Universitat de Barcelona

Facultat d'Economia i Empresa

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Facultat de Matemàtiques i Estadística

Treball Final de Grau

Classificació de proteïnes segons on es troben a la cel · lula mitjançant tècniques de *Machine Learning*

Grau d'Estadística

Autor: Àurea Cartanyà Hueso

Director: Esteban Vegas Lozano

Convocatòria: Juny 2015

Resum

La predicció de la funció d'una proteïna és un dels problemes més desafiadors en l'era post genomica. El nombre de noves proteïnes identificades ha augmentat exponencialment amb els avenços de les tècniques d'alt rendiment. Encara que, la caracterització funcional d'aquestes noves proteïnes no ha augmentat en la mateixa proporció.

Tenint en compte aquest punt, aquest treball proporciona una descripció comprensible de l'enfocament de l'aprenentatge auomàtic, que s'apliquen actualment en problemes de classificció de proteïnes i predicció de la funció. En aquest treball principalment es farà servir el mètode de *support vector machine* per realitzar la classificació i la predicció.

Primer de tot es defineix que és una proteïna i perquè és important tenir una classificació de proteïnes, llavors es presenta l'aprenentatge automàtic per cadenes de caràcters i algunes eines útils per treballar-lo en R. Un cop es sap que és l'aprenentatge automàtic i les seves característiques, l'objectiu es exposar la classificació de proteïnes i la seva predicció, tal com s'ha fet en l'apartat de l'aprenentatge automàtic es proposen algunes eines útils per poder-ho treballar amb R. Finalment es fa una classificació de proteïnes segons on es troben a la cèl · lula i la seva predicció amb una base de dades proteòmica real.

Paraules clau: *proteïna dades òmiques*, *aprenentatge automàtic*, *classificació de proteïnes*, *predicció de proteïnes*, *localització subcellular*.

Abstract

Protein function prediction is one of the most challenging problems in the post-genomic era. The number of newly identified proteins has been exponentially increasing with the advances of the high-throughput techniques. However, the functional characterization of these new proteins was not incremented in the same proportion.

Regarding this point, this work provides a comprehensible descriptions of machine learning approaches that are currently applied to protein classification and protein function prediction problems. In this work we mainly use the support vector machine method to make the classification and prediction.

We start by definig what is a protein and why is important to have a protein classification, then we present the machine learing for strings and some useful tools for to work it in R. Once we know what is the machine learning and its features, we aim to expose what is the protein classification and the protein prediction and as in machine learning section we propose some useful tools for to work it in R. Finally we make a protein classification according on its subcellular location and protein prediction for a real proteomic database.

Keywords: *protein*, *omics data*, *machine learning*, *protein classification*, *protein prediction*, *subcellular location*.

Índex

Índex de figures

Notacions

x	Valor escalar
$\mathbf{x}$	Vector columna
$\mathbf{x}^T$	Vector transposat
$\mathbf{X}$	Matriu
$\mathbf{X}^{-1}$	Matriu inversa
n	Nombre d'observacions
p	Nombre de variables
m	Combinació de strings de longitud k
$\mathcal{X}$	Espai d'entrada o *input space*
$\mathcal{M}$	Espai de característiques o *feature space*
$N(m, \mathbf{x})$	Nombre d'aparicions de la combinació m en la seqüència $\mathbf{x}$
$k(\mathbf{x}, \mathbf{y})$	Funció kernel
$\mathbf{K}$	Matriu kernel
$\mathbb{R}$	Espai de Reals
$\mathbf{1}(m, \mathbf{x}, l)$	Específica funció indicadora kernel
$E(p, q)$	Funció distància ponderada
v	Variable predictiva
c	Etiqueta

Capítol 1

INTRODUCCIÓ

1.1 Introducció

El present document constitueix la memòria del treball de fi de grau del grau interuniversitari (UB-UPC) d'Estadística. En aquest capítol s'exposarà la justificació, els objectius, la metodologia i l'estructura del treball.

1.1.1 Justificació

Saber la funció que fan les proteïnes és una qüestió clau per entendre com funciona l'organisme dels éssers vius.
Es pot veure que en els últims 10 anys el nombre de seqüències d'aminoàcids i el nombre d'estructures 3D de proteïnes disponibles en bases de dades publiques ha augmentat exponencialment, de fet aquesta gran quantitat de dades exigeixen un gran esforç d'investigació, i tot això per aconseguir resposta a una pregunta fonamental en el món de la Biologia: Quina funció realitza cada proteïna?
Augmentar el coneixement de caracterització funcional biològica de noves proteïnes pot afectar directament al desenvolupament de nous fàrmacs, ja que moltes malalties es produeixen com a conseqüència de l'alteració de la funció de la proteïna en qüestió. Cal destacar que el 40% de les proteïnes disponibles al National Centre for Biothecnology Information (NCBI)[25] no tenen assignada una funció.

1.1.2 Objectius

L'objectiu principal d'aquest treball és realitzar una classificació de proteïnes segons on es troben localitzades a la cèl · lula.
La classificació de les proteïnes es farà mitjançant una tècnica de classificació en *machine learning* anomenada *support vector machine* i les dades s'extreuran d'una base de dades de proteïnes com potser uniprot[16].

A més de l'objectiu final, que és fer la classificacio, el treball té un seguit de passos intermedis, que al mateix temps són objectius necessaris per poder arribar a crear la classificació. Aquests objectius són:

- Conèixer com es produeixen les proteïnes i quina és la seva importància en els organismes vius.
- Saber tractar dades de tipus proteòmic i quins són els seus problemes a l'hora de tractar-les.
- Conèixer que són les tècniques *machine learning* i amb més profunditat mètodes kernel per *strings*.
- Conèixer que són les tècniques de classificació binària i amb més profunditat *support vector machine*.
- Fer una classificació a una base de dades real.

1.1.3 Metodologia

Aquest treball de fi de grau ha estat bàsicament un treball d'aprenentatge, ja que les tècniques d'aprenentatge automàtic no es veuen durant el grau. La idea d'aquest treball ha estat extreta principalment del article *A review of protein function prediction under Machine learning prospective*[3], per tant molta informació ha estat extreta d'aquest article. A més a més, s'han fet servir recursos bibliogràfics i on-line referents als temes d'aprenentatge automàtic, biologia i bioinformàtica. Pel que fa a l'aplicació, s'ha utilitzat una base de dades proteòmica extreta de la font uniprot.

1.1.4 Estructura

Aquesta memòria està composta de sis capítols incloent aquest que correspon a la introducció. En aquest primer capítol s'introdueix la temàtica d'aquest treball de fi de grau, els principals objectius, la metodologia que s'ha seguit i finalment en aquest apartat l'estructura de la memòria amb una breu introducció de cada capítol.

El segon capítol està dividit en dos parts. En la primera d'aquestes es presenten les principals idees de la biologia mol · lecular així com la producció de proteïnes i la importància en la seva classificació, com ja s'ha esmentat anteriorment, tenir una classificació completa ajudaria a tenir grans avenços en el món de la medicina. En la segona part del segon capítol s'expliquen les idees principals de la bioinformàtica, i les dades òmiques així les seves principals característiques.

El tercer capítol també està comprés amb dos parts. En la primera part s'expliquen les idees principals del *machine learning* i les funcions kernel de forma teòrica, a partir de que s'han descrit els conceptes anteriors s'aprofundeix amb les funcions kernels per *strings*. Pel que fa a la segona part d'aquest capítol s'explica l'aplicació de les funcions kernel per *strings* amb el software R mitjançant els paquets **kernlab**[9] i **kebabs**[4].

Pel que fa al quart capítol està estructurat de la mateixa que el capítol anterior. En la primer part s'expliquen les idees principals de classificació en el context de *Machine learning* i amb més profunditat el mètode de classificació supervisada *support vector machine*. En la segona part d'aquest capítol s'explica com fer una classificació amb el mètode *support vector machine* implementat en els paquets d'R **kernlab** i **kebabs**.

En el cinquè capítol es realitza una classificació d'una base de dades proteòmica real, per tant en aquest capítol s'utilitzen les funcions implementades en els paquets **kernlab** i **kebabs** i explicades, tant per les funcions kernel per *strings* com pel *support vector machine*, en els capítols anteriors.

Finalment, en l'últim capítol de la memòria s'exposen les principals conclusions que s'han extret al llarg de la realització d'aquest document. Seguidament es troben totes les referències consultades per l'elaboració d'aquest treball i a l'annex s'adjunta el codi R utilitzat en els diferents apartats del projecte.

Capítol 2

BIOLOGIA I BIOINFORMÀTICA

2.1 Introducció a la Biologia Molecular

En el 1953 es van publicar tres articles sota el títol: *Molecular structure of nucleic acids*[7]. En un d'ells, dos científics, Watson i Crick van proposar la estructura de la doble hèlix de l'ADN, però no va ser fins al 1954 quan George Gamow va proposar que la seqüència de nucleòtids formava un codi.
D'aquesta manera es va descobrir la importància genètica que conté l'ADN a nivell d'informació.

2.1.1 Transferència d'informació genètica

El terme transferència d'informació genètica fa referència al procés en que un organisme comparteix informació genètica amb un altre organisme. Hi ha dos maneres de transferir aquesta informació, la vertical i la horitzontal. Es parla de transferència genètica vertical, quan un organisme rep material genètic dels seus antecessors per exemple dels seus pares o d'un antecessor del que ha evolucionat, en canvi, la transferència genètica horitzontal, és el procés en que un organisme transfereix material genètic a una altra cèl · lula que no és descendent.

Un dels processos biològics implicats en la transmissió d'informació genètica és l'expressió genètica que ve explicada pel dogma central de la biologia molecular.

2.1.2 Diferència entre l'ADN i l'ARN

L'ADN o àcid desoxiribonucleic, és un àcid nucleic que conté les instruccions genètiques utilitzades en el desenvolupament i funcionament de tots els éssers vius coneguts, així com en alguns virus. La funció principal de les molècules d'ADN és l'emmagatzematge d'informació a llarg termini.
L'ARN o àcid ribonucleic és un polimer que consisteix en una cadena llarga de nucleòtids

units entre ells. La funció principal de l'ARN és la síntesi de proteïnes.
Cal destacar que tant la cadena d' ADN com la cadena d'ARN estan formades per una base nitrogenada, un sucre i una sèrie de grups fosfats. Les diferències que hi ha entre elles són que l'ADN presenta com a bases nitrogenades l'adenina, la guanina, la citosina i la timina i com a sucre la desoxiribosa, en canvi, l'ARN presenta l'uracil en comptes de la timina i pel que fa al sucre la ribosa en comptes de la desoxiribosa.

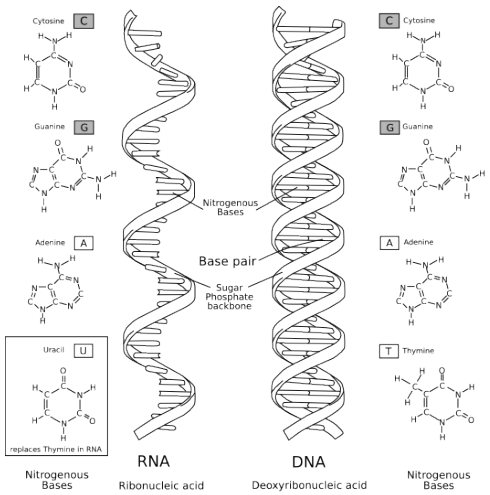


Figura 2.1: Diferències entre l'ADN i l'ARN [29]

2.1.3 Dogma central de la biologia molecular

El dogma central de la biologia molecular es basa en transferir la informació codificada en l'ADN a proteïnes a través de l'ARN. Aquest dogma forma la columna vertebral de la biologia molecular i està format per quatre etapes principals: replicació, transcripció, processament i traducció.

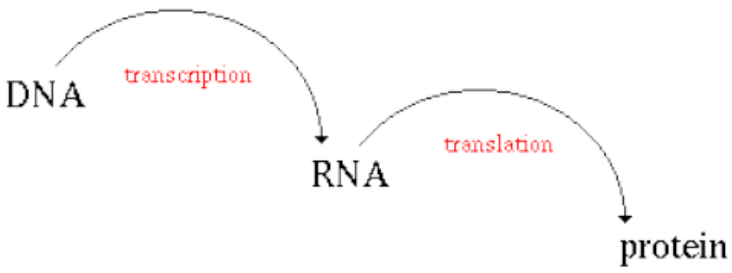


Figura 2.2: Dogma central biologia molecular [15]

- **La replicació [21]:** Es podria resumir amb una frase, com el procés en el qual la cadena d'ADN fa una copia complementària de si mateixa.

Si es fa una descripció més detallada del procés, primer de tot la doble cadena d'ADN es separa. Seguidament es crea una cadena complementària d'ADN a cada una de les cadenes originals.
Finalment les cadenes —la nova i la vella— s'ajunten i és finalitza el procés de la replicació.

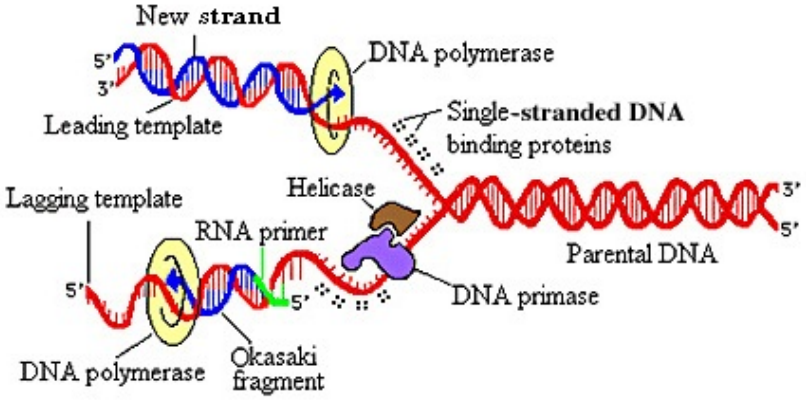


Figura 2.3: La replicació [21]

- **La transcripció [22]:** es resumeix com el procés en que la cadena d'ADN es transcriu en una cadena complementària d'ARN, mitjançant l'ARN polimerasa.

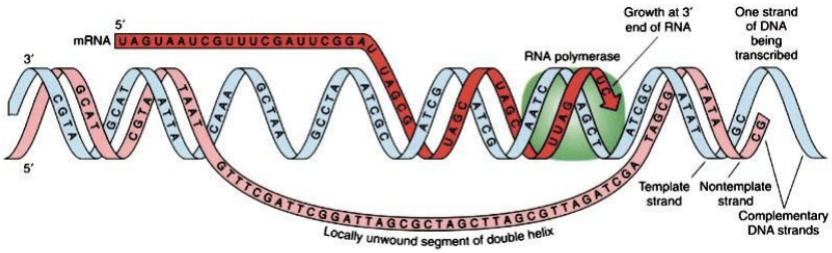


Figura 2.4: La transcripció [15]

- **El processament [20]:** Un primer pas és eliminar els introns de la cadena d'ARN per deixar només els exons, aquest ARN final s'anomena ARN missatger, en forma abreujada ARN(m), ja que serà el "transportador"de la informació genètica. Després aquest ARN(m) surt del nucli i viatja al citoplasma, on es troba amb uns orgànuls cel · lulars anomenats ribosomes, on es produïran les proteïnes.

