref50*(ProtT5) z knihovny *Bio Embeddings* [35]. Z použitých modelů je nejsložitější a zároveň i největší modelem ProtT5. Předpokládáme, že proto bude nejvhodnějším zdrojem embeddingů.

V této sekci představíme výsledky u modelů, které používají k reprezentaci reziduí embeddingy. Následně vyhodnotíme rozdíl mezi druhy použitých embeddingů. Výsledky z této sekce jsou poté použity v nadcházejících sekcích 4.4 a 4.5, kde srovnáváme zkoumané metody a srovnáváme způsoby reprezentací reziduí.

4.2.1 Datasety rozdělené podle ligandů

Nyní se věnujeme 12 datasetům, u které byly zkonstruovány vždy pro jeden ligand. U každého datasetu je předem dané rozdělení na testovací a trénovací množinu. Důvodem proč by mohlo být vhodné dělit vazebná místa podle ligandu, který se na ně váže, je možnost natrénovat pro každý ligand jeden model zvlášť. Což nám dává možnost vyladit hyperparametry tak, aby se model lépe přizpůsobil danému ligandu.

Pro každý ligand, testovanou metodu a druh embeddingů byl natrénován model. V tabulce 4.12 je uvedeno výsledné *MCC* a jeho standartní směrodatná odchylka na trénovací množině. V tabulce 4.13 je uvedeno *MCC* změřené na testovací množině. Údaje v těchto tabulkách také používáme k porovnání zkoumaných metod v sekci 4.5, a také k porovnání účinnosti jednotlivých embeddingů v sekci 4.3.

Z výsledků na testovací množině můžeme vidět, že záleží na druhu datasetu, neboť *MCC* se mezi jednotlivými modely výrazně liší. Najdeme zde model dosahující 0.108 ale i model dosahující 0.699, který se podařilo natrénovat na datasetu zaměřeného na ligand *FE* a používajícího *neuronovou síť*. Ačkoliv se velikosti trénovacích množin pohybovaly od 21 850 vzorků po 354 576 vzorků, tak jsme zde závislost velikosti MCC na velikosti trénovací množiny nepozorovali. Což můžeme například vidět v tabulce 4.11, kde jsou vzestupně seřazeny datasety podle velikosti a k nim největší hodnota MCC. V této tabulce můžeme například vidět, že na mnoho ligandech v levé části tabulky s malou trénovací množinou bylo změřeno vyšší MCC než na ligandech v pravé části s velkou trénovací množinou.

Dataset	GTP	GDP	AMP	HEME	FE	DNA	ATP	ADP	MN	CA	ZN	MG
MCC	573	691	426	680	699	364	545	517	523	312	577	331

Tabulka 4.11: Vzestupně seřazené datasety podle velikosti, s nejvyšším dosaženým MCC na nich.

Datasety jsou seřazeny vzestupně, podle velikosti trénovací množiny. Hodnota MCC je nevyšší hodnota změřená na testovací množině modelu natrénovaném na daném datasetu.

Nemyslíme si, že by velikost trénovací množiny neměla vliv na natrénování modelu. Ale usuzujeme, že určité datasety zkrátka byly jednoduší pro předpověď vazebných míst. To může být způsobeno jak výběrem sekvencí v datasetu tak tím, že některé typy vazebných míst mohou být snadnější pro predikci.

Dataset	Embedder	RF MCC	GB MCC	NN MCC
ATP	Xlnet	0.234 (±0.010)	0.283 (±0.016)	0.348 (±0.013)
	ProtBert	0.230 (±0.028)	0.275 (±0.023)	0.354 (±0.024)
	ProtT5	0.361 (±0.007)	0.412 (±0.022)	0.500 (±0.025)
ADP	Xlnet	0.305 (±0.021)	0.373 (±0.031)	0.430 (±0.041)
	ProtBert	0.262 (±0.023)	0.311 (±0.017)	0.444 (±0.029)
	ProtT5	0.424 (±0.031)	0.489 (±0.030)	0.613 (±0.029)
AMP	Xlnet	0.130 (±0.026)	0.171 (±0.028)	0.258 (±0.034)
	ProtBert	0.157 (±0.022)	0.199 (±0.025)	0.242 (±0.033)
	ProtT5	0.222 (±0.021)	0.303 (±0.016)	0.390 (±0.037)
GDP	Xlnet	0.486 (±0.065)	0.513 (±0.069)	0.507 (±0.083)
	ProtBert	0.377 (±0.020)	0.414 (±0.030)	0.546 (±0.052)
	ProtT5	0.591 (±0.067)	0.608 (±0.081)	0.618 (±0.057)
GTP	Xlnet	0.284 (±0.108)	0.314 (±0.127)	0.331 (±0.153)
	ProtBert	0.250 (±0.050)	0.277 (±0.063)	0.316 (±0.095)
	ProtT5	0.410 (±0.111)	0.429 (±0.126)	0.450 (±0.143)
CA	Xlnet	0.120 (±0.023)	0.169 (±0.026)	0.209 (±0.025)
	ProtBert	0.185 (±0.016)	0.234 (±0.019)	0.277 (±0.024)
	ProtT5	0.184 (±0.020)	0.274 (±0.024)	0.331 (±0.019)
MG	Xlnet	0.135 (±0.027)	0.217 (±0.020)	0.285 (±0.023)
	ProtBert	0.286 (±0.019)	0.301 (±0.008)	0.361 (±0.015)
	ProtT5	0.305 (±0.023)	0.343 (±0.012)	0.424 (±0.012)
MN	Xlnet	0.151 (±0.014)	0.246 (±0.020)	0.331 (±0.028)
	ProtBert	0.405 (±0.037)	0.437 (±0.032)	0.465 (±0.027)
	ProtT5	0.430 (±0.027)	0.467 (±0.039)	0.521 (±0.035)
FE	Xlnet	0.276 (±0.039)	0.348 (±0.032)	0.423 (±0.043)
	ProtBert	0.523 (±0.034)	0.540 (±0.019)	0.573 (±0.020)
	ProtT5	0.552 (±0.031)	0.575 (±0.029)	0.624 (±0.026)
ZN	Xlnet	0.304 (±0.011)	0.350 (±0.006)	0.428 (±0.011)
	ProtBert	0.449 (±0.013)	0.466 (±0.006)	0.504 (±0.010)
	ProtT5	0.480 (±0.021)	0.509 (±0.018)	0.565 (±0.013)
DNA	Xlnet	0.156 (±0.032)	0.202 (±0.029)	0.229 (±0.032)
	ProtBert	0.212 (±0.016)	0.286 (±0.019)	0.297 (±0.016)
	ProtT5	0.273 (±0.027)	0.351 (±0.009)	0.364 (±0.011)
HEME	Xlnet	0.286 (±0.051)	0.311 (±0.035)	0.310 (±0.023)
	ProtBert	0.302 (±0.036)	0.338 (±0.044)	0.379 (±0.055)
	ProtT5	0.434 (±0.060)	0.462 (±0.052)	0.477 (±0.048)