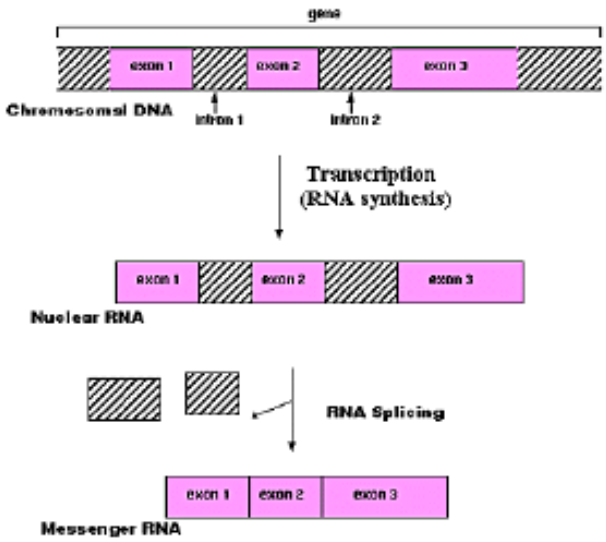


Figura 2.5: El processament [20]

- **La traducció [22]:** Aquesta és la etapa final, la qual és on es produeixen els aminoàcids que compondran la seqüència de la proteïna.
  Primerament el ribosoma s'uneix a l'ARN(m), exactament en el codó d'inici, aquest és el codó (AUG) que és reconegut només per l'iniciador ARN(t). El ribosoma procedeix a la fase d'elongació de la síntesi de proteïnes, durant aquesta etapa complexos compostos d'un aminoàcid lligat a l'ARN(t) s'uneixen seqüencialment al codó apropiat en l'ARN(m) per formar parelles de bases complementàries amb l'anticodó d'ARN(t). El ribosoma es mou de codó a codó al llarg de l'ARN(m) "traduïnt" i així formant els aminoàcids que s'afegeixen a les seqüències polipeptídiques dictades per l'ADN i representades per l'ARN(m). Finalment un factor d'alliberament s'uneix al codó de parada acabant la traducció i alliberant la cadena completa del ribosoma.

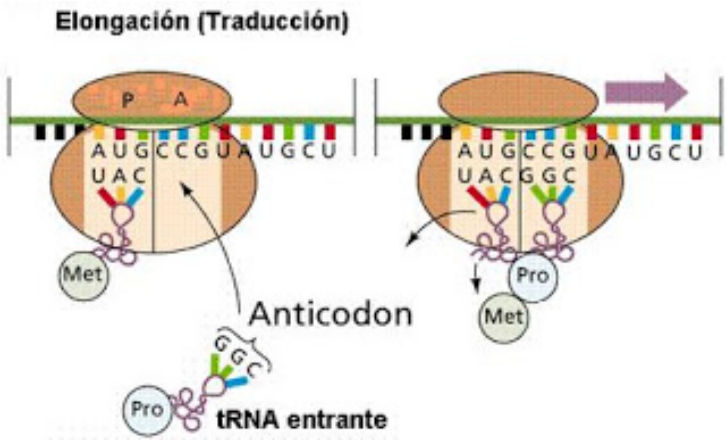
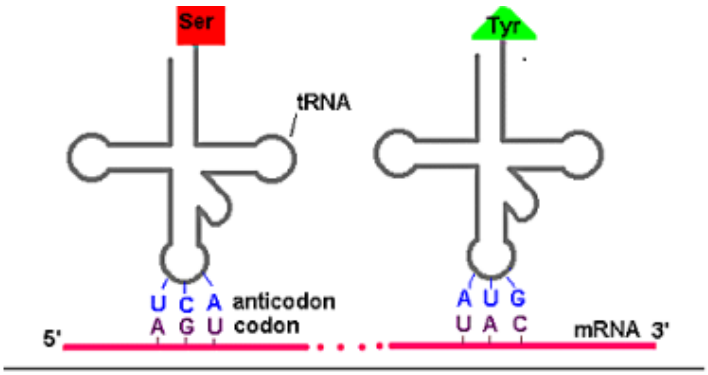


Figura 2.6: La traducció [8]

2.1.4 Proteïnes

Les proteïnes són grans molècules biològiques o macromolècules, que consisteixen en una o més cadenes de residus d'aminoàcids —existeixen 20 aminoàcids diferents—. Les proteïnes són caracteritzades per realitzar una àmplia gama de funcions dins dels organismes vius per exemple catalitzar reaccions metabòliques, la replicació de l'ADN, el transport de molècules d'un lloc a un altre, entre d'altres. Les proteïnes es difereixen d'unes a altres principalment per la seva seqüència d'aminoàcids.



2nd base in codon

1st base in codon	U	C	A	G	3rd base in codon
U	Phe	Ser	Tyr	Cys	U
	Phe	Ser	Tyr	Cys	C
	Leu	Ser	STOP	STOP	A
	Leu	Ser	STOP	Trp	G
C	Leu	Pro	His	Arg	U
	Leu	Pro	His	Arg	C
	Leu	Pro	Gln	Arg	A
	Leu	Pro	Gln	Arg	G
A	Ile	Thr	Asn	Ser	U
	Ile	Thr	Asn	Ser	C
	Ile	Thr	Lys	Arg	A
	Met	Thr	Lys	Arg	G
G	Val	Ala	Asp	Gly	U
	Val	Ala	Asp	Gly	C
	Val	Ala	Glu	Gly	A
	Val	Ala	Glu	Gly	G

Figura 2.7: El codi genètic [19]

2.1.5 Com es determina la funció de la proteïna?

El terme "funció d'una proteïna" fa referència a les funcions moleculars d'una proteïna, com per exemple regulació de gens, transport de materials, catalització de reaccions bioquímiques, entre altres.

La representació de diferents aspectes de la funció de les proteïnes, vocabularis especials per estandarditzar l'anotació funcional i facilitar el processament computacional, han estat proposats en varis esquemes de classificació. Un dels més utilitzats és el *Gene Ontology*

(GO)[24], tot i que aquesta font conté molta més informació a part de la que conté sobre la funció de les proteïnes.

GO descriu tres aspectes de la funció de les proteïnes:

- La funció molecular.
- El procés biològic.
- La localització cel · lular.

Els mètodes computacionals més populars per la identificació de la funció de les proteïnes són basats en seqüències d'aminoàcids —des de que aquesta informació es disponible per la majoria de les proteïnes existents—. Assignar la funció de proteïnes desconegudes a partir d'altres proteïnes conegudes semblants és la estratègia més comuna per la predicció de la funció d'una proteïna.

2.1.6 Localització sub-cel · lular

Les proteïnes són sintetitzades en el citoplasma i normalment es traslladen als orgànuls, membrana cel · lular o fora de la cèl · lula, per realitzar la seva funció. Moltes vegades conèixer la localització sub-cel · lular de les proteïnes desconegudes pot ser d'ajuda per predir la seva funció.

Determinar la localització sub-cel · lular de manera experimental pot ser molt costós, per tant s'estan posant molts esforços per desenvolupar recursos informàtics per predir-la. La predicció de la localització sub-cel · lular de la proteïna pot ser formulat com un problema de classificació, sent l'entrenament positiu les seqüències de proteïnes amb localització sub-cel · lular particular, mentre que l'entrenament negatiu és quan les proteïnes prenen una altra localització. Alguns mètodes estan especialitzats en donar la localització sub-cel · lular, però també existeixen uns altres que són d'àmplia gamma i cobreixen les principals ubicacions.

La majoria dels mètodes existents assumeixen per simplicitat que les proteïnes només tenen una localització sub-cel · lular, però avui en dia ja hi ha suficients evidències que indiquen l'augment del nombre de proteïnes amb multilocalització.

2.2 Bioinformàtica

La bioinformàtica es defineix com, la creació i l'aplicació de tècniques informàtiques a l'estudi de la informació genètica o bioquímica. Els conjunts de dades que es fan servir en la bioinformàtica s'anomenen "dades òmiques".

2.2.1 Dades òmiques

El terme dades òmiques fa referència al conjunt de dades relacionades amb la biologia molecular o bioquímica. Exemples d'aquests conjunts de dades podrien ser la genòmica que conté informació sobre gens, la proteòmica que conté informació de proteïnes. Aquestes tenen en comú que es basen en l'anàlisi d'un gran volum de dades biològiques moleculars i fan ús de la bioinformàtica per la seva interpretació [11].

2.2.2 Tecnologies d'alt rendiment

El terme tecnologies d'alt rendiments (*high-throughput techniques*) fa referència a l'automatització per augmentar el rendiment d'un procediment experimental, capacitant-lo per a dur a terme un gran nombre de processos en paral · lel.
Les ciències òmiques requereixen l'ús de tecnologies d'alt rendiment, ja que poden manejar mostres biològiques extremadament complexes en gran quantitats amb alta sensibilitat i especificitat.

2.2.3 Problemàtica de les dades òmiques

Les dades òmiques es caracteritzen per tenir un gran nombre de variables, això implica que les dades presentin un nombre molt elevat de dimensions. La Bioinformàtica està preparada per treballar en aquests conjunts de dades, tot i així, existeixen alguns problemes que fan que el seu anàlisi sigui més dificultós.

La problemàtica de treballar amb aquestes dades es pot resumir en cinc punts:

- Comparacions múltiples.
- Gran dimensionalitat de les dades.
- El paradigma de la n-petita i p-gran.
- Soroll en les dades biològiques "d'alt rendiment".
- Integració de fonts d'informació múltiples i heterogènies.

A continuació s'explica amb més deteniment cada un d'aquests punts:

- **Comparacions múltiples**: Aquest problema només sorgeix quan es fa inferència. Quan s'efectua més d'un contrast estadístic en l'anàlisis de dades, el criteri aplicat per la majoria d'investigadors és el d'ajustar o corretgir el nivell de significació depenent del número de contrastos efectuats, ja que augmenta el nombre de falsos positius.

- **Gran dimensionalitat de les dades:** Una de les principals característiques de les dades biològiques, és que són conjunts de dades amb un nombre molt elevat de variables, això fa que no es puguin representar gràficament i la seva interpretació sigui bastant complicada. Per aquest motiu es necessari tenir alguna idea en entendre i saber utilitzar tècniques de reducció de dimensió, ja que llavors les dades es poden representar gràficament en el pla i la seva interpretació és senzilla.

- **El paradigma de la n-petita i la p-gran**: Pel que fa a les persones que treballen amb conjunts de dades estan acostumades a tractar en grups de dades caracteritzats per un nombre molt gran d'observacions i un nombre més petit de variables, però en el cas de les dades biològiques succeeix tot el contrari, el nombre de variables normalment és més elevat que el nombre d'observacions. Per tot això, és important seleccionar eines d'anàlisi estadística que proporcionin alta especificitat i sensibilitat sota aquestes circumstancies.

- **Soroll en les dades biològiques d'alt rendiment**: Pel que fa a grans conjunts de dades biològiques és inevitable el soroll, ja que la informació biològica i els senyals d'interès són sovint observats per molts altres factors aleatoris i esbiaixats. Això fa que possiblement obstrueixin les senyals principals i la informació d'interès.

- **Integració de fonts d'informació múltiples i heterogènies**: L'últim problema que sorgeix en l'anàlisi de dades òmiques, és quan es vol combinar dos o més conjunts de dades que contenen els mateixos individus però diferents variables. La principal problemàtica que presenta la combinació d'aquests fitxers és quan aquestes variables són completament diferents, això vol dir que pertanyen a branques biològiques diferents.

Capítol 3

MACHINE LEARNING I MÈTODES KERNEL PER *STRINGS*

3.1 Machine Learning

3.1.1 Definició

La primera definició de Machine Learning va aparèixer al 1959 proposada per un pioner Nord-Americà en el camp del joc d'ordinador i la inteligència artificial, anomenat Arthur Lee Samuel. Aquesta proposta va ser: El camp d'estudi que dòna a les computadores la capacitat d'aprendre sense estar programat explícitament.

Però no va ser fins al 2006 que Mitchell un científic informàtic Americà va proposar una definició més formal: Es diu que un programa d'ordinador apren respecte a una tasca T, una mesura de rendiment R i un tipus d'experiència E, si el rendiment R de la tasca T sofreix una millora amb l'experiència E.

L'aprenentatge automàtic es pot considerar un subcamp de la ciència computacional i l'estadística, ja que fa servir aplicacions de les dos disciplines per treballar, pel que fa la ciència computacional l'emmagatzematge de grans volums de dades i pel que fa l'estadística el tractament de les dades per així poder obtenir resultats. L'aprenentatge automàtic s'empra en una gamma molt àmplia de tasques de computació, alguns exemples d'aplicacions podrien ser el filtrat de correu SPAM, reconeixement òptic de caràcters o la visió per computador.

3.1.2 Paradigmes de l'aprenentatge automàtic per a *strings*

Principalment cal destacar que l'aprenentage automàtic per string es preocupa bàsicament dels problemes de classificació. Es poden distingir tres tipus d'aprenentatge automàtic,

aquests són; Supervisat, No-supervisat i Semi-supervisat. A grans trets, la principal diferència que hi ha entre aquests tipus d'aprenentatge automàtic és si les observacions estan associades o no a una informació coneguda que s'anomena etiqueta.

- **Supervisat**: és un procés en el qual el model és parametritzat per un conjunt d'observacions en que cada una està associada a una etiqueta, per tant en aquest cas l'objectiu d'aquest mètode és predir el resultat d'una variable resposta basada en numeroses variables d'entrada, anomenades variables predictores o etiquetes. Pel cas de la predicció de la funció de la proteïna les observacions podrien ser el conjunt de característiques que presenten les propietats de la proteïna, mentre que la etiqueta podria ser una funció específica de la proteïna.
- **No-supervisat**: pel que fa al no-supervisat no es tenen variables predictores, per tant l'objectiu ja no serà el mateix que en el cas supervisat, en aquest cas l'objectiu a aconseguir és descriure les relacions i patrons entre un conjunt de variables d'entrada.
- **Semi-supervisat**: és la combinació entre l'aprenentatge automàtic supervisat i no-supervisat. Normalment el model es ajustat fent servir una quantitat petita de dades etiquetades, per exemple, proteïnes que han estat anotades per una determinada funció barrejades amb un nombre més gran de dades no etiquetades.

3.2 Mètodes kernel

La majoria dels mètodes d'anàlisi de dades primer defineixen la representació de cada objecte i després representen el conjunt d'objectes per a cada conjunt de les seves representacions. Els mètodes kernel són basats en una idea completament diferent, les dades no són representades individualment sinó a través d'un conjunt de comparacions dos a dos.

3.2.1 Idea general

El terme kernels fa referència als elements bàsics que comparteixen tots els mètodes kernels, per tant es pot dir que un kernel és una funció que s'aplica a un objecte la qual calcula la similitidud que hi ha entre dos objectes. Els metòdes kernel proporcionen un marc general de representació i han de satisfer algunes condicions matemàtiques.

Aquestes condicions matemàtiques són:

1. La representació sempre és una matriu quadrada, no depén de la naturalesa dels objectes que es volen analitzar.

2. La mida de la matriu que serveix per representar un conjunt de dades amb n objectes és sempre de dimensió $n \times n$, sigui quina sigui la naturalesa o la complexitat dels objectes.

3. Hi ha molts casos on la comparació d'objectes és més fàcil que trobar una representació per a cada objecte.

Avui en dia els mètodes kernel, en general, i el *support vector machine*, en particular, són utilitzats amb èxit en molts problemes reals.

3.2.2 Definició formal de funció kernel

Per tal què la funció k sigui una funció kernel cal que sigui simètrica i definida positiva. De manera més formal, una funció $k : \mathcal{X} \times \mathcal{X} \longrightarrow \mathbb{R}$, on $\mathcal{X}$ és l'espai d'entrada o *input space*, és una funció kernel si i només si és:

- **Simètrica:** és a dir, $k(\mathbf{x}, \mathbf{y}) = k(\mathbf{y}, \mathbf{x})$ per a dos objectes $\mathbf{x}, \mathbf{y} \in \mathcal{X}$ qualsevol.
- **Definida positiva:** és a dir, $\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j k(\mathbf{x}_i, \mathbf{x}_j) \geq 0$ per a qualsevol $n > 0$, qualsevol $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathcal{X}$ i qualsevol $c_1, \ldots, c_n \in \mathbb{R}$

3.2.3 Kernels com un producte escalar

Si es suposa que les dades que es volen analitzar són vectors de reals, $\mathbf{x} \in \mathbb{R}^p$ i $\mathbf{y} \in \mathbb{R}^p$, cada objecte es pot escriure com $\mathbf{x} = (x_1, \ldots, x_p)^T$ i $\mathbf{y} = (y_1, \ldots, y_p)^T$. Un està temptat de comparar aquests vectors utilitzant el seu producte escalar per qualsevol $\mathbf{x}, \mathbf{y} \in \mathbb{R}^p$

$$k(\mathbf{x}, \mathbf{y}) := \mathbf{x}^T \mathbf{y} = \sum_{i=1}^{p} x_i y_i$$

Aquesta funció és un kernel, ja que, és simètrica, ($\mathbf{x}^T \mathbf{y} = \mathbf{y} \mathbf{x}^T$) i definida positiva. Aquesta propietat es pot verificar mitjançant un càlcul molt simple, per a qualsevol $n > 0$ $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^p$ i $c_1, \ldots, c_n \in \mathbb{R}$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j k(\mathbf{x}_i, \mathbf{x}_j) = \sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j \mathbf{x}_i^T \mathbf{x}_j = \| \sum_{i=1}^{n} c_i \mathbf{x}_i \|^2 \geq 0$$

3.2.4 Kernels com una mesura de similitud

Tant en aquest treball, com en molts llibres que parlen de kernel i la comunitat de mètodes kernel, els kernels són presentats com una mesura de similitud, en aquest context, $k(\mathbf{x}, \mathbf{y})$ és "gran" quan $\mathbf{x}$ i $\mathbf{y}$ són similars.

Matriu kernel

Aquestes mesures es disposen en una matriu, aquesta matriu s'anomena matriu kernel $\mathbf{K}$, on l'element k_{ij} $(i = 1, \ldots, n, j = 1, \ldots, n)$ és el producte escalar entre l'objecte $\mathbf{x}_i$ i l'objecte $\mathbf{y}_j$.

$$\mathbf{K} = \begin{pmatrix} \mathrm{k}\,(\mathrm{x}_1, \mathbf{y}_1) & \mathrm{k}\,(\mathrm{x}_1, \mathbf{y}_2) & \ldots & \mathrm{k}\,(\mathrm{x}_1, \mathbf{y}_n) \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{k}\,(\mathrm{x}_n, \mathbf{y}_1) & \mathrm{k}\,(\mathrm{x}_n, \mathbf{y}_2) & \ldots & \mathrm{k}\,(\mathrm{x}_n, \mathbf{y}_n) \end{pmatrix}$$

Les principals característiques de la matriu kernel $\mathbf{K}$ són:

- És una matriu simètrica.
- En la matriu kernel calculada amb la funció gaussiana, els valors de la diagonal són uns i la resta de valors es troben a l'interval [0,1).
- És una matriu semidefinida positiva.
- És quadrada de dimensió n.
- Els components de la matriu kernel, k_{ij}, sempre pertanyen a $\mathbb{R}$, independentment de la naturalesa de les dades.

3.3 Kernel per *strings*

Es parla de kernel per *strings* quan l'objecte que es vol estudiar no és numèric, sinó que és de tipus caràcter. La primera pregunta que s'ha de fer un és: Com es pot aplicar el producte escalar en una cadena de caràcters?

La resposta a aquesta pregunta es resol fent un pas previ per obtenir un vector de naturals i així poder aplicar el producte escalar a aquest vector sense problemes. Aquest pas previ es crear un vector de comptatges de dimensió N^k, on N es l'allargada de l'abecedari i k la longitud de les possibles combinacions. Aquest vector és el nostre espai de característiques $\mathcal{M}$. Un cop s'ha fet aquest pas les seqüències d'entrada ja no són seqüències de caràcters sino vectors de comptatges, per tant ja es pot aplicar el producte escalar i el kernel queda representat d'aquesta manera:

$$k(\mathbf{x}, \mathbf{y}) = \textstyle\sum_{m \in \mathcal{M}} N(m, \mathbf{x}) \cdot N(m, \mathbf{y})$$

on $\mathbf{x}$ i $\mathbf{y}$ són les dos seqüències d'entrada, m és una de les combinacions pertanyent a l'espai de característiques $\mathcal{M}$, aquest conté totes les combinacions possibles i $N(m, \mathbf{x})$ és el nombre d'aparicions de la combinació m en la seqüència $\mathbf{x}$.

Per veure aquesta idea de forma més intuïtiva es fa un exemple molt senzill amb dues paraules: Abracadabra i cabra. La longitud de la combinació que es farà servir és 2, per tant, com s'ha esmentat abans l'espai de característiques $\mathcal{M}$ serà de dimensió $A^k = 26^2$.

x= ABRACADABRA

- Les combinacions de **x** de longitud 2 són: AB, BR, RA, AC, CA, AD, DA, AB, BR, RA.
- El vector de comptatges es resumeix a:

m	AB	BR	RA	AC	CA	AD	DA
$N(m, \mathbf{x})$	2	2	2	1	1	1	1

Per a la resta de combinacions que no apareixen en aquesta paraula — AA, BB, TT, etc. . . — el seu comptatge és 0.

y= CABRA

- les combinacions de **y** de longitud 2 són: CA, AB, BR, RA.
- El vector de comptatges es resumeix a:

m	CA	AB	BR	RA
$N(m, \mathbf{y})$	1	1	1	1

A partir dels vectors de comptatges es calcularàn els productes escalars, que en aquest cas s'anomenen kernel, corresponents per a construir la matriu kernel.

- Kernel per a ABRACADABRA:

$$k(\mathbf{x}, \mathbf{x}) = \sum_{m \in \mathcal{M}} N(m, \mathbf{x}) \cdot N(m, \mathbf{x}) = 2 \cdot 2 + 2 \cdot 2 + 2 \cdot 2 + 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 = 16$$

- Kernel per a CABRA:

$$k(\mathbf{y}, \mathbf{y}) = \sum_{m \in \mathcal{M}} N(m, \mathbf{y}) \cdot N(m, \mathbf{y}) = 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 = 4$$

- Kernel entre ABRACADABRA i CABRA:

Les combinacions comunes són: AB, BR, RA i CA.

$$k(\mathbf{x},\mathbf{y}) = \sum_{m\in\mathcal{M}} N(m,\mathbf{x})\cdot N(m,\mathbf{y}) = 2\cdot 1 + 2\cdot 1 + 2\cdot 1 + 1\cdot 1 = 7$$

Finalment la matriu kernel d'aquest exemple és:

$$\mathbf{K}(\mathbf{x},\mathbf{y}) = \begin{pmatrix} 16 & 7 \\ 7 & 4 \end{pmatrix}$$

En aquest exemple s'ha fet servir el tipus de kernel spectrum, ja que aquest és el més intuïtiu i el més fàcil d'entendre, però hi ha molts més tipus de kernel que es veuràn a continuació en el següent apartat.

3.3.1 Kernel per a *strings* amb R

En l'apartat anterior s'ha vist que calcular la matriu kernel per la comparació de dues paraules no és molt complicat, ara si l'objecte d'estudi és una proteïna o una cadena d'ADN, llavors el càlcul resulta ser més laboriós, per aquest motiu en aquest apartat s'expliquen unes eines les quals treballen amb algorismes basats ens kernels que t'ajuden a calcular les matriu kernels, ja sigui, entre dues paraules o milers de proteïnes. Aquestes són els paquets de R **kernlab** i **kebabs**.

- **Kernlab**: Pel que fa als kernels per *strings* estan implementats en la funció **stringdot**. Un dels arguments d'aquesta funció és el tipus de kernel a utilitzar (**type**). Actualment hi ha 4 kernels diferents implementats.

  - **Spectrum**: Aquest kernel només considera coincidència les subcadenes amb la mateixa llargada. A cada una d'aquestes subcadenes coincidents se li dóna un pes constants. El paràmetre d'aquest tipus de kernel és **length**, que correspon a la llargada de la subcadena i aquest ha de ser superior a 1.
    Primer de tot s'implementerà el problema que s'ha realitzat en l'apartat anterior per veure que el resultat coincideix. Es fa servir un kernel spectrum de llargada 2.

```
> library(kernlab)
> sk <- stringdot(type="spectrum", length=2, normalized=F)
> sk('cabra','abracadabra')

[1] 8

> #es el producte escalar entre les dos paraules
> kernelMatrix(sk,c('abracadabra','cabra'))
```

```
An object of class "kernelMatrix"
     [,1] [,2]
[1,]   17    8
[2,]    8    5
```

```
> #crea la matriu kernel entre les dos paraules
```

S'observa que no coincideix exactament amb el calculat a mà, això és perquè la funció stringdot també agafa com a possible seqüència lletra més espai, que és la seqüència final. Per aquest motiu la matriu realitzada amb R, és la mateixa que realitzada a mà però sumant-li 1 a cada valor.
A continuació es mostra un exemple més real, utilitzant com a seqüències d'entrada de 4 cadenes d'aminoàcids. Igual que en l'apartat anterior es fa servir un kernel spectrum de llargada 2. La matriu kernel s'ha obtingut mitjançant la funció **kernelMatrix**.

```
> (seq <- c("GACGAGGACCGA","AGTAGCGAGGT",
+           "ACGAGGTCTTT","GGACCGAGTCGAGG"))
```

```
[1] "GACGAGGACCGA"   "AGTAGCGAGGT"    "ACGAGGTCTTT"    "GGACCGAGTCGAGG"
```

```
> sk <- stringdot(type="spectrum", length=2, normalized=F)
> (km.sk <- kernelMatrix(sk, seq))
```

```
An object of class "kernelMatrix"
     [,1] [,2] [,3] [,4]
[1,]   28   10   10   23
[2,]   10   19    9   15
[3,]   10    9   13   12
[4,]   23   15   12   26
```

Es pot veure que la matriu et proporciona uns números, però amb aquests números és molt complicat saber si dos cadenes són molt o poc similars, per aquest motiu és millor utilitzar la matriu normalitzada. La matriu normalitzada és una matriu on els seus valors estan dins de l'interval [0,1] on 0 indica que no hi ha cap similitud i 1 indica màxima similitud. En aquest cas la matriu seria:

```
> skn <- stringdot(type="spectrum", length=2, normalized=T)
> (km.skn <- kernelMatrix(skn, seq))
```

```
An object of class "kernelMatrix"
          [,1]      [,2]      [,3]      [,4]
[1,] 1.0000000 0.4335550 0.5241424 0.8524367
[2,] 0.4335550 1.0000000 0.5726563 0.6748819
```

```
[3,] 0.5241424 0.5726563 1.0000000 0.6527140
[4,] 0.8524367 0.6748819 0.6527140 1.0000000
```

- **Boundrange**: Aquest kernel considera subcadenes coincidents de longitud inferior o igual a un nombre donat. El paràmetre d'aquest tipus de kernel és **length** que correspon a la llargada màxima possible de la subcadena i aquest ha de ser superior a 1.
  A continuació es mostra implementat el kernel boundrange de llargada igual a 2 amb l'exemple de les 4 cadenes s'aminoàcids.

```
> skb <- stringdot(type="boundrange", length=2, normalized=T)
> (km.skb <- kernelMatrix(skb, seq))

An object of class "kernelMatrix"
          [,1]      [,2]      [,3]      [,4]
[1,] 1.0000000 0.7470172 0.6564437 0.9234527
[2,] 0.7470172 1.0000000 0.7785902 0.8682115
[3,] 0.6564437 0.7785902 1.0000000 0.7482716
[4,] 0.9234527 0.8682115 0.7482716 1.0000000
```

- **Constant**: Aquest tipus de kernel considera totes les subcadenes coincidents i assigna un pes constant (per exemple, 1) a cada una d'elles . El kernel constant no requereix cap paràmetre addicional.
  A continuació es mostra implementat el kernel constant de llargada igual a 2 amb l'exemple de les 4 cadenes d'aminoàcids.

```
> skc <- stringdot(type="constant", length=2, normalized=T)
> (km.skc <- kernelMatrix(skc, seq))

An object of class "kernelMatrix"
          [,1]      [,2]      [,3]      [,4]
[1,] 1.0000000 0.4450230 0.4136918 0.6267035
[2,] 0.4450230 1.0000000 0.4729527 0.5069419
[3,] 0.4136918 0.4729527 1.0000000 0.4287465
[4,] 0.6267035 0.5069419 0.4287465 1.0000000
```

- **Exponential**: Aquest tipus de kernel presenta un paràmetre de decaïment, el qual aquest és fa més petit a mesura que la cadena coincident és fa més llarga. Aquest kernel requereix un factor de decaïment $\lambda > 1$.
  Seguidament es mostra implementat el kernel exponential de llargada igual a 2 i λ igual a 2 amb l'exemple de les 4 cadenes s'aminoàcids.

```
> ske <- stringdot(type="exponential", length=2,lambda = 2, normalized=T)
> (km.ske <- kernelMatrix(ske, seq))
```

```
An object of class "kernelMatrix"
          [,1]      [,2]      [,3]      [,4]
[1,] 1.0000000 0.7640946 0.6507107 0.9133114
[2,] 0.7640946 1.0000000 0.7714950 0.8561649
[3,] 0.6507107 0.7714950 1.0000000 0.7248606
[4,] 0.9133114 0.8561649 0.7248606 1.0000000
```

- **kebabs**: Aquest és un paquet que proporciona funcionalitat per analitzar seqüències biologíques —ADN, ARN i seqüències d'aminoàcids— a partir de mètodes kernel, en particular pel mètode de *support vector machine* (SVM).
  Primer de tot cal destacar que en aquest paquet es pot determinar si les seqüències amb que es treballen són de posició independent o dependent.