Tabulka 4.12: MCC, Datasety rozdělené podle ligandů - trénovací množina

Dataset	Embedder	RF MCC	GB MCC	NN MCC
ATP	Xlnet	0.209	0.301	0.367
	ProtBert	0.231	0.296	0.411
	ProtT5	0.365	0.442	0.545
ADP	Xlnet	0.272	0.311	0.376
	ProtBert	0.222	0.278	0.379
	ProtT5	0.368	0.468	0.517
AMP	Xlnet	0.108	0.174	0.287
	ProtBert	0.177	0.213	0.238
	ProtT5	0.210	0.287	0.426
GDP	Xlnet	0.363	0.412	0.509
	ProtBert	0.310	0.336	0.588
	ProtT5	0.549	0.576	0.691
GTP	Xlnet	0.368	0.346	0.469
	ProtBert	0.270	0.369	0.465
	ProtT5	0.483	0.471	0.573
CA	Xlnet	0.080	0.146	0.195
	ProtBert	0.161	0.207	0.268
	ProtT5	0.158	0.245	0.312
MG	Xlnet	0.161	0.209	0.243
	ProtBert	0.219	0.248	0.287
	ProtT5	0.259	0.287	0.331
MN	Xlnet	0.163	0.214	0.304
	ProtBert	0.382	0.406	0.450
	ProtT5	0.430	0.472	0.523
FE	Xlnet	0.261	0.361	0.408
	ProtBert	0.476	0.514	0.564
	ProtT5	0.591	0.597	0.699
ZN	Xlnet	0.321	0.371	0.437
	ProtBert	0.469	0.465	0.502
	ProtT5	0.508	0.521	0.577
DNA	Xlnet	0.133	0.196	0.242
	ProtBert	0.198	0.247	0.280
	ProtT5	0.274	0.345	0.363
HEME	Xlnet	0.381	0.405	0.450
	ProtBert	0.350	0.351	0.462
	ProtT5	0.580	0.578	0.680

Tabulka 4.13: MCC, datasety rozdělené podle ligandů - testovací množina

4.2.2 Datasety z P2Rank

Reprezentaci reziduí jsme vyzkoušeli také na datasety, které nejsou rozděleny podle ligandů. Konkrétně jako trénovací dataset byl využit dataset *CHEN11* a jako testovací *COACH420*. Díky těmto výsledkům jsme schopni porovnat účinnost reprezentace reziduí pomocí vlastností aminokyselin s reprezentací pomocí embeddingů. V tabulce 4.14 můžeme vidět výsledné *MCC* spolu se standardní směrodatnou odchylkou pro každou zkoumanou metodu a druh embeddingu změřené na trénovací množině. V tabulce 4.15 můžeme vidět výsledné *MCC* na testovací množině. Tyto výsledky jsou použité v sekci 4.4, kde srovnáváme dva zvolené způsoby reprezentace reziduí. Nejlepších výsledků dosáhla *neuronová síť* používající embeddingy *ProtT5*. Pro porovnání, při stejném použití datasetů, jsme na modelu s reprezentací pomocí vlastností aminokyselin změřili hodnotu 0.120 u *MCC*.

Embedder	**Trénovací množina (5-fold CV)**		
	RF MCC	**GB MCC**	**NN MCC**
ProtXlnet	0.153 (±0.050)	0.174 (±0.055)	0.226 (±0.054)
ProtBert	0.257 (±0.038)	0.278 (±0.047)	0.308 (±0.041)
ProtT5	0.341 (±0.065)	0.374 (±0.073)	0.394 (±0.059)

Tabulka 4.14: Porovnání trénovací MCC, embeddingy, dataset *CHEN11*

Embedder	**Testovací množina (COACH420)**		
	RF MCC	**GB MCC**	**NN MCC**
ProtXlnet	0.180	0.198	0.242
ProtBert	0.255	0.280	0.303
ProtT5	0.332	0.366	0.395

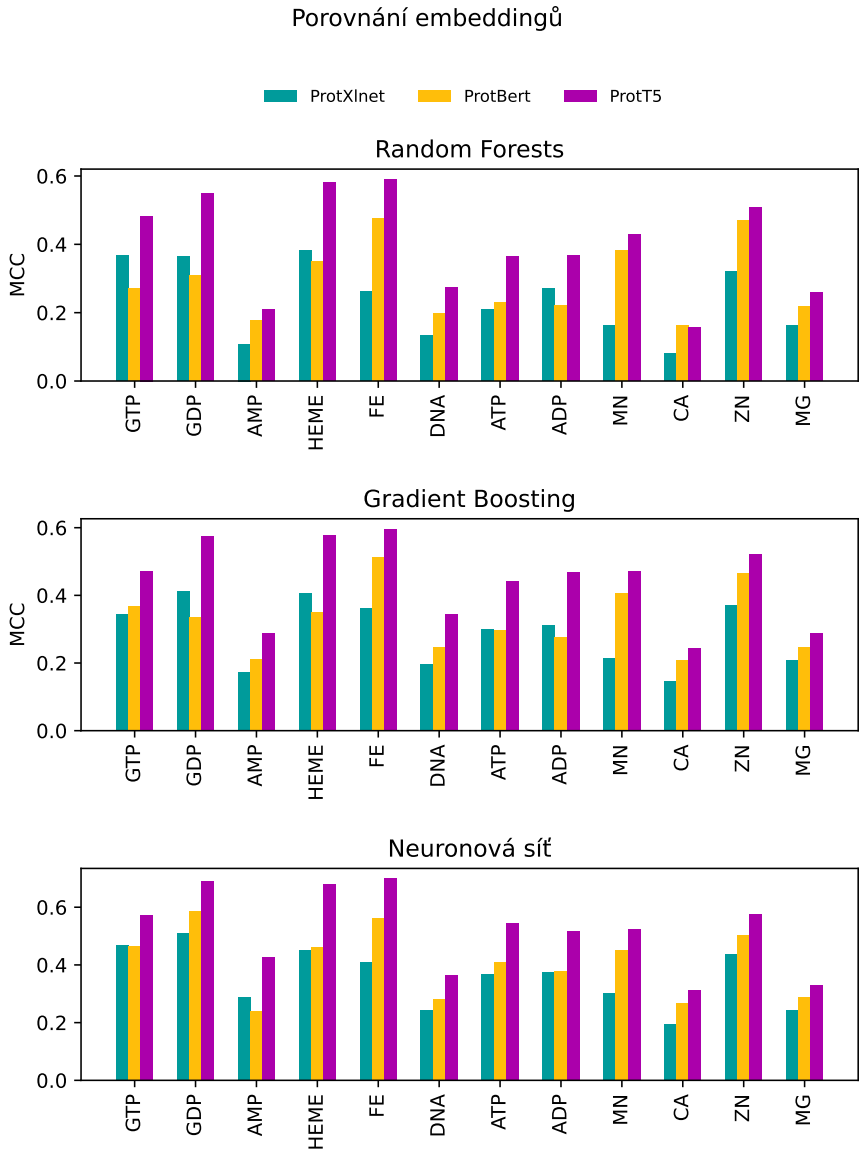
Tabulka 4.15: Porovnání testovacího MCC, embeddingy, dataset *COACH420*