  - **Posició independent**: en aquest cas la localització de les combinacions és irrellevant per a determinar el valor de similitud. Aquest és el kernel més típic que és fa servir en seqüències.
  - **Posició dependent**: pel que fa a les seqüències de posició dependent la posició de la combinació si que és rellevant. Dins de les seqüències de posició dependent es pot distingir tres tipus: posició específica, distància ponderada i anotació específica

    * **Posició específica**: es considerarà coincidència sempre i quan les combinacions estiguin localitzades al mateix lloc de les dos seqüències. L'espai de característiques $\mathcal{M}$ per a la seqüència $\mathbf{x}$ de llargada L per aquest kernle és definit com:

      $$\mathcal{M}(\mathbf{x}) = ((\mathbf{1}(m, \mathbf{x}, 1), \ldots, \mathbf{1}(m, \mathbf{x}, L)))$$

      Aquest indicador, $\mathbf{1}(m, \mathbf{x}, l)$ és igual a 1 si la combinació m es situa en la posició l, on $l = 1, \ldots, L$, de la seqüència $\mathbf{x}$, en cas contrari és igual a 0. Llavors el kernel amb posició específica queda de la següent manera:

      $$k(\mathbf{x}, \mathbf{y}) = \sum_{m \in \mathcal{M}} \sum_{p=1}^{L} \mathbf{1}(m, \mathbf{x}, p) \cdot \mathbf{1}(m, \mathbf{y}, p)$$

    * **Distància ponderada**: pel que fa al kernel de posició específica només marca com a coincidència aquelles que estàn en la mateixa posició, en canvi aquest mètode no es tant estricte i marca com a coincidència les que es troben relativament a prop. Per poder decidir que és a prop es determina una ponderació segons a distància en que es troben les posicions.
      Llavors el kernel amb distància ponderada queda de la següent manera:

      $$k(\mathbf{x}, \mathbf{y}) = \sum_{m \in \mathcal{M}} \sum_{p=1}^{L_\mathbf{x}} \sum_{q=1}^{L_\mathbf{y}} \mathbf{1}(m, \mathbf{x}, p) \cdot E(p, q) \cdot \mathbf{1}(m, \mathbf{y}, q)$$

      On $L_\mathbf{x}$ i $L_\mathbf{y}$ són les longituds de les seqüències $\mathbf{x}$ i $\mathbf{y}$, $E(p, q)$ és la funció de la distància ponderada per les posicions p i q, m és una combinació de l'espai de característiques $\mathcal{M}$.

* **Anotació específica**: en aquest cas s'evalua l'informació d'anotació juntament amb les combinacions de la seqüència. Per exemple les consideracions de similitud de seqüències de gens podrien dividir-se d'acord amb les parts intròniques i exòniques de les seqüències.

El paquet proporciona 3 tipus de kernels per a l'anàlisi de seqüències, tant si es tracta de seqüències amb posició independent com amb seqüències de posició dependent, i un kernel el qual només es pot utilitzar per seqüències de posició independent.

- **Spectrum Kernel**: per l'spectrum kernel amb el paràmetre k, les característiques són totes les subcadenes de longitud k. Si $|N|$ és el nombre de caràcters en el conjunts de caràcters, la dimensió de l'espai característic és $|\mathcal{M}| = |N|^k$
- **Mismatch Kernel**: el mismatch kernel és similar a l'spectrum kernel ja que el paràmetre k té el mateix significat que en l'spectrum kernel, per tant també extreu subcadenes amb longitud k. El paràmetre m defineix el nombre màxim de desajustos que estàn permesos per a cada combinació. Aques paràmetre s'entendra millor si s'exposa un petit exemple. Per exemple si es té un mismatch kernel de $k = 2$ i $m = 1$, la subcadena TA, extreta d'una cadena d'aminoàcids, serà coincidència amb qualsevol d'aquestes subcadenes : AA, CA, GA, TA, TC, TG, TT, ja que qualsevol d'aquestes subcadenes conté o una T en primera posició o una A a segona posició més un desajust.
  L'espai característic i la seva dimensió $|\mathcal{M}| = |N|^k$ és idèntica a l'spectrum kernel, el paràmetre m no té un impacte en la mida de l'espai característic.
- **Gappy Pair Kernel**: aquest kernel presenta dos paràmetres, k i m, i la principal diferència que presenta en vers als altres kernels és que inclou com a combinació, dos lletres separades per un espai comodí. El paràmetre m és el que diu quin és el nombre màxim que hi pot haver de posicions comodí, com es pot veure aquest paràmetre no presenta el mateix significat que s'ha vist en el mismatch kernel. Pel que fa al paràmetre k aquest ens diu la longitud fixa de la primera part de la combinació així com de la segona part.
  Seguidament es pot veure com sería l'estructura d'un Gappy Pair Kernel:

$$\underbrace{part1}_{k}\underbrace{.....}_{0\leq i\leq m}\underbrace{part2}_{k}$$

  Aquest kernel s'entendrà més fàcilment si s'exposa amb un exemple, és té una seqüència d'ADN, CAGAT, i es realitza un gappy pair kernel amb $k = 1$ i $m = 2$ les combinacions que surten són les següents: CA, C.G, C..A, AG, A.A, A..T, GA, G.T, i AT, els punts indiquen les posicions comodí que hi ha entremig.
  L'espai característic del gappy pair kernel és $|\mathcal{M}| = (m + 1)|N|^{2k}$. Com es

pot veure és considerablement més gran que l'espai característic de l'Spectrum Kernel o Mismatch Kernel.

- **Motif Kernel**: el motif kernel es creat a partir d'una llista de motifs predefinida. Amb aquest kernel la seqüència d'anàlisi pot ser restringida amb un conjunt de característiques per l'usuari. Aquest Kernel també dóna major flexibilitat pel que fa a la definició de les característiques individuals.

  Es tenen dues possibilitats o posar només un sol motif o posar-ne més d'un que s'anomena col · lecció de motifs. A continuació es veuen els possibles motifs:

  1. Un sol caràcter del conjunt de caràcters, per exemple C en el cas de les seqüències d'ADN, aquest només coincideix amb aquest caràcter.
  2. El caràcter comodí (.) que coincideix amb qualsevol caràcter del conjunt de caràcters.
  3. La definició d'un grup de substitució, que és una llista dels caràcters individuals dels conjunt de caràcters, per exemple [CT] els quals només coincideixen amb els caràcters C i T. Altrament si s'afegeix el circumflex davant d'aquesta llista llavors el que s'esta dient és, que volem tots els caràcters menys els que hi ha dins dels claudàtors, per exemple si es té [^CT] els caràcters que coincideixen són A i G —en el cas que s'estigues treballant amb una cadena d'ADN—.

  Si el que es vol es fer una col · lecció de motifs, llavors s'haurà de crear un vector de motifs, el qual amb R és pot implementar molt fàcilment amb la funció concatenar.
  La mida de l'espai característic del motif kernel correspon al nombre de motifs que hi ha en la col · lecció.

Tal i com s'ha fet pel paquet **kernlab**, s'implementa un exemple per tenir petita idea de com funcionen aquestes funcions de kernel. Com s'ha dit a la definició, aquest paquet només treballa amb seqüències biològiques, per tant no es pot fer l'exemple entre dues paraules.
Per aquest motiu es fa directament l'exemple de les 4 cadenes d'aminoàcids.

```
> ### baixar kebabs#####
> source("http://bioconductor.org/biocLite.R")
> biocLite("kebabs")

package 'kebabs' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\Aurea\AppData\Local\Temp\Rtmpc1KVr7\downloaded_packages
```

```
> library(kebabs)

> (seq1 <- AAStringSet(c("GACGAGGACCGA",
+                        "AGTAGCGAGGT",
+                        "ACGAGGTCTTT",
+                        "GGACCGAGTCGAGG")))

  A AAStringSet instance of length 4
    width seq
[1]    12 GACGAGGACCGA
[2]    11 AGTAGCGAGGT
[3]    11 ACGAGGTCTTT
[4]    14 GGACCGAGTCGAGG
```

Primer de tot es fa servir el kernel de tipus spectrum amb $k = 2$, com que es tracten de cadenes d'ADN, el seu espai característic estarà compost per A,G,C i T i la seva dimensió és $4^2 = 16$.

```
> specK2 <- spectrumKernel(k=2,normalized = T)
> (km.spec <- getKernelMatrix(specK2, seq1))

An object of class "KernelMatrix"
          [,1]      [,2]      [,3]      [,4]
[1,] 1.0000000 0.4536092 0.5555556 0.8852704
[2,] 0.4536092 1.0000000 0.5443311 0.7071068
[3,] 0.5555556 0.5443311 1.0000000 0.6928203
[4,] 0.8852704 0.7071068 0.6928203 1.0000000
```

A continuació es realitza el kernel mismatch amb $k = 2$ i $m = 1$, l'espai característic és el mateix que en el kernel spectrum i per tant la seva dimensió també és $4^2 = 16$.

```
> mismK2M1 <- mismatchKernel(k=2, m=1,normalized = T)
> (km.mis <- getKernelMatrix(mismK2M1, seq1))

An object of class "KernelMatrix"
          [,1]      [,2]      [,3]      [,4]
[1,] 1.0000000 0.8957561 0.7526056 0.9705975
[2,] 0.8957561 1.0000000 0.8645009 0.9601510
[3,] 0.7526056 0.8645009 1.0000000 0.8410636
[4,] 0.9705975 0.9601510 0.8410636 1.0000000
```

El tercer kernel que s'utilitza és el gappy pair kernel amb $k = 1$ i $m = 3$, en aquest cas l'espai característic ja no coincideix amb els kernels vistos anteriorment així com la seva dimensió ja que ara és de $(3 + 1)|4|^{2*1} = 64$.

```
> gappyK1M3 <- gappyPairKernel(k=1, m=3,normalized = T)
> (km.gap <- getKernelMatrix(gappyK1M3 , seq1))

An object of class "KernelMatrix"
          [,1]      [,2]      [,3]      [,4]
[1,] 1.0000000 0.5542789 0.5185189 0.8317287
[2,] 0.5542789 1.0000000 0.5217391 0.7027309
[3,] 0.5185189 0.5217391 1.0000000 0.5141934
[4,] 0.8317287 0.7027309 0.5141934 1.0000000
```

Finalment es fa servir l'últim tipus de kernel que s'ha esmentat anteriorment —motif kernel—. En aquest cas es l'usuari qui sel · lecciona l'espai característic. Pel que fa a l'exemple l'espai característic sel · leccionat és: "A.G","[CT]", com que hi ha 5 possibles combinacions dins de l'espai característic sel · leccionat la seva dimensió també és 5.

```
> motifCollection1 <- c("A.G","[CT]")
> motif1 <- motifKernel(motifCollection1,normalized = T)
> (km.mot <- getKernelMatrix(motif1 , seq1))

An object of class "KernelMatrix"
          [,1]      [,2]      [,3]      [,4]
[1,] 1.0000000 0.9647638 0.9647638 0.9417419
[2,] 0.9647638 1.0000000 1.0000000 0.9970545
[3,] 0.9647638 1.0000000 1.0000000 0.9970545
[4,] 0.9417419 0.9970545 0.9970545 1.0000000
```

Tots els exemples que s'han implementat anteriorment són kernels de posició independent, si el que es vol és realitzar un kernel amb posició dependent els paràmetres canviaràn de la següent manera:
Els exemples només es faràn amb el kernel de tipus spectrum, però totes les instruccions són equivalents pels kernels gappy pair i motif.

- **posició específica**: Per poder fer un kernel amb posició específica només caldrà afegir el paràmetre **distWeight** igual a 1 a la nostra funció kernel, ja sigui de tipus spectrum, gappy pair o motif.

  ```
  > specK2.esp <- spectrumKernel(k=2,normalized = T,distWeight = 1)
  > (km.spec.esp <- getKernelMatrix(specK2.esp, seq1))
  ```

```
An object of class "KernelMatrix"
           [,1]       [,2] [,3]      [,4]
[1,] 1.00000000 0.09534626  0.0 0.1672484
[2,] 0.09534626 1.00000000  0.1 0.0000000
[3,] 0.00000000 0.10000000  1.0 0.0000000
[4,] 0.16724840 0.00000000  0.0 1.0000000
```

A més a més, també es pot específicar l'alineament de les cadenes, dit d'una altra manera, es pot especificar la posició d'inici de les cadenes. Amb **distWeight** igual a 1 les cadenes comencen per la posició 1, ara si es vol que comencin per una altra posició llavors s'haurà d'aplicar la funció **positionMetadata** a la nostra cadena i això igualar-ho a un vector de naturals que indiqui la posició de principi de cada cadena.

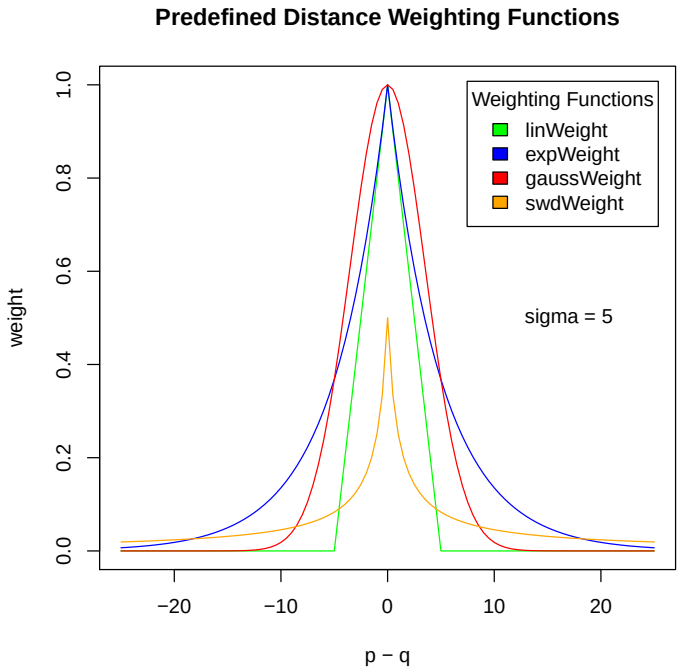
```
> specK2.esp2 <- spectrumKernel(k=2,normalized = T,distWeight = 1)
> positionMetadata(seq1) <- c(3, 4, 2, 10) #vector d'iniciadors
> (km.spec.esp2 <- getKernelMatrix(specK2.esp2, seq1))
```

```
An object of class "KernelMatrix"
          [,1]      [,2]      [,3]      [,4]
[1,] 1.0000000 0.0000000 0.4767313 0.3344968
[2,] 0.0000000 1.0000000 0.0000000 0.1754116
[3,] 0.4767313 0.0000000 1.0000000 0.3508232
[4,] 0.3344968 0.1754116 0.3508232 1.0000000
```

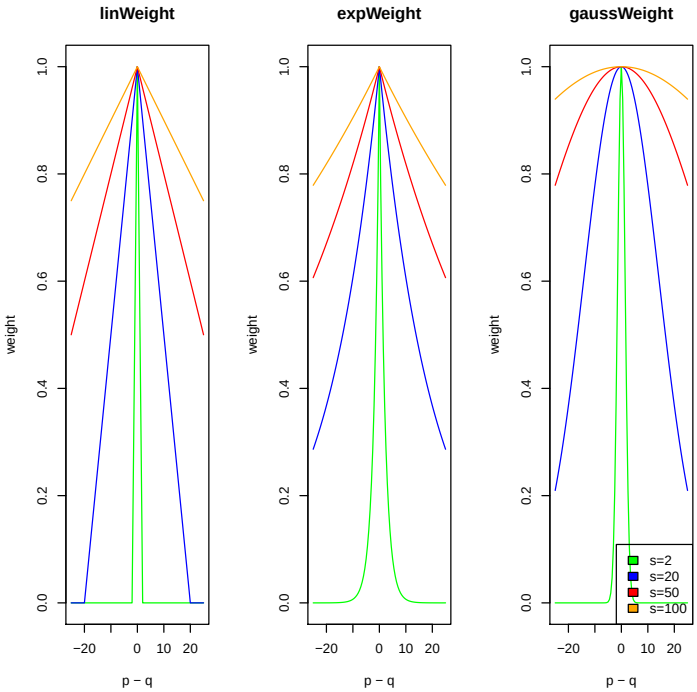
– **Distància ponderada**: El paquet proporciona quatre funcions predeterminades de distància ponderada on la funció de distància és definida com $d = |p-q|$

  - **Distància ponderada lineal**: $E_{lin,\sigma}(d) = max(1 - \frac{d}{\sigma}, 0)$
  - **Distància ponderada exponencial**: $E_{exp,\sigma}(d) = exp(-\frac{d}{\sigma})$
  - **Distància ponderada gaussiana**: $E_{exp,\sigma}(d) = exp(\frac{-d^2}{\sigma^2})$
  - ***The shifted degree*** **kernel (SWD)**: $E_{SWD}(d) = \frac{1}{2\cdot(|d|+1)}$

  A continuació es mostra un gràfic en el qual es pot veure la ponderació envers la distància que hi ha entre les dues posicions, i això per les quatre funcions que s'ha vist anteriorment.



En els tres primers casos el paràmetre σ controla el comportament de proximitat. Per valors petits de σ, pròxims a 0, el kernel de distància ponderada s'assembla més al kernel de posició específica, en canvi per valors grans de σ llavors s'aproxima al kernel de posició independent.
Seguidament es pot veure a partir dels gràfics següents com la ponderació de la distància augmenta quan la sigma és fa més gran.



Per tant si es vol fer un kernel amb distància ponderada lineal, llavors el pa-

ràmetre **distWeight** haurà de ser igual a linWeight i la σ que ens interessi, expWeight per la distància ponderada exponencial, gaussWeight per la distància ponderada gaussiana i swdWeight per *the shifted degree weighted* kernel (SWD).

A continuació es mostren els resultats obtinguts per cada tipus de distància ponderada i σ igual a 5 fent servir el kernel de tipus spectrum.

```
> specK2.esp.dist <- spectrumKernel(k=2,normalized = T,
+                                   distWeight = linWeight(sigma = 5))
> (km.spec.esp.dist <- getKernelMatrix(specK2.esp.dist, seq1))

An object of class "KernelMatrix"
          [,1]      [,2]      [,3]      [,4]
[1,] 1.0000000 0.3149219 0.4723102 0.4091379
[2,] 0.3149219 1.0000000 0.3158381 0.4407906
[3,] 0.4723102 0.3158381 1.0000000 0.3208327
[4,] 0.4091379 0.4407906 0.3208327 1.0000000

> specK2.esp.dist2 <- spectrumKernel(k=2,normalized = T,
+                                    distWeight = expWeight(sigma = 5))
> (km.spec.esp.dist2 <- getKernelMatrix(specK2.esp.dist2, seq1))

An object of class "KernelMatrix"
          [,1]      [,2]      [,3]      [,4]
[1,] 1.0000000 0.3861585 0.4965267 0.5377418
[2,] 0.3861585 1.0000000 0.3829607 0.5042891
[3,] 0.4965267 0.3829607 1.0000000 0.4503818
[4,] 0.5377418 0.5042891 0.4503818 1.0000000

> specK2.esp.dist3 <- spectrumKernel(k=2,normalized = T,
+                                    distWeight = gaussWeight(sigma = 5))
> (km.spec.esp.dist3 <- getKernelMatrix(specK2.esp.dist3, seq1))

An object of class "KernelMatrix"
          [,1]      [,2]      [,3]      [,4]
[1,] 1.0000000 0.4604992 0.4945751 0.5034965
[2,] 0.4604992 1.0000000 0.4613056 0.5398002
[3,] 0.4945751 0.4613056 1.0000000 0.4195430
[4,] 0.5034965 0.5398002 0.4195430 1.0000000

> specK2.esp.dist4 <- spectrumKernel(k=2,normalized = T,
+                                    distWeight = swdWeight())
> (km.spec.esp.dist4 <- getKernelMatrix(specK2.esp.dist4, seq1))
```

```
An object of class "KernelMatrix"
          [,1]      [,2]      [,3]      [,4]
[1,] 1.0000000 0.2273825 0.4808522 0.4650585
[2,] 0.2273825 1.0000000 0.2064921 0.3643473
[3,] 0.4808522 0.2064921 1.0000000 0.4007641
[4,] 0.4650585 0.3643473 0.4007641 1.0000000
```

Capítol 4

TÈCNIQUES DE CLASSIFICACIÓ: SUPPORT VECTOR MACHINE

4.1 Classificació

Es pot definir classificació com: "Distribució d'un conjunt d'objectes en un cert nombre de classes, coordinades o subordinades, segons un criteri determinat." L'estadística i la intel · ligència artificial han desenvolupat eines per a la construcció de models per classificar objectes. En intel · ligència artificial aquests mètodes formen part del que es coneix com a aprenentatge automàtic.

4.2 Classificació en l'aprenentatge automàtic

Pel que fa a la classificació en l'aprenentatge automàtic es pot distingir dos tipus de classificació, la classificació supervisada i la classificació no-supervisada. A continuació s'explica més detalladament els dos tipus de classificació.

Classificació supervisada[2][3]: La classificació supervisada, també anomenada com a predicció de classes, és una aplicació de la disciplina aprenentatge automàtic.
El punt de partida de la classificació supervisada és una base de dades d'entrenament formada per un conjunt d'n mostres independents, $D_n = \{(\mathbf{x}_1, c_1), \ldots, (\mathbf{x}_n, c_n)\}$ distribuïdes en una distribució de probabilitat desconeguda $p(\mathbf{x}, \mathbf{c})$. Cada mostra $(\mathbf{x}_i, c_i)$ és caracteritzada per un grup de d variables predictives $x_1, \ldots, x_d$ i una etiqueta o variable d'interés c, la qual està associada a les variables predicitives.
Una vegada s'ha fet el preprocessament a les dades disponibles, l'algorisme de classificació supervisat fa servir la base de dades d'entrenament per a induïr un classificador amb l'objectiu de predir el valor de l'etiqueta futura, com per exemple amb mostres on l'etiqueta és desconeguda.
La classificació supervisada s'utilitza per resoldre problemes molt diferents de la bioin-

formàtica, com la predicció de l'estructura secondària d'una proteïna, l'expresió genètica basada amb la diagnòsi. Les tècniques actuals de classificació supervisada han demostrat poder obtenir resultats força satisfàctoris.
Encara que, l'aplicació de l'algorisme per induïr un classificador és el principal pas per la disciplina de classificació supervisada, però també hi ha dos altres aspectes que són vitals en la classificació supervisada:

1. La necessitat d'estimar raonablement l'exactitud predictiva del model construït.
2. la necessitat d'un procés de reducció de dimensionalitat, amb la finalitat de millorar la precisió de la predicció o per utilitzar un nombre manejable d'atributs.

Molts estudis asseguren que no hi ha un millor mètode de classificació per resoldre tots els problemes de classificació. A continuació s'expliquen breument alguns d'aquests mètodes.

- **Xarxes neuronals**: Consisteix en unes unitats de processament elementals anomenades neurones, aquestes són combinades en capes després d'estar sotmeses a una transformació no lineal. En general, només les neurones pertanyents a capes consecutives són conectades. L'arquitectura basica consisteix en tres tipus de capes de neurones: entrada, hidden i sortida.
- ***K-nearest neighbours***: Es classifica una observació assignant-li la etiqueta amb major freqüència dels seus k veïns més propers. La estrategia més comuna usada en classificació es l'esquema de "majoria de vot".
- **Arbres de decisió**: Es classifica una observació fent servir una seqüència de preguntes les quals la següent pregunta depen de la resposta actual. El model és induït per un procediment recursiu *top-down* i pot ser fàcilment entés i comprobat per experts.
- ***Suport vector machine***: Projecta mostres d'entrada en un espai de grans dimensions i construeix un hiperpla de màxima separació. SVM assumeix que com més gran sigui la separació entre classes més bona serà la precisió predictiva. El marge més llarg es determinat per un conjunt d'observacions, les quals són els punts d'entrenament més díficils de classificar, s'anomenen *suport vectors*.

Classificació no supervisada[3]: No és molt usual fer servir el tèrme classificació no supervisada per aquests tipus de problemes de classificació, en aquest cas és més comú utilitzar el terme *clustering*. El *clustering* consisteix en la partició del conjunt de dades en subconjunts (*clusters*) segons les similituds entre observacions. A diferència de la classificació supervisada, el *clustering* no serveix per predicció de classes, sinó per a descobrir-les. Pel cas de la predicció de la funció d'una proteïna, els mètodes de *clustering* es fan servir per agrupar proteïnes relacionades i poder descobrir la seva família.

S'expliquen alguns algorismes de classificació d'aprenentatge automàtic no-supervisat:

- ***Hiererchical clustering***: Construeix una jerarquia de clusters per a cada nivell del cluster. Els clusters dins del mateix cluster són més similars entre si que els de diferent cluster.
- Els models de xarxes neuronals com *Self-Organizing Map* i *Adaptive Resonance Theory* també són usats com algoritme d'aprenentatge automàtic no supervisat.

4.3 *Support vector machine*

Avui en dia *support vector machine*[2] és una de les tècniques de classificació més popular. És un mètode robust i la bona precisió que demostra en moltes tasques reals l'han col · locat entre les tècniques preferides dels pràcticants. *Support vector machine* representa les mostres d'entrada en un espai de dimensions superiors on es construeix un hiperplà de separació màxima entre les instàncies de les diferents classes.
El mètode funciona mitjançant la construcció de dos hiperplans paral · lels a cada costat d'aquest hiperplà. El mètode de *support vector machine* intenta trobar la separació de l'hiperplà que maximitza l'àrea de separació entre els dos hiperplans paral · lels. Aquest mètode assumeix que com més gran sigui la separació entre aquests dos hiperplans paral · lels millor és la precisió predictiva de classificació.
En moltes situacions no és possible separar perfectament tots els punts d'entrenament en diferents classes. Tot i que els classificadors de *support vector machine* són molt populars a causa dels notables nivells de precisió aconseguits en molts problemes de la bioinformàtica, també són critícats per la manca d'interpretabilitat del criteri de classificació.

Com exemple d'aplicació, es té una base de dades d'entrenament amb 14 observacions i dos classes per tant $D_{14} = \{(\mathbf{x}_1, c_1), \ldots, (\mathbf{x}_{14}, c_{14})\}$, on $\mathbf{x}_1, \ldots, \mathbf{x}_{14} \in \mathcal{X}$ i $c \in \{-1, +1\}$. Amb aquesta base de dades s'aplica l'algorisme de classificació supervisada *support vector machine* i a continuació s'observen els resultats gràficament.

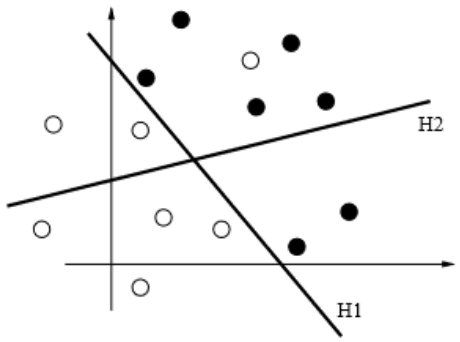


Figura 4.1: *support vector machine*[14]

A partir del gràfic anterior es pot veure que l'hiperplà *H1* diferència els cercles blancs dels cercles negres amb només un error. L'hiperplà *H2* també separa els punts en dos grups, però en aquest cas amb cinc errors. El principi de minimització de riscos empírics indica que s'ha d'optar per un hiperplà amb un nombre mínim d'errors en el conjunt d'entrenament, que és *H1* en aquest cas.

4.4 *Support vector machine* amb R

4.4.1 Classificació binària

En aquest apartat el que es farà és implementar la funció de *support vector machine* en R, tal i com s'ha fet en l'apartat de mètodes kernel amb R per implementar aquesta funció també s'utilitzaràn els paquets d'R **kernlab** i **kebabs**.

- **Kernlab**: Pel que fa al paquet **kernlab** el mètode *support vector machine* està implementat en la funció **ksvm**. Aquesta funció pot realitzar tasques de classificació, detecció de novetat i regressió, però en aquest treball només s'aplicarà per fer classificació de *strings*.
  Els paràmetres imprescindibles d'aquesta funció són:

  - **x**: és el paràmetre que equival a les dades d'entrenament, aquest ha de ser una llista de vectors de caràcters.
  - **y**: és el paràmetre que equival al vector de respostes, aquest ha de contenir una etiqueta per a cada component contingut en **x**.
  - **type**: aquest argument és per distingir la tasca que ha de fer el *support vector machine*, com que en aquest cas s'utilitza per classificació assumint que només hi ha una classe per observació i que sempre es tindrà el vector d'etiquetes, aquest paràmetre serà igual a $\mathbf{C-svc}$.
  - **kernel**: en aquest paràmetre s'ha de dir amb quin tipus de kernel vols que es realitzi la classificació. Pel que fa al paquet **kernlab** aquest pot ser: spectrum, boundrange, constant o exponential amb els seus paràmetres corresponents.
  - **C**: és el paràmetre que equival al cost de la classificació, aquest pot ser un escalar o un vector.