Můžeme si všimnout, že v některých případech ja MCC na testovací množině větší než na trénovací a v dalších případech se tyto dvě hodnoty liší v tisícinách. Když vezmeme v úvahu, že testovací množina a trénovací množina jsou dva odlišné datasety, tak bychom očekávali, že výsledky na trénovací množině budou lepší. Naše výsledky si vysvětlujeme tím, že tyto datasety jsou si svými sekvencemi značně podobné. Také výsledky na trénovací množině byly počítány při cross-validaci, takže 20% z trénovacích dat tvořil validační dataset. Model produkující výsledky na trénovací množině, byl tedy natrénován na pohých 80% trénovacích dat. Zatímco model produkující výsledky na testovací množině byl natrénován sice se stejnými hyperparametry ale na celé trénovací množině. Toto pokládáme za důvody toho proč se výsledky na testovací množině tak přiblížili těm na trénovací.

4.3 Srovnání embeddingů

Pro srovnání typů embeddingů jsme využili výsledky z tabulky 4.13, kde je uveden *MCC* pro modely natrénované na datasetech rozdělených podle ligandů. V

této tabulce můžeme celkem najít výsledky 108 natrénovaných modelů, které jsou znázorněny na grafech na obrázku 4.4. Na zmíněném obrázku jsou tři grafy, každý pro jednu testovanou metodu. V každém grafu jsou pro každý ligand znázorněny výsledky pro použité 3 druhy embeddingů. Což znamená, že máme pro 3 metody a 12 ligandů celkem 36 případů kdy srovnáváme vyzkoušené druhy embeddingů mezi sebou. V každém z 36 případů jsme přiřadili každému typu embeddingů první až třetí místo. To kolikrát se jaký typ embeddingů umístil na jakém místě je uvedeno v tabulce 4.16.



Obrázek 4.4: (MCC)Porovnání embeddingů

Modely co používali embeddingy vygenerované modelem *ProtT5* byly pro daný ligand a metodu nejlepší v 35 případech z 36, jediný případ kdy byly překonány byl u ligandu CA(vápníku), při použití metody *Random Forests*, kde byl překonán modelem používající embeddingy vygenerované *ProtBert* a to o 0.003. Což je velmi malý rozdíl, když si uvědomíme, že standardní směrodatná odchylka při cross-validaci byla 0.02. Tento výsledek je očekávatelný, protože embedder

Embedder	Prvních míst	Druhých míst	Třetích míst
ProtT5	35	1	0
ProtBert	1	25	10
ProtXlnet	0	10	26

Tabulka 4.16: Porovnání embedderů

ProtT5 je nejsložitější a největší z použitých embeddingů. Jak jsme zmínili v sekci 3.1.2, tak je více jak dvakrát větší než ostatní dva embeddery. Jeho výrazně lepší výsledky vysvětlujeme právě jeho velikostí.

Modely používající embeddingy vygenerované pomocí *ProtBert* byly v 26 případech z 36 lepší než ty používající *ProtXlnet*. Mezi *ProtBert* a *ProtXlnet* je malý rozdíl ve velikosti. *ProtBert* zabírá 2.8 GB zatímco *ProtXlnet* 2.7 GB. I tak podle výsledků je vidět, že ve více případech generuje *ProtBert* lepší embeddingy pro předpověď vazebných reziduí. Nemyslíme že by za to mohl rozdíl 0.1 GB ve velikosti spíše tento jev vysvětlujeme tím, že pro naše účelu má *ProtBert* lepší strukturu než *ProtXlnet* a tím pádem generuje lepší embeddingy pro naše účely. To že *ProtBert* je lepší v 26 nepovažujeme za, náhodu, protože pokud by rozdíl mezi *ProtBert* a *ProtXlnet* byl čistě náhodný. Dalo by se předpokládat že s šancí 1/2 je jeden lepší než druhý, avšak šance na to, že jeden bude za tohoto předpokladu 10 a vícekrát lepší z 36 případů je 0.566%. Tento jev bereme jako ukázku toho, že vliv má i struktura embedderu použitého ke generování embeddingů a nejen jeho velikost.

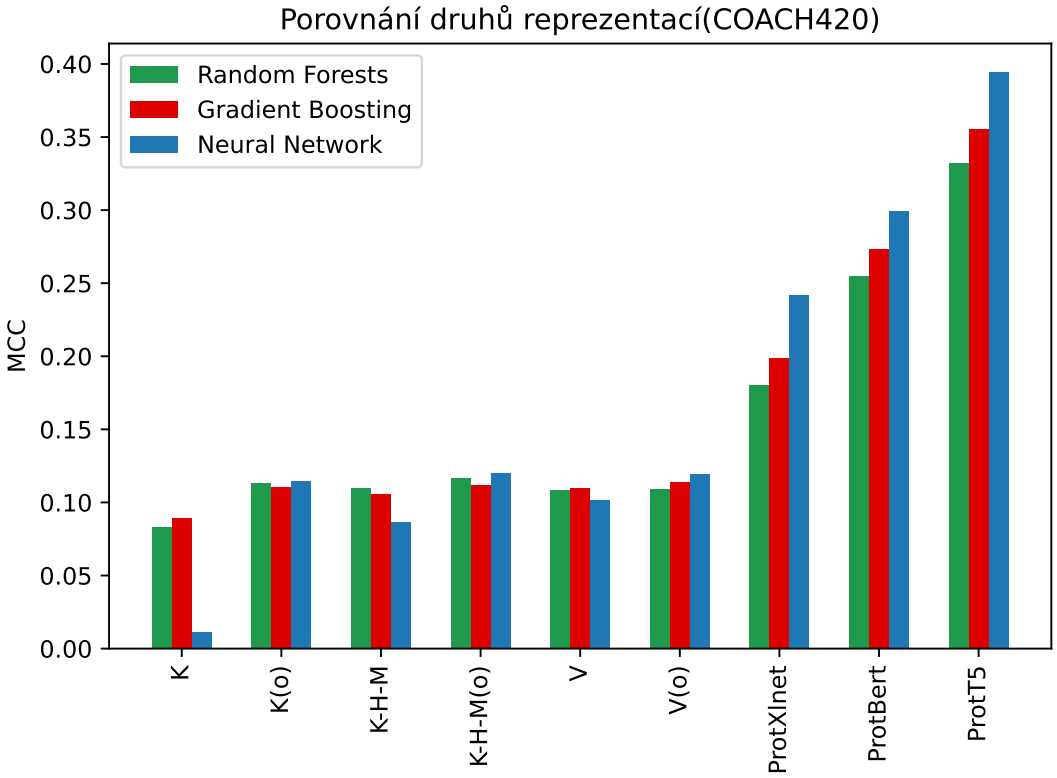
Celkově považujeme *ProtT5* za nejlepší model pro generování embeddingů. Když zprůměrujeme výsledky všech modelů, tak průměrné *MCC* je u *ProtT5* 0.453 (± 0.14), u *ProtBert* 0.341 (± 0.115) a u *ProtXlnet* 0.293 (± 0.109).

4.4 Srovnání způsobů reprezentace reziduí

Nyní porovnáme reprezentaci reziduí pomocí vlastností aminokyselin s reprezentací pomocí embeddingů. K tomuto používáme modely, které byly natrénované na datasetu *CHEN11* a otestované na datasetu *COACH420*. V grafu na obrázku 4.5 je vidět porovnání těchto dvou reprezentací. V grafu je *MCC* změřené na testovací množině pro 6 druhů reprezentace reziduí pomocí vlastností aminokyselin a tři druhy embeddingů. Hodnoty v tomto grafu pocházejí z tabulek 4.7 a 4.15.

Zkratky v grafu znamenají:

- **ProtXlnet**(prottrans_xlnet_uniref100), **ProtBert**(prottrans_bert_bfd), **ProtT5**(prottrans_t5_uniref50) - jsou modely použité pro generování embeddingů.
- **K** - Označuje způsob reprezentace reziduí pomocí vlastností aminokyselin, kde byl jako vlastnost použit, kód aminokyseliny v ordinálním kódování.
- **K(o)** - Označuje způsob reprezentace reziduí pomocí vlastností aminokyselin, kde byl jako vlastnost použit, kód aminokyseliny v *one-hot* kódování.



Obrázek 4.5: (MCC)Porovnání způsobů reprezentací reziduí.

- **K-H-M** - Označuje způsob reprezentace reziduí pomocí vlastností aminokyselin, kde byly použity vlastnosti: kód aminokyseliny, hydropatický idex a hmotnost vše v ordinálním kódování.
- **K-H-M(o)** - Označuje způsob reprezentace reziduí pomocí vlastností aminokyselin, kde byly použity vlastnosti: kód aminokyseliny v *one-hot* kódování, hydropatický idex a hmotnost v ordinálním kódování.
- **V** - Označuje způsob reprezentace reziduí pomocí vlastností aminokyselin, kde byly použity vlastnosti: kód aminokyseliny, třída, polarita, náboj, hydropatický idex, hmotnost a četnost vše v ordinálním kódování.
- **V(o)** - Označuje způsob reprezentace reziduí pomocí vlastností aminokyselin, kde byly použity vlastnosti: kód aminokyseliny, třída, polarita, náboj v *one-hot* kódování, hydropatický idex, hmotnost a četnost v ordinálním kódování.

V grafu na obrázku 4.5 u modelů vyžívajících k reprezentaci vlastnosti aminokyselin je průměrné *MCC* 0.102 ($\pm$ 0.024) a u reprezentací pomocí embeddingů je 0.281 ($\pm$ 0.067). U reprezentací s embedingy vychází téměř třikrát větší směrodatná odchylka, to je způsobeno tím, že jednotlivé typy embeddingů se mezi sebou výrazně liší. Když zprůměrujeme hodnoty pro každý druh embeddingu zvlášť tak získáme pro *ProtXlnet* 0.207 ($\pm$ 0.026), pro *ProtBert* 0.276 ($\pm$ 0.018) a pro *ProtT5* 0.361 ($\pm$ 0.026). vzhledem k výsledkům můžeme říct, že i modely s nejslabším typem embeddingů jsou dvakrát lepší než modely reprezentující rezidua pomocí

vlastností aminokyselin. Z průměrných hodnot pro jednotlivé typy embeddingů, je patrné že i typ embeddingů může výrazně ovlivnit účinnost modelu. To však už je ukázáno v podsekci 4.3. Srovnání *MCC* nejlepších modelů používajících embeddingy a nejlepších modelů používající vlastnosti aminokyselin je vidět v tabulce 4.18.