  Per poder entendre millor aquesta funció d'R s'exposa un exemple que vol classificar textos en dos tipos, acq i crude, en funció d'unes dades conegudes. La base de dades que s'utilitza és reuters i està composta per 20 textos de tipus acq i 20 textos de tipus crude. La base de dades d'entrenament està composta pel 70% dels textos

que componen la base de dades original amb les seves etiquetes corresponent. El mètode de classificació es realitza per un kernel de tipus spectrum amb $k = 2$.

```
> data(reuters)
> numSamples <- length(reuters)
> trainingFraction <- 0.7
> train <- sample(1:numSamples, trainingFraction * numSamples)
> test <- c(1:numSamples)[-train]
> sk <- stringdot(type="spectrum", length=2, normalized=TRUE)
> svp <- ksvm(x=reuters[train],y=rlabels[train],type="C-svc",C=1,kernel=sk)
> svp

Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 1

String kernel function.  Type =  spectrum
 Hyperparameters : sub-sequence/string length =  2
 Normalized

Number of Support Vectors : 28

Objective Function Value : -20.113
Training error : 0
```

Com es pot veure en la taula de resultats la funció **ksvm** proporciona el nombre de *supports vectors*, el valor de la funció objectiu i l'error d'entrenament, per tant la funció **ksvm** només calcula els *supports vectors* per veure la predicció del mètode s'ha de fer servir la funció **predict**. Els paràmetres imprescindibles que s'han d'especificar en aquesta funció són:

- **object**: aquest paràmetre equival a un objecte de tipus **ksvm**.
- **newdata**: aquest paràmetre equival a la base de dades test.

A continuació es pot veure la implementació de la funció **predict** utilitzant la funció **ksvm** que s'ha fet servir en l'apartat anterior i com a base de dades test el 30% restant que no s'ha fet servir en la base de dades d'entrenament. Aquesta funció ens proporciona com a resultats la matriu de confussió on es pot veure perfectament quins textos s'han classificat correctament i quins no.

```
> ypred <- predict(object=svp,newdata=reuters[test])
> print(table(ypred,rlabels[test]))

ypred   acq crude
  acq     5     0
  crude   1     6
```

Sembla ser que en la mostra de test hi ha molts més textos crude que acq. Aquests resultats estàn desbanlancejats.

- **Kebabs**: pel que fa al paquet **kebabs** la funció de *support vector machine* està implementada en la funció **kbsvm**, amb aquesta funció es poden realitzar tasques de classificació i regressió, però tal i com s'ha explicat en l'apartat del **kernlab** només s'utilitzarà aquesta funció per classificació de strings.
  Els paràmetres imprescindibles d'aquesta funció són:

  - **x**: aquest paràmetre equival a la base de dades d'entrenament. Aquesta base de dades ha de ser composta per seqüències biologiques, ja sigui per seqüències d'ADN, ARN o AA.
  - **y**: és el paràmetre que equival al vector de respostes, aquest ha de contenir una etiqueta per a cada component contingut en **x**.
  - **kernel**: aquest paràmetre és equivalent al tipus de kernel amb que és vol realitzar la classificació, ja pot ser un kernel implementat en el paquet **kebabs** o en el paquet **kernlab**.
  - **pkg**: aquest paràmetre equival al nom del paquet que conté la implementció del *support vector machine* que es usat per l'entrenament, aquests paquets poden ser: kernlab, e1071, LiblineaR. L'el · lecció d'aquest paràmetre depen del tipus de classificció.
  - **svm**: aquest paràmetre equival al nom del tipus de classificació que es vol usar. Tal i com s'ha esmentat en l'apartat de **kernlab** en aquest treball només es farà servir el tipus $\mathbf{C-svc}$.
  - **C**: aquest paràmetre equival al cost de classificació.

  Tal i com s'ha fet pel paquet **kernlab** s'implementa un exemple per entendre millor la funció **kbsvm**. Per aquest exemple es fa servir la base de dades TFBS[23], aquesta base de dades conté 259 seqüències d'ADN de potenciadors específics de teixits d'embrions de ratolins d'onze dies i mig i 241 seqüències negatives en la mostra del genoma mm9 amb les seves corresponents etiquetes.

```
> data(TFBS)
> enhancerFB

  A DNAStringSet instance of length 500
      width seq                                               names
  [1]   827 ACTAAACAACATCAATCAATAC...TTCTCTAGGCAAAATCCTGACA chr19.21240050.21...
  [2]   678 GAATATAGACCCTTGGTGGTGG...GGAGGAAGTTATATTAATTTAT chr2.144463827.14...
  [3]   878 CACCCACATGGTGGCTCTCAAC...TCTGCTCCCCACTGTATCACTC chr7.38525408.385...
  [4]   927 TGCCGTAGTGTGCCAGCTTTTC...TTCTCTGATTCTCCTGGATCAA chr4.90747075.907...
  [5]   953 CTTCATATACCTATTAATTCAG...AATAACCTAATTTAAAAAGGGG chr4.9148679.9149631
  ...   ... ...
[496]   478 ACGAGAATGAGGGAAAGCACTG...TTGCAGTCTGCCTTCCGAGTGA chr8.45499713.455...
[497]  1552 AGGGGACTGGAAGAAAGGAACC...AGATCTCTCTTCTAAGCGAGCA chr8.94657200.946...
[498]   503 GATGAACGTAAAATGCAAGCTG...GAACCTAATGACTCCCTTCTGA chr17.73900306.73...
[499]  1577 ACCCCTCTGAGACCAAGAGGTG...GGCAAGTAGGTCTGTATTCCTG chr9.90587075.905...
[500]   778 CACTGTCATAACTTGCTTTCTA...GGCTTGTAGAGTAGCTGCTCTC chr17.48680137.48...

> train <- sample(1:length(enhancerFB), length(enhancerFB) * 0.7)
> test <- c(1:length(enhancerFB))[-train]
> specK2 <- spectrumKernel(k=2,normalized=T)
> model <- kbsvm(x=enhancerFB[train], y=yFB[train], kernel=specK2,
+                pkg="kernlab", svm="C-svc", C=10, explicit="no")
> model

KEBABS result object of class "KBModel"

Number of samples     : 350
Kernel                : Spectrum Kernel: k=2

Available Packages    : kernlab, e1071, LiblineaR
Package               : kernlab
SVM                   : C-svc
Classifier Parameters : C = 10

Call:
kbsvm(x = enhancerFB[train], y = yFB[train], kernel = specK2,
     pkg = "kernlab", svm = "C-svc", C = 10, explicit = "no")

Classifier specific model    :
```

```
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 10

[1] " Kernel matrix used as input."

Number of Support Vectors : 234

Objective Function Value : -2159.254
Training error : 0.228571
```

Com es pot veure en la taula de resultats la funció **kbsvm** proporciona el nombre de *supports vectors*, el valor de la funció objectiu i l'error d'entrenament, per tant la funció **kbsvm** només calcula els *supports vectors* per veure la predicció del mètode s'ha de fer servir la funció **predict** i **evaluatePrediction**. Els paràmetres imprescindibles que s'han d'especificar en la funció **predict** són:

- **object**: aquest paràmetre equival a un objecte de classe KBModel creat per la funció **kbsvm**.
- **x**: aquest paràmetre equival a la base de dades test.

La funció **predict** només et calcula les prediccions, llavors es fa servir la funció **evaluatePrediction** la qual et proporciona la matriu de confussió per veure més fàcilment on s'ha classificat correctament i on no. Els paràmetres imprescindibles que s'han d'especificar en la funció **evaluatePrediction** són:

- **prediction**: aquest paràmetre equival al resultats obtinguts en la funció **predict**.
- **label**: aquest paràmetre equival al vector d'etiquetes de la base de dades test.
- **allLabels**: vector que conté totes les etiquetes que aparèixen una vegada. Aquest paràmetre és necessari només si el vector d'etiquetes és numèric.

```
> pred <- predict(model, enhancerFB[test])
> evaluatePrediction(pred, yFB[test], allLabels=unique(yFB))
```

```
    1 -1
1  57 22
-1 27 44

Accuracy:              67.333% (101 of 150)
Balanced accuracy:     67.262% (57 of 84 and 44 of 66)
Matthews CC:            0.343

Sensitivity:           67.857% (57 of 84)
Specificity:           66.667% (44 of 66)
Precision:             72.152% (57 of 79)
```

A partir de les funcions anteriors s'obté la matriu de confussió, el percentatge de *accuracy*, el percentatge de *balanced accuracy*, el coeficient de correlació de Matthews, el percentatge de sensibilitat, el percentatge d'especificitat, i el percentatge de precisió.

- ***Accuracy***: grau de concordança entre el valor mesurat i el valor veritable d'un mesurand.
- ***Balanced accuracy***: la *accuracy* mitjana obtinguda en cada classe.
- **Coeficient de correlació de Matthews**: es usat en l'aprenetatge automàtic com una mesura de qualitat de les classificacions binàries (dos classes), introduït pel bioquímic Brian W. Matthews[12]. Aquest valor ha de pertanyer a l'inteval [1,-1], si aquest coeficient és igual a 1 significa que la predicció és perfecta en canvi si aquest coeficient és igual a -1 llavors significa que la predicció és totalment contrària a l'observació.

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

- **Sensibilitat**: indica la capacitat del nostre estimador per donar com a casos positius els casos realment malalts, o proporció de malalts correctament identificats.
- **Especificitat**: indica la capacitat del nostre estimador per donar com a casos negatius els casos realment sans; o proporció de sans correctament identificats.
- **Precisió**: grau de concordança entre les indicacions obtingudes per mesures repetides en els mateixos objectes o similars i en condicions especifiques.

4.4.2 Sel · lecció dels paràmetres kernel, hiperparàmetres i model

cross-validation

És una tècnica utilitzada per avaluar els resultats d'una anàlisi estadística i garantir que són independents de la partició entre dades d'entrenament i prova. Consisteix a repetir i calcular la mitjana aritmètica obtinguda de les mesures d'avaluació sobre diferents particions [28].
Pel que fa el paquet **kebabs**, aquest proporciona tres variants de *cross-validation*: *k-fold cross-validation*, *Leave-One-Out cross-validation*, *Grouped cross-validation*.

- ***k-fold cross-validation*:** per aquest tipus el conjunt de dades es divideix en k parts més o menys iguals anomenades *folds*. Un d'aquests *folds* es fa servir com a conjunt de prova, els altres $k-1$ *folds* com a conjunts d'entrenament. En les $k-1$ execucions d'entrenament el *fold* de prova es canvia cíclicament donant diferents valors de rendiment k en diferents subconjunts de les dades.
  Aquesta variant és sel · leccionada en el paràmetre **cross** de la funció **kbsvm**, aquest defineix el nombre màxim de *folds*, aquest nombre ha de ser major que 1.

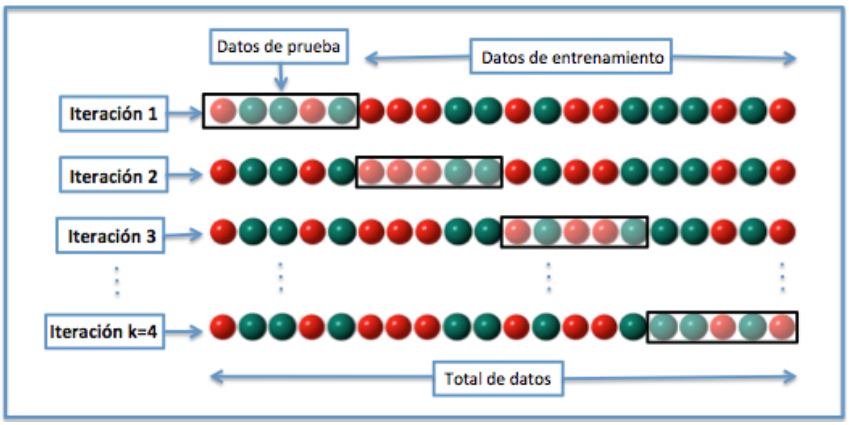


Figura 4.2: *4-fold cross-validation* [28]

- ***Leave-One-Out cross-validation*:** Aquesta variant és l'anterior però portada a l'extrem, amb una mida de *fold* per una sola mostra. Aquesta variant és dinàmicament bastant exigent i amb el mateix nombre d'execucions d'entrenament que mostres en el conjunt de dades.
  Per especificar aquesta validació el paràmetre **cross** ha de ser igual a -1.

- ***Grouped cross-validation*:** Per aquesta variant les mostres han de ser assignades en un grup abans d'executar la *cross-validation*. Aquesta variant és una versió especial de *k-fold cross-validation* perque respecta la pertineça a un grup quan la base de dades es divideix en *folds*, els grups s'han de formar abans de fer la *cross-validation*.

Pel que fa a aquesta variant s'han d'especificar dos paràmetres: **groupBy**, amb l'assignació de les mostres a un grup i el paràmetre **cross** que significa el nombre màxim de *folds* i en aquest cas ha de ser més petit o igual a el nombre de grups.

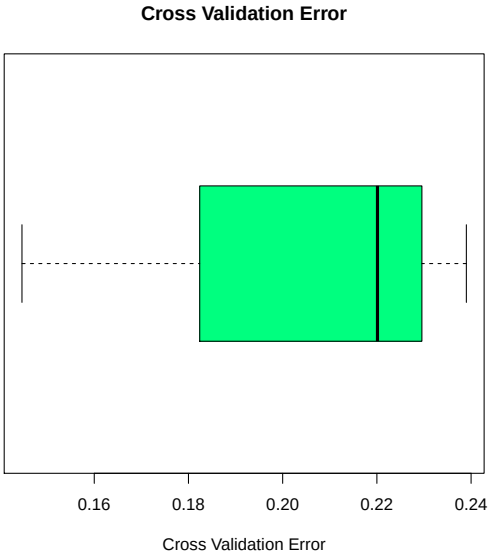
Pel que fa als resultats obtinguts en la *cross-validation* es poden veure mitjançant la funció implementada en el paquet **kebabs**, **cvResult**, en aquesta funció se li ha de passar com atribut d'entrada un objecte de tipus **kbsvm**. En el cas que es volguès fer repeticions de la *cross-validation* s'hauria d'implementar l'atribut **noCross** amb el nombre de repeticions.