4.5 Srovnání metod

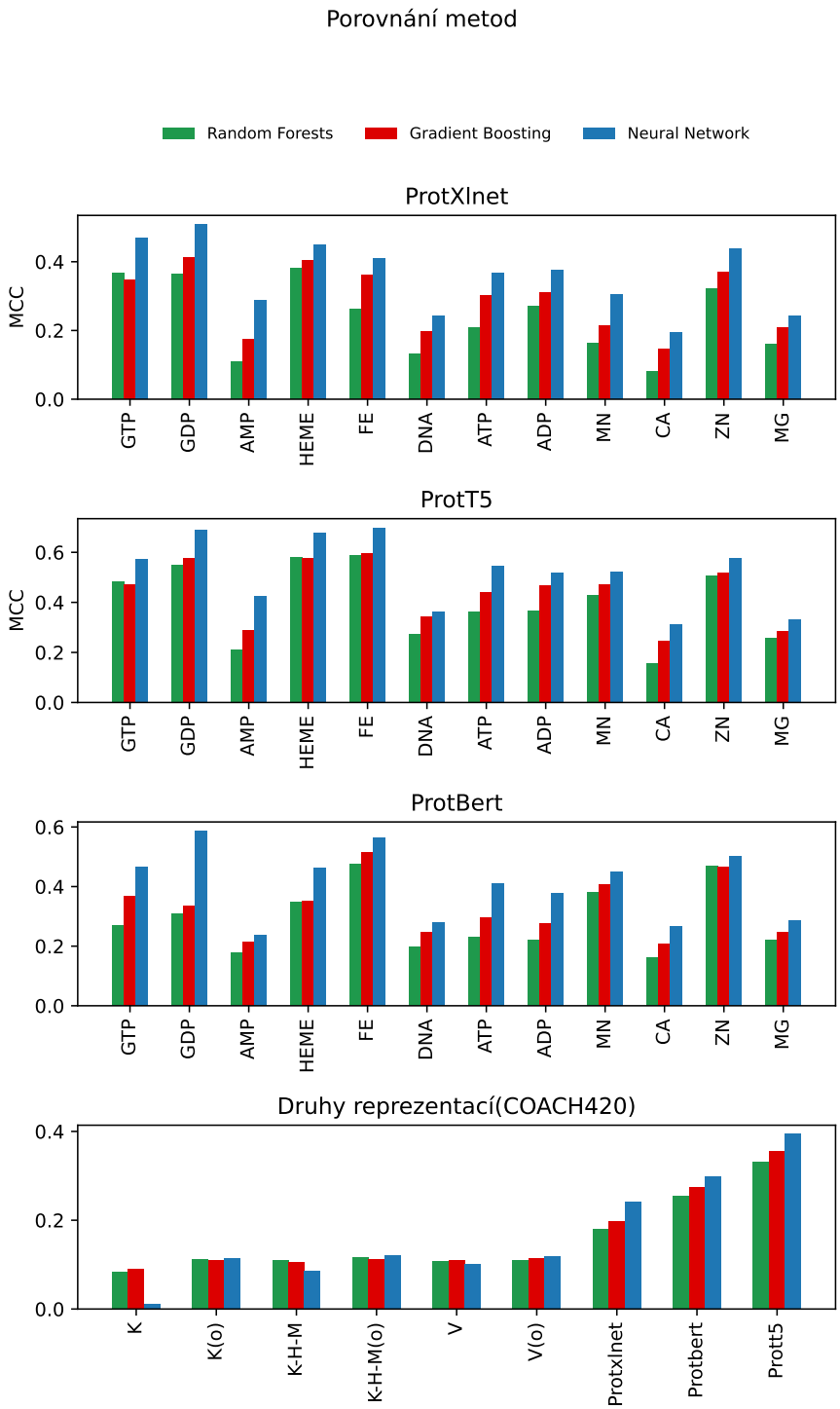
Pro srovnání metod jsme využili výsledky z tabulky s výsledky pro datasety rozdělené podle ligandů 4.13, a z tabulek4.15 a 4.7 s výsledky testů na datasetech z P2Rank. Výsledky z těchto tabulek jsou znázorněny v grafech na obrázku 4.6. Na tomto obrázku jsou 4 grafy znázorňující výsledky z tabulky 4.13 s modely používající datasety rozdělené podle ligandů. Čtvrtý graf znázorňuje výsledky modelů natrénovaných na datasetu *CHEN11* a testovaných na *COACH420*, je to stejný graf jako graf na obrázku 4.5 v sekci 4.4 srovnávající způsob reprezentace reziduí.

Právě ve čtvrtém grafu jsou znázorněny výsledky, které patří modelům používající k reprezentaci vlastnosti aminokyselin(modely v prvních 6 trojsloupcích čtvrtého grafu). Na výsledcích těchto modelů jsme mezi použitými metodami nezaznamenali výrazný rozdíl. Na druhou stranu toto nelze říct o modelech používající k reprezentaci embeddingy. Ve zmíněných grafech můžeme najít 39 případů rozdělených podle datasetů a typu embeddingu. Pro každý případ jsou zde výsledky tří modelů, každý představující jednu testovanou metodu strojového učení. Ve zmíněných 39 případech jsme přiřadili každé metodě první až třetí místo. To kolikrát se jaká metoda umístila na jakém místě je uvedeno v tabulce 4.17.

Metoda	**Prvních míst**	**Druhých míst**	**Třetích míst**
NN	39	0	0
GB	0	35	4
RMF	0	4	35

Tabulka 4.17: Porovnání embedderů

Ve všech zkoumaných případech Byla na prvním místě metoda založená na *neuronové síti(NN)*. Takto výborný výsledek může být dán i díky způsobu získávání embeddingů. Neboť embedder je neuronová síť, ze které byla odstraněna její poslední vrstva, a výstupy z poslední skryté vrstvy jsou námi používané embeddingy. Pokud tyto embeddingy uchovávají reprezentaci jednotlivých reziduí v sekvenci, pak předpokládáme, že tuto znalost dokážou nejlépe využít právě modely založené také na *neuronové síti(NN)*. Průměrná hodnota MCC u modelů založených na *neuronové síti(NN)* je 0.423 ($\pm$ 0.129). U modelů založených na metodě *Gradient Boosting* je to 0.346 ($\pm$ 0.115) a u modelů založených na metodě *Random Forests* je to 0.303 ($\pm$ 0.129).



Obrázek 4.6: (MCC)Porovnání metod

4.6 Srovnání s P2rank

Nyní porovnáme nejlepší modely využívající reprezentaci pomocí vlastností aminokyselin a embeddingů s P2Rank. K porovnání využíváme modely natrénované na množině *CHEN11* a otestované na množině *COACH420*. P2Rank je nástroj využívající znalost 3D struktury proteinu. Jakožto nejmodernější nástroj

tohoto typu dokázal překonat ostatní nástroje založené na hlubokém učení. Proto jej můžeme použít jako zástupce těchto metod.

V tabulce 4.18 najdeme výsledky sedmi různých modelů dohromady zastupující testované metody *Random Forests (RF)*, *Gradient Boosting (GB)*, *neuronová síť (NN)* a P2Rank. Tři modely zastupují nejlepší reprezentaci reziduí pomocí vlastností aminokyselin, každý byl vytvořen pomocí jedné z testovaných metod. Další tři modely zastupují reprezentaci reziduí pomocí embeddingů, každý byl také vytvořen pomocí jedné z testovaných metod a poslední je P2Rank. Můžeme vidět, že v hlavní metrice(*MCC*) P2Rank značně předstihl všechny námi vytvořené modely.