A continuació és pot veure l'especificació de la *k-fold cross-validation* en la funció **kbsvm** de R. La base de dades que es fa servir és CCoil[26] implementada en R aquesta està formada per 477 cadenes d'aminoàcids amb les seves corresponents etiquetes, un kernel de tipus gappy amb longitud fixa de la primera part de la subcadena $k = 1$ i el nombre màxim de comodins $m = 6$, una *k-fold cross-validation* amb 3 *folds* i un cost de 30.

```
> data(CCoil)
> x <- ccseq
> y <- as.numeric(yCC)
> gappyK1M6 <- gappyPairKernel(k=1, m=6)
> model <- kbsvm(x=x, y=y, kernel=gappyK1M6,
+                pkg="kernlab", svm="C-svc", cost=30, cross=3)
> cvResult(model)

Cross validation result object of class  "CrossValidationResult"

cross                 : 3
noCross               : 1

CV error:             : 0.20125786

Fold CV errors:
0.23899371, 0.14465409, 0.22012579

No of support vectors:
215, 227, 212
```

A partir de la funció anterior s'obtè l'error mitjà de la *cross-validation*, així com l'error corresponent a cada *fold*. A partir del gràfic següent es pot veure un *boxplot* corresponent als errors, amb el mínim, la mediana i el màxim. Aquest gràfic s'obte mitjançant la funció **plot** amb el paràmetre d'entrada **cvResult**, tal com es veu a continuació.

```
> plot(cvResult(model))
```



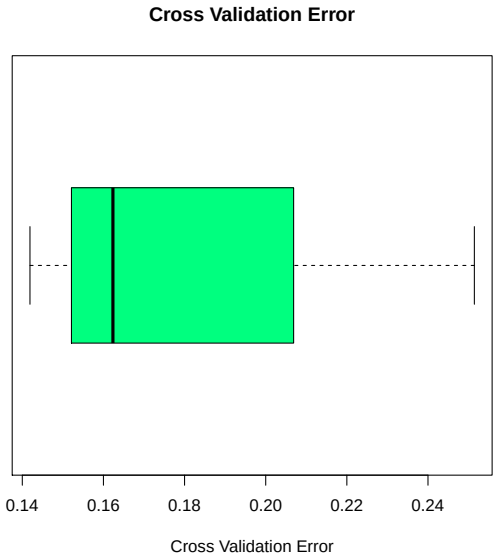
A continuació es farà el mateix que s'ha fet en l'apartat anterior però en comptes de la *k-fold cross-validation* amb la *Grouped cross-validation*. Pel que fa els grups estan implementats en la funció **ccgroups**, aquests grups es van determinar a través un sol vincle d'agrupació de similituds entre seqüències derivat d'un alineament de seqüències de tipus *ungapped heptadspecific pairwise*. A continuació es poden veure els grups i els resultats de la *Grouped cross-validation*.

```
> head(ccgroups)

PDB1 PDB2 PDB3 PDB4 PDB5 PDB6
  45    2   44   43   38   42

> model2 <- kbsvm(x=x, y=y, kernel=gappyK1M6,
+                 pkg="kernlab", svm="C-svc", cost=30, cross=3, groupBy=ccgroups)
> cvResult(model2)

Cross validation result object of class  "CrossValidationResult"

cross                : 3
noCross              : 1

CV error:            : 0.18521938

groupBy:
45,2,44 ... 154,47,121
```

```
Fold CV errors:
0.16233766, 0.14189189, 0.25142857

No of support vectors:
215, 219, 207
```

Com es pot veure en la anterior sortida de resultats, el tipus de resultats són els mateixos que s'han obtingut en l'apartat de la *k-fold cross-validation*, l'error mitjà de la *cross-validation*, així com l'error corresponent a cada *fold*. A continuació es mostra el *boxplot* amb als errors, amb el mínim, la mediana i el màxim.

```
> plot(cvResult(model2))
```



Grid search

El *Grid search* proporciona una visió àmplia del comportament del rendiment per diferents especificacions dels paràmetres i ajuda a l'usuari en la sel · lecció en la seqüència kernel, els valors òptims dels paràmetres kernel i els hiperparàmetres del *support vector machine*. La *grid* es defineix com una estructura de dades rectangular, on les files corresponen a les diferents seqüències kernel incloent els seus paràmetres kernel i les columnes corresponen als pàrametres del *support vector machine*.

Pel que fa al paquet **kebabs**, aquest presenta una funció la qual t'ajuda en la sel · lecció dels paràmetres esmentats. Aquesta funció s'anomena **modelSelResult** i l'objecte que se li ha de passar d'entrada ha de ser de tipus **kbsvm**.

A continuació es realitza un exemple de *grid search*, amb la base de dades CCoil utilitzada en l'exemple anterior, en aquest cas com es un mètode de sel · lecció s'han d'específicar més d'un kernel, en aquest cas s'han fet servir els kernels de tipus spectrum amb $k = 2$, $k = 3$ i $k = 4$, i els kernels de tipus gappy pair amb $k = 1$ i $m = 2$, $k = 1$ i $m = 3$, i $k = 1$ i $m = 4$, un vectorde costos igual a (1,10, 100, 1000, 10000) i una *k-fold cross-validation* amb 3 *folds*.

```
> data(CCoil)
> x <- ccseq
> y <- as.numeric(yCC)
> specK24 <- spectrumKernel(k=2:4)
> gappyK1M24 <- gappyPairKernel(k=1, m=2:4)
> gridKernels <- c(specK24, gappyK1M24)
> cost <- c(1,10, 100, 1000, 10000)
> model <- kbsvm(x=x, y=y, kernel=gridKernels,
+                pkg="kernlab", svm="C-svc", cost=cost, cross=3)
> modelSelResult(model)

Grid search result object of class  "ModelSelectionResult"

cross                : 3
noCross              : 1
Smallest CV error    : 0.16771488

Grid Rows:
Kernel_1    SpectrumKernel: k=2
Kernel_2    SpectrumKernel: k=3
Kernel_3    SpectrumKernel: k=4
Kernel_4    GappyPairKernel: k=1, m=2
Kernel_5    GappyPairKernel: k=1, m=3
Kernel_6    GappyPairKernel: k=1, m=4

Grid Columns:
           cost
GridCol_1     1
GridCol_2    10
GridCol_3   100
GridCol_4  1000
GridCol_5 10000
```

```
Grid Errors:

         GridCol_1 GridCol_2 GridCol_3 GridCol_4 GridCol_5
Kernel_1 0.2431866 0.3333333 0.3501048 0.3794549 0.3354298
Kernel_2 0.1865828 0.2138365 0.2096436 0.2410901 0.2809224
Kernel_3 0.1865828 0.1677149 0.1865828 0.3270440 0.1865828
Kernel_4 0.1781971 0.2285115 0.2662474 0.2201258 0.2306080
Kernel_5 0.1844864 0.2012579 0.2306080 0.3312369 0.3228512
Kernel_6 0.1719078 0.2180294 0.3375262 0.2138365 0.2557652

Selected Grid Row:
SpectrumKernel: k=4

Selected Grid Column:
          cost
GridCol_2   10
```

Es pot veure que la funció anterior proporciona com a resultats, el menor error aconseguit de tots els mètodes kernels amb els costos corresponents, una matriu amb l'error de cada mètode kernel amb el cost corresponent i finalment et diu quin són el mètode kernel i el cost sel · leccionats per realitzar la classificació amb el menor error possible.

Sel · lecció del model

Pel que fa la manera de funcionar en la sel · lecció del model és molt semblant a la mecànica del *grid search*, però amb la principal diferència que la sel · lecció del model és basa en la *neasted cross-validation* i *grid search* és basa en la *cross-validation*.
El terme *nested cross-validation* fa referència a un bucle de *cross-validation* interna que determina el millor ajustament dels hiperparàmetres i un bucle de *cross-validation* externa per provar el rendiment del model amb els millors valors dels paràmetres en els conjunts de proves independents.

Pel que fa a la sel · lecció del model, s'han d'específicar dos atributs: **nestedCross** el qual fa referència a la *cross-validation* externa i ha de ser major que 1 i **cross** depenent de la *cross-validation* interna que es vulgui fer. En el cas que es volguès fer repeticions de la *neasted cross-validation* s'hauria d'implementar l'atribut **noNestedCross** amb el nombre de repeticions.
Ara es fa el mateix exemple que s'ha fet anteriorment però incloent la *neasted cross-validation*. Per tant a l'específicació que teniem abans s'ha d'afegir l'atribut **nestedCross**

que en aquest cas s'iguala a 4, també es canvia el nombre de costos perque no hi hagi tantes iteracions.

```
> data(CCoil)
> x <- ccseq
> y <- as.numeric(yCC)
> specK24 <- spectrumKernel(k=2:4)
> gappyK1M24 <- gappyPairKernel(k=1, m=2:4)
> gridKernels <- c(specK24, gappyK1M24)
> cost <- c(10, 50, 100)
> model <- kbsvm(x=x, y=y, kernel=gridKernels,
+                pkg="kernlab", svm="C-svc", cost=cost, cross=3, nestedCross=4)
> modelSelResult(model)

Model selection result object of class  "ModelSelectionResult"

cross                : 3
noCross              : 1
nestedCross          : 4
noNestedCross        : 1

Grid Rows:
Kernel_1    SpectrumKernel: k=2
Kernel_2    SpectrumKernel: k=3
Kernel_3    SpectrumKernel: k=4
Kernel_4    GappyPairKernel: k=1, m=2
Kernel_5    GappyPairKernel: k=1, m=3
Kernel_6    GappyPairKernel: k=1, m=4

Grid Columns:
          cost
GridCol_1   10
GridCol_2   50
GridCol_3  100

Selected Grid Row:
1  SpectrumKernel: k=4
2  SpectrumKernel: k=4
3  SpectrumKernel: k=4
4  SpectrumKernel: k=4
```

```
Selected Grid Column:
  selGridCol cost
1  GridCol_1   10
2  GridCol_3  100
3  GridCol_2   50
4  GridCol_1   10
```

Els resultats obtinguts a partir de la funció implementada anteriorment són: una llista ordenada de menor a major error amb el mètode kernel i el cost corresponent, a partir d'aquesta llista es podrà sel · leccionar el model més adequat. Aquest model és:

```
> c(selGridRow(modelSelResult(model))[[1]],selGridCol(modelSelResult(model))[1,])

[[1]]
Spectrum Kernel: k=4

$selGridCol
[1] GridCol_1
Levels: GridCol_1 GridCol_3 GridCol_2

$cost
[1] 10
```

Capítol 5

APLICACIÓ DEL SUPPORT VECTOR MACHINE A DADES ÒMIQUES

5.1 Descripció de la base de dades

The UniProt Knowledgebase (UniProtKB) [16] és l'eix central per a la recollida d'informació funcional sobre les proteïnes , amb l'anotació precisa, coherent i rica . A més a més de les dades bàsiques obligatòries per a cada entrada UniProtKB, aquestes són: la seqüència d'aminoàcids , el nom de la proteïna o descripció , dades taxonòmiques, així com s'afegeix la major informació possible d'anotació.
The UniProt Knowledgebase (UniProtKB) consta de dues seccions : una secció que conté els registres de forma manual amb informació extreta de la literatura i un curós anàlisi computacional. I una altra secció la qual els registres estan incomplets els quals se'ls hi pot introduïr informacó faltant manualment. La primera secció s'anomena UniProtKB/Swiss-Prot i la segona s'anomena UniProtKB/TrEMBL.

En aquest treball es fa servir una base de dades extreta de la primera secció UniProtKB/Swiss-Prot. La base esta formada per les cadenes de proteïnes i la seva localització subcel · lular, aquestes són: cloroplastica i nucleotica. Per tant la base total està composta per:

- 14714 proteïnes amb localització subcellular cloroplastica.
- 39448 proteïnes amb localització subcellular nucleotica.

Per fer la classificació no s'han fet servir totes les 54162 proteïnes, en aquest cas s'han sel · leccionat 100 proteïnes aleatoriament. La raó de perquè només s'han agafat 100 proteïnes en comptes de les 54162 és perquè el temps de CPU augmenta exponencialment quan s'augmenta la mida de la mostra.

5.2 Aplicació del *support vector machine*

En aquest apartat el que es farà es classificar les proteïnes de la base de dades explicada anteriorment, mitjançant la tècnica de support vector machine a partir de tots els tipus de kernel que s'han esmentat anteriorment.

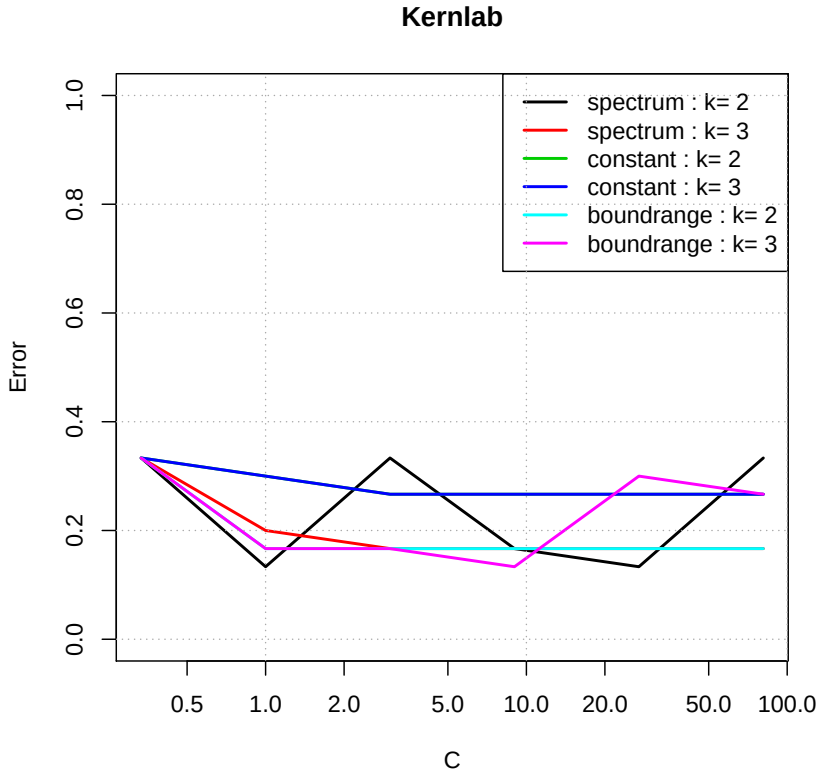
5.2.1 Kernlab

Classificació

En aquesta secció s'ha fet la classificació amb el mètode de vector support machine pels kernels implementats en el paquet d'R kernlab, aquests kernels són: spectrum, boundrange, constant i exponential. Primer de tot el que s'ha fet és crear una funció la qual se li ha d'especificar 5 atributs: el fitxer d'entrada **x** amb les cadenes de proteïnes, el vector **y** amb les etiquetes corresponents al fitxer d'entrada **x**, un vector de strings el qual indiqui el tipus de kernel pel qual es vol fer la classificació, aquest pot tenir com llargada mínima igual a 1 (això vol dir que només vols fer la classificació per un kernel) i llargada màxima 4 (en aquest cas estàs comparant tots els tipus de kernel), un vector **ks** on indiqui la longitud o les longituds de les subcadenes, el cas que es vulgui realitzar la classificació pel tipus kernel exponential s'haurà d'especificar un vector **ls** que correspon al paràmetre λ, en cas contrari aquest paràmetre s'haurà d'especificar com a 0, percentatge de dades equivalent a la base de dades d'entrenament **prop**. Per últim un vector **clist** amb els diferents costos pel qual es vol realitzar la classificació.