Reprezentace	**Metoda**	**MCC(COACH420)**	**Accuracy**	**Precission**	**Recall**
3D struktura	P2Rank	0.548	0.945	0.473	0.700
embeddingy T5	RF	0.332	0.916	0.474	0.295
	GB	0.355	0.881	0.662	0.247
	NN	0.394	0.892	0.701	0.275
vlastnosti	RF	0.116	0.731	0.486	0.096
	GB	0.112	0.842	0.300	0.115
	NN	0.120	0.829	0.339	0.115

Tabulka 4.18: Porovnání metod

V tabulce také můžeme vidět, že námi vytvořené modely mají nižší *recall* než *precission*, zatímco u P2Rank je tomu naopak. To můžeme být způsobeno tím, že existují dva přístupy k tomu jak prezentovat předpovězená vazebná místa. V prvním přístupu předpovídáme konkrétní vazebné rezidua a díváme se na problém jako na binární klasifikaci. Příkladem metriky použité při tomto přístupu je právě MCC, který využíváme v této práci, protože se zajímáme pouze o předpověď ze sekvence. Druhý přístup můžeme použít, pokud předpovídáme vazebná místa z 3D struktury, jako to dělá P2Rank. Pak můžeme reprezentovat vazebná místa pomocí jejich středů, nebo množiny bodů, popisujících tvar vazebného místa. Následně k vazebných míst s nejlepším hodnocením označíme jako vazebná. Příkladem metriky použité při tomto přístupu je DCCcriterion(vzdálenost centra vazebného místa k nejbližšímu atomu ligandu). Jelikož P2Rank využívá 3D strukturu k předpovědi, tak předpovídá vazebná místa druhým způsobem a to pomocí stanovení středu vazebného místa. Pro porovnání jsme zvolili $k = 1$ protože vede k nejlepším výsledkům. Abychom mohli porovnat natrénované modely s P2Rank, tak musíme i na výsledky z P2Rank pohlížet jako na binární klasifikaci. To znamená že P2Rank označí rezidua, která jsou součástí vazebného místa, jako vazebná. Naneštěstí při tom může označit více reziduí, než je třeba, což vyprodukuje falešně pozitivní vzorky, které snižují výslednou *precission*. A to je důvod, proč můžeme pozorovat takovýto rozdíl mezi výsledným *precission* a *recall* u P2Rank.

Celkově můžeme říct, že modely používající embeddingy se dokázali značně přiblížit k výsledkům P2Rank. Nejlepší náš natrénovaný model používá embeddingy vygenerované pomocí ProtT5 modelu a je založen na metodě neuronových sítí. *MCC* tohoto modelu dosahuje na testovací množině hodnoty 0.394, zatímco u P2Rank dosahuje 0.548. Pro srovnání je dobré si uvědomit, že námi natrénované modely využívají pouze znalosti vycházející z 1D struktury proteinu, zatímco P2Rank vychází ze 3D struktury. Z tohoto důvodu je očekávatelné, že P2Rank

bude lepší. Naproti tomu při srovnání s nejlepším modelem používajícím pouze vlastnosti aminokyselin, který dosahuje *MCC* 0.120. Je náš nejlepší model používající embeddingy více jak třikrát lepší. Z těchto důvodů považujeme výsledek našeho nejlepšího modelu za dobrý.

4.7 Limitace a možné zlepšení

Z časových důvodů nebylo možné v naší práci prozkoumat všechny možné přístupy k predikci protein-ligand vazebných reziduí. Nicméně jsme zde ukázali, že je možné se přiblížit k výsledkům metod využívajících 3D strukturu proteinu i když jen vzdáleně. Proto zde chceme nabídnou myšlenky na možné zlepšení. Jako jednou z možných cest vidíme využívat při tvorbě reprezentace jednotlivých reziduí nejen embeddingů ale i modelů, které predikují 3D strukturu proteinu [45]. V této práci jsme ukázali, že znalost 3D struktury proteinu poskytuje značnou výhodu, proto by nám mohla pomoct i jen její pouhá predikce. Dalším zlepšení by mohl být přesun ke složitějším metodám založeným na neuronových sítích, jako jsou hluboké neuronové sítě. Neboť jak víme, tak náš nejúspěšnější model byla právě neuronová síť. Poslední naší myšlenkou je, že výsledky by mohl zlepšit jiný typ embeddingů, protože to byly právě embedding,y díky kterým jsme dosáhli našich nejlepších výsledků. Bohužel nebylo v našich silách vzkoušet všechny dostupné typy embeddingů.

Závěr

Předpověď vazebných míst na proteinech je důležitým úkolem, dovolujícím nám pochopit interakce mezi proteinem a ligandem, jejichž pochopení je nezbytné při navrhování léčiv nebo v jiných oblastech biologie. Ačkoliv v tomto odvětví již byly vytvořeny nástroje pro predikci vazebných míst, tak stále je zde místo pro zlepšení. Doposud se vytvořené metody zajímali o predikci vazebných míst ze 3D struktury, která však není známá pro většinu známých proteinů. Proto jsme se v naší práci zaměřili na předpověď vazebných míst u proteinů, u kterých známe pouze jejich 1D strukturu. Cílem naší práce nebylo vytvořit nástroj pro předpověď vazebných reziduí, ale prozkoumat možné řešení tohoto problému. Vzhledem k existujícím nástrojům a všeobecné znalosti strojového učení jsme vybrali 3 metody, které jsme mezi sebou porovnávali. Součástí našeho cíle bylo také prozkoumat přístupy k reprezentaci reziduí. V tomto ohledu jsme porovnávali způsob reprezentace vycházející z vlastností aminokyselin se způsobem vycházejícím z oboru zpracování přirozeného jazyka.

Z výsledků naší práce jsme vyvodili, že způsob reprezentace reziduí má zásadní vliv na výkon modelu. Jako dobrý způsob reprezentace se prokázal ten, který používá znalosti z oboru zpracování jazyka. V tomto případě jsme použili k reprezentaci reziduí embeddingy. Všechny vytvořené modely používající embeddingy si vedly několikrát lépe než nejlepší model využívající vlastnosti aminokyselin. Tuto demonstraci rozdílu mezi dvěma zvolenými způsoby reprezentace považujeme za jeden z hlavních přínosů naší práce. Nadále jsme zde demonstrovali vliv druhu použitých embeddingů na předpověď vazebných míst. Ačkoliv zde nebyl tak velký rozdíl, jako mezi způsoby reprezentace reziduí, tak jako nelepší typ embeddingů se ukázaly ty, které byly vygenerované embedderem *prottrans_t5_uniref50*. Tento embedder je nejsložitějším a největším ze zkoumaných embedderů, a to považujeme za klíč k jeho úspěchu.

Ze 3 zvolených metod strojového učení se jako nejlepší mezi modely používající embeddingy k reprezentaci reziduí ukázala *neuronová síť*. Tato metoda vyprodukovala vždy mezi zmíněnými modely nejlepší výsledek. Tento dobrý výsledek přisuzujeme tomu, že na stejné metodě jsou založeny modely generující zmíněné embeddingy. Mezi modely používající k reprezentaci reziduí vlastnosti aminokyselin, jsme z výsledků nebyli schopni určit nejlepší metodu. Avšak výsledky těchto modelů značně zaostávali za ostatními, a proto jsme se jimi více nezabývali.

Dále jsme natrénovali a vyhodnotili modely na datasetech rozdělených podle ligandů, které jsou volně dostupné. U kázali jsme zde, že druh úspěšnost predikce výrazně záleží na druhu ligand vzhledem ke kterému byl daný dataset vytvořen. Tímto jsme také vytvořili možnost porovnat naše výsledky s výsledky v budoucnu zkoušených přístupů používající tyto datasety.

Na závěr jsme porovnali nejlepší námi vytvořený model s P2Rank, který používáme jako zástupce metod využívajících znalost 3D struktury proteinu. Náš nejlepší model dosáhl téměř tří čtvrtin výkonu P2Rank. Fakt že jsme jej nepřekonali nás nepřekvapuje, neboť P2Rank je nejmodernějším nástrojem v tomto odvětví. Na druhou stranu považujeme takovýto výsledek našeho modelu za výborný, protože ukazuje, že předpověď vazebných míst z 1D struktury má smysl

u proteinů s neznámou 3D strukturou. Tuto demonstraci možné využitelnosti metod využívajících pouze znalost 1D struktury považujeme za nejhodnotnější výsledek v naší práci.

Seznam použité literatury

[1] The UniProt Consortium. UniProt: the Universal Protein Knowledgebase in 2023. *Nucleic Acids Research*, 51(D1):D523–D531, 11 2022.

[2] Wikipedia. Amino acid — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Amino%20acid&oldid=1150022646`, 2023. [Online; accessed 16-April-2023].

[3] Cvrčková Fatima. *Úvod do praktické bioinformatiky*. Academia, Praha, 2006.

[4] Anne Marie Helmenstine. What is an aliphatic hydrocarbon? review your chemistry concepts. `https://www.thoughtco.com/definition-of-aliphatic-hydrocarbon-604763`, August 2010. Accessed: 2023-4-17.

[5] Glendale Community College. Chapter 8 – nomenclature. `https://web.gccaz.edu/~kimld88531/chm130lec_files/Ch%208%20OER.pdf`. Accessed: 17-4-2022.

[6] Anne Marie Helmenstine. Amide definition and examples in chemistry. `https://www.thoughtco.com/definition-of-amide-604772`, February 2012. Accessed: 2023-4-17.

[7] Anne Marie Helmenstine. Chemistry basics: What is an anion? `https://www.thoughtco.com/definition-of-anion-and-examples-604344`, December 2008. Accessed: 2023-4-17.

[8] Henderson James (Jim) Cleaves. *Thiol*, pages 1668–1668. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[9] Anne Marie Helmenstine. Cation definition and examples. `https://www.thoughtco.com/cation-definition-and-examples-602142`, July 2003. Accessed: 2023-4-17.

[10] Anne Marie Helmenstine. Functional groups in organic chemistry. `https://www.thoughtco.com/functional-groups-in-organic-chemistry-4054178`, April 2009. Accessed: 2023-4-17.

[11] Michal Filippi. Predikce sekundární struktury proteinu pomocí hlubokých neuronových sítí, 2017.

[12] C Anfinsen. Principles that govern the folding of protein chains. " science. *Science*, 181:223–230, 1973.

[13] EMBL-EBI. Current release statistics < uniprot < embl-ebis — ebi.ac.uk. `https://www.ebi.ac.uk/uniprot/TrEMBLstats`, 2023. [Accessed 18-Apr-2023].

[14] D. Voet, J.G. Voet, and C.W. Pratt. *Voet's Principles of Biochemistry*. Wiley, 2018.

[15] Marcia A. Kaetzel and John R. Dedman. Calmodulin. In Helen L. Henry and Anthony W. Norman, editors, *Encyclopedia of Hormones*, pages 241–245. Academic Press, New York, 2003.

[16] D. Voet, J.G. Voet, and C.W. Pratt. *Voet's Principles of Biochemistry*. Wiley, 2018.

[17] wwPDB consortium. Protein data bank: the single global archive for 3D macromolecular structure data. *Nucleic Acids Res.*, 47(D1):D520–D528, January 2019.

[18] Radoslav Krivák and David Hoksza. P2rank: machine learning based tool for rapid and accurate prediction of ligand binding sites from protein structure. *Journal of Cheminformatics*, 10(1):39, Aug 2018.

[19] Tom Mitchell. *Machine Learning*. McGraw Hill, 1997.

[20] Davide Chicco and Giuseppe Jurman. The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC Genomics*, 21(1):6, Jan 2020.

[21] Lukas Jendele, Radoslav Krivak, Petr Skoda, Marian Novotny, and David Hoksza. PrankWeb: a web server for ligand binding site prediction and visualization. *Nucleic Acids Res*, 47(W1):W345–W349, July 2019.

[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and Duchesnay. sklearn.ensemble.randomforestclassifier — scikit-learn.org. `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`, 2011. [Accessed 02-May-2023].

[23] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and Duchesnay. sklearn.ensemble.HistGradientBoostingClassifier — scikit-learn.org. `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.HistGradientBoostingClassifier.html`, 2011. [Accessed 02-May-2023].

[24] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Neural networks — pytorch tutorials 2.0.0+cu117 documentation — pytorch.org. `https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html`, 2017. [Accessed 02-May-2023].

[25] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794, New York, NY, USA, 2016. Association for Computing Machinery.

[26] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. torch.nn — pytorch 2.0 documentation — pytorch.org. https://pytorch.org/docs/stable/nn.html, 2017. [Accessed 25-pr-2023].

[27] Robert Kwiatkowski. Gradient descent algorith a deep dive. https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21, 2021. [Accessed 02-May-2023].

[28] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[29] Austin a Marina DANILEVSKY EOVITO. *Language Models in Plain English [online]*. O'Reilly Media, Inc.

[30] Ehsaneddin Asgari and Mohammad R K Mofrad. Continuous distributed representation of biological sequences for deep proteomics and genomics. *PLoS One*, 10(11):e0141287, November 2015.

[31] Michael Heinzinger, Ahmed Elnaggar, Yu Wang, Christian Dallago, Dmitrii Nechaev, Florian Matthes, and Burkhard Rost. Modeling aspects of the language of life through transfer-learning protein sequences. *BMC Bioinformatics*, 20(1):723, Dec 2019.

[32] Ahmed Elnaggar, Michael Heinzinger, Christian Dallago, Ghalia Rihawi, Yu Wang, Llion Jones, Tom Gibbs, Tamas Feher, Christoph Angerer, Martin Steinegger, Debsindhu Bhowmik, and Burkhard Rost. Prottrans: Towards cracking the language of life's code through self-supervised deep learning and high performance computing, 2021.

[33] Lena Voita. Nlp course | for you — lena-voita.github.io. https://lena-voita.github.io/nlp_course.html. [Accessed 03-May-2023].

[34] Ahmed Elnaggar, Michael Heinzinger, Christian Dallago, Ghalia Rehawi, Yu Wang, Llion Jones, Tom Gibbs, Tamas Feher, Christoph Angerer, Martin Steinegger, Debsindhu Bhowmik, and Burkhard Rost. Prottrans: Towards cracking the language of life's code through self-supervised deep learning and high performance computing. *bioRxiv*, 2020.

[35] bio_embeddings.embed - bio_embeddings — docs.bioembeddings.com. https://docs.bioembeddings.com/v0.2.3/api/bio_embeddings.embed.html. [Accessed 04-May-2023].

[36] Github - rdk/p2rank-datasets: Datasets for p2rank project. https://github.com/rdk/p2rank — github.com. https://github.com/rdk/p2rank-datasets. [Accessed 04-May-2023].

[37] Ke Chen, Marcin J. Mizianty, Jianzhao Gao, and Lukasz Kurgan. A critical comparative assessment of predictions of protein-binding sites for biologically relevant organic compounds. *Structure*, 19(5):613–621, 2011.

[38] Ambrish Roy, Jianyi Yang, and Yang Zhang. COFACTOR: an accurate comparative algorithm for structure-based protein function annotation. *Nucleic Acids Research*, 40(W1):W471–W477, 05 2012.

[39] Janez Konc and Dušanka Janežič. Binding site comparison for function prediction and pharmaceutical discovery. *Current Opinion in Structural Biology*, 25:34–39, 2014. Theory and simulation / Macromolecular machines.

[40] Dong-Jun Yu, Jun Hu, Jing Yang, Hong-Bin Shen, Jinhui Tang, and Jing-Yu Yang. Designing template-free predictor for targeting protein-ligand binding sites with classifier ensemble and spatial clustering. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 10(4):994–1008, 2013.

[41] Hamza Gamouh. Github - hamzagamouh/protein_embeddings — github.com. `https://github.com/hamzagamouh/protein_embeddings`. [Accessed 08-May-2023].

[42] clusters. Sign in gitlab — gitlab.mff.cuni.cz. `https://gitlab.mff.cuni.cz/mff/hpc/clusters`, 2023. [Accessed 08-May-2023].

[43] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and Duchesnay. 3.1. Cross-validation: evaluating estimator performance — scikit-learn.org. `https://scikit-learn.org/stable/modules/cross_validation.html`, 2011. [Accessed 08-May-2023].

[44] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[45] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, Aug 2021.

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

- **RMF**(Random forests) - Je používaná v tabulkách, kde označuje sloupce s výsledky modelů, kde byla použita metoda strojového učení *Random forest*.
- **GB**(Gradient Boosting) - Je používaná v tabulkách, kde označuje sloupce s výsledky modelů, kde byla použita metoda strojového učení *Gradient Boosting*.
- **NN**(Neural Network) - Je používaná v tabulkách, kde označuje sloupce s výsledky modelů, kde byla použita metoda strojového učení *Neuronová síť*.
- **MCC**(Matthews correlation coefficient) - Je hlavní metrika používaná pro porovnání vytvořených modelů.
- **ProtXlnet**(prottrans_xlnet_uniref100) - Je model použitý pro generování embeddingů.
- **ProtBert**(prottrans_bert_bfd) - Je model použitý pro generování embeddingů.
- **ProtT5**(prottrans_t5_uniref50) - Je model použitý pro generování embeddingů.
- **K** - Označuje způsob reprezentace reziduí pomocí vlastností aminokyselin, kde byl jako vlastnost použit, kód aminokyseliny v ordinálním kódování.
- **K(o)** - Označuje způsob reprezentace reziduí pomocí vlastností aminokyselin, kde byl jako vlastnost použit, kód aminokyseliny v *one-hot* kódování.
- **K-H-M** - Označuje způsob reprezentace reziduí pomocí vlastností aminokyselin, kde byly použity vlastnosti: kód aminokyseliny, hydropatický idex a hmotnost vše v ordinálním kódování.
- **K-H-M(o)** - Označuje způsob reprezentace reziduí pomocí vlastností aminokyselin, kde byly použity vlastnosti: kód aminokyseliny v *one-hot* kódování, hydropatický idex a hmotnost v ordinálním kódování.
- **V** - Označuje způsob reprezentace reziduí pomocí vlastností aminokyselin, kde byly použity vlastnosti: kód aminokyseliny, třída, polarita, náboj, hydropatický idex, hmotnost a četnost vše v ordinálním kódování.
- **V(o)** - Označuje způsob reprezentace reziduí pomocí vlastností aminokyselin, kde byly použity vlastnosti: kód aminokyseliny, třída, polarita, náboj v *one-hot* kódování, hydropatický idex, hmotnost a četnost v ordinálním kódování.

A. Přílohy

A.1 Kód a výsledky

Jak už bylo zmíněno v sekci 3.5.2 tak kód je na GitHub repozitáři na:`https://github.com/ProkopDivin/PBSPrediction/tree/main/pbsPrediction`

Výsledky spolu s hyperparametry jsou ve stejném repozitáři a to konkrétně na: `https://github.com/ProkopDivin/PBSPrediction/tree/main/pbsPrediction/finalResults`

V repozitáři jsou vždy složky README.md, která popisují věci v něm konkrétněji, případně se v nich nachází návody pro spuštění programu.