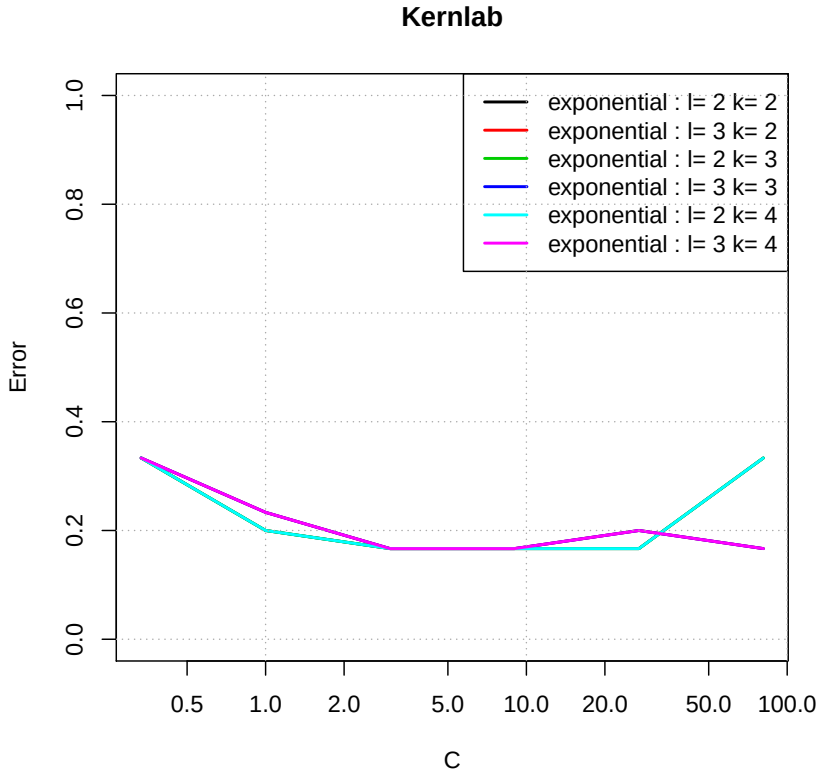
Aquesta funció proporciona com a resultat un gràfic de lines el qual indica quin ha estat l'error total de classificació de cada tipus de kernel per cada cost específicat.
A continuació es poden veure els resultats que s'ha obtingut si s'especifiquen els següents atributs:
Cas 1.

- **x**: Base de dades UniProtKB.
- **y**: Vector d'etiquetes corresponent a la base de dades UniProtKB.
- **type**: Un vector on inclou els quatre tipus de kernel implementats en el paquet d'R kernlab $spectrum, constant, boundrange$.
- **ks**: longitud de la subcadena = {2,3}.
- **ls**: λ = {2,5}.
- **prop**: proporció de dades en la base de dades d'entrenament = 0.7.
- **clist**: {0.33, 1, 3, 9, 27, 81}.



Cas 2.

- **x**: Base de dades UniProtKB.
- **y**: Vector d'etiquetes corresponent a la base de dades UniProtKB.
- **type**: Un vector on inclou els quatre tipus de kernel implementats en el paquet d'R kernlab *exponential*.
- **ks**: longitud de la subcadena = {2,3}.
- **ls**: λ = {2,3,5}.
- **prop**: proporció de dades en la base de dades d'entrenament = 0.7.
- **clist**: {0.33, 1, 3, 9, 27, 81}.



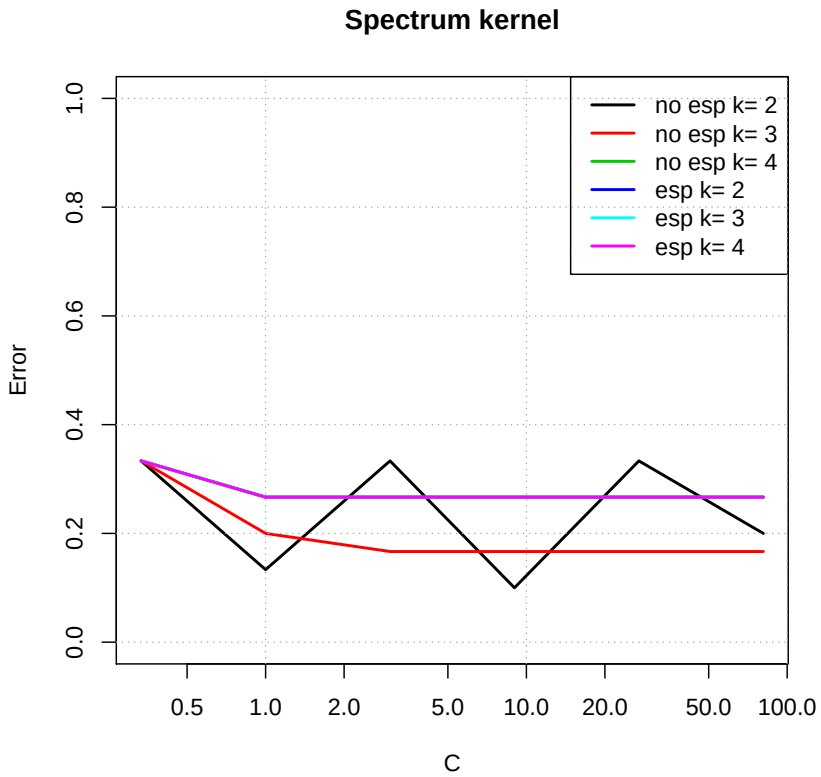
5.2.2 Kebabs

Kebabs: Spectrum kernel

Tal i com s'ha explicat en l'apartat kebabs del capítol 3, pel paquet kebabs es pot fer la classificació determinant la posició de la subseqüència dependent o independent. Si es determina posició independent, les subseqüències es consideraran coincidència totes aquelles que siguin iguals, independentment d'on es trobin en la cadena. Pel cas de posició dependent, les subseqüències es consideraran coincidència aquelles que siguin igual i depenentment d'on es trobin en la cadena. En aquest apartat s'ha realitzat la classificació de la base de dades que s'ha descrit en el principi del capítol, pel mètode de vector support machine que està implementat en el paquet d'R kebabs. En aquesta secció només s'ha fet servir el kernel de tipus spectrum. Per fer aquesta classificació s'ha creat una funció la qual els atributs d'entrada són: la base d'entrada **x**, el vector d'etiquetes **y** corresponent a la base d'entrada **x**, un vector **ks** on indiqui les longituds de les subcadenes per les quals es vol fer la classificació, un vector numèric **pos** el qual indica si es vol fer la classificació amb posició independent, posició dependent específica i/o posició dependent pondreada, aquest vector és numèric per tant posició independent equival a 0 i posició dependent específica equival a 1, aquest vector es pot combinar de la forma que un prefereixi, percentatge de dades equivalent a la base de dades d'entrenament **prop**.
Finalment l'últim atribut és un vector de reals **clist** corresponent als diferents costos pel

qual es vol realitzar la classificació.
Aquesta funció proporciona com a resultat un gràfic de lines el qual indica quin ha estat l'error total de classificació de cada tipus de kernel per cada cost específicat. A continuació es poden veure els resultats que s'ha obtingut si s'especifiquen els següents atributs: Cas 1.

- **x**: Base de dades UniProtKB.
- **y**: Vector d'etiquetes corresponent a la base de dades UniProtKB.
- **ks**: longitud de la subcadena = {2,3}.
- **pos**: posició independent o dependent ={0,1}, en aquest cas s'agafen els dos casos.
- **prop**: proporció de dades en la base de dades d'entrenament = 0.7.
- **clist**: {0.33, 1, 3, 9, 27, 81}.



Kebabs: Gappy pair kernel

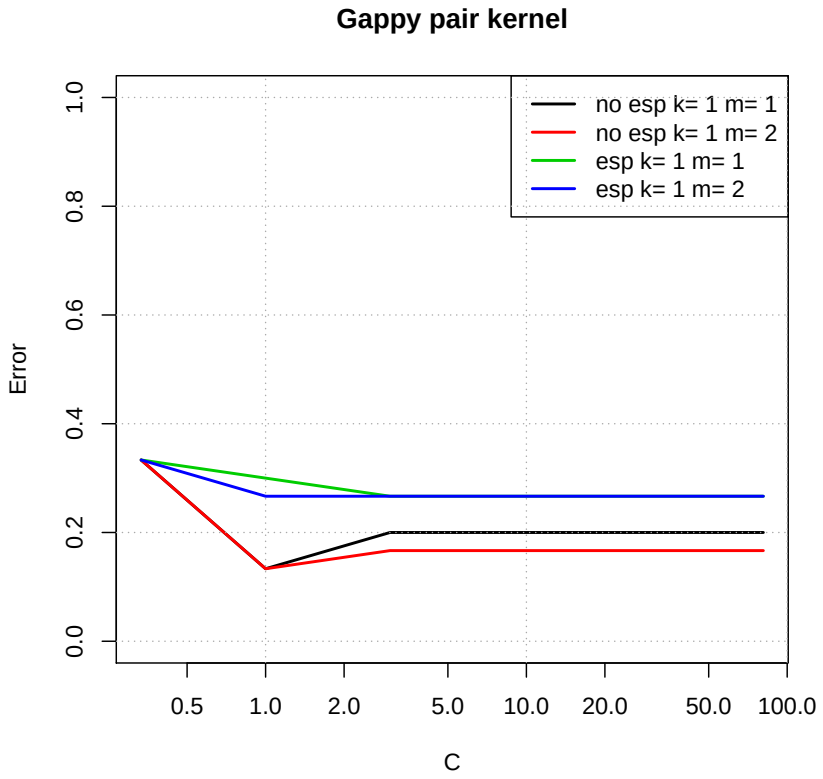
En aquest apartat també es realitzarà una classificació tal i com s'ha fet en l'apartat anterior però en comptes de fer servir el kernel de tipus spectrum, s'utilitza el kernel de tipus gappy pair. La principal diferència que presenta aquest kernel en vers el kernel anterior és que aquest s'implementa a partir de dos paràmetres, el paràmetre k que equival

a la la longitud fixa de la primera part de la combinació així com de la segona part i el paràmetre m que equival al nombre màxim que hi pot haver de posicions comodí. Per tant els atributs que s'hauran d'especificar són els mateixos que en l'anterior funció més el paràmetre m.

Els atributs d'entrada són: la base d'entrada **x**, el vector d'etiquetes **y** corresponent a la base d'entrada **x**, un vector **ks** on indiqui les longituds fixes de la primera i segona part de les subcadenes per les quals es vol fer la classificació, un vector numèric **ms** el qual indiqui el nombre màxim de comodins permesos, un vector numèric **pos** el qual indica si es vol fer la classificació amb posició independent, posició dependent específica i/o posició dependent pondreada, aquest vector és numèric per tant posició independent equival a 0 i posició dependent específica equival a 1, aquest vector es pot combinar de la forma que un prefereixi, percentatge de dades equivalent a la base de dades d'entrenament **prop**. Finalment l'últim atribut és un vector de reals **clist** corresponent als diferents costos pel qual es vol realitzar la classificació.

Aquesta funció proporciona com a resultat un gràfic de lines el qual indica quin ha estat l'error total de classificació de cada tipus de kernel per cada cost específicat. A continuació es poden veure els resultats que s'ha obtingut si s'especifiquen els següents atributs:
Cas 1.

- **x**: Base de dades UniProtKB.
- **y**: Vector d'etiquetes corresponent a la base de dades UniProtKB.
- **ks**: longitud fixa de la primera i segona part de la subcadena = {1,2}.
- **ms**: nombre màxim de comodins = {1,2}
- **pos**: posició independent o dependent ={0,1}, en aquest cas s'agafen els dos casos.
- **prop**: proporció de dades en la base de dades d'entrenament = 0.7.
- **clist**: {0.33, 1, 3, 9, 27, 81}.



Kebabs: Mismatch kernel

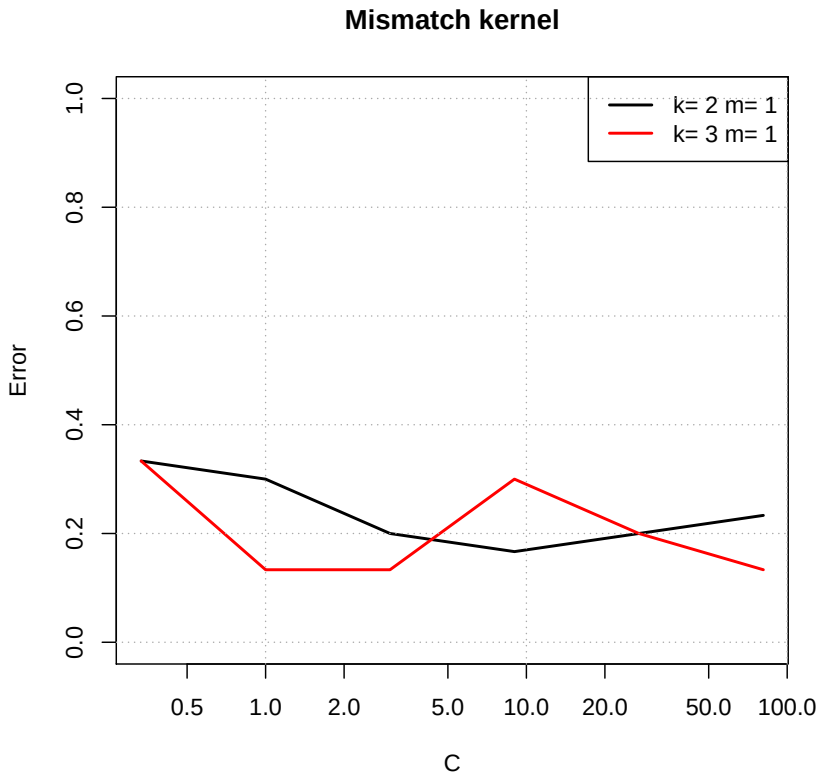
Seguin en la mateixa dinàmica que s'ha estat utilitzant al llarg d'aquest capítol, en aquest apartat es torna a crear una altra funció que realitza una classificació de la base d'entrada explicada al principi del capítol però amb el mismatch kernel com a funció kernel.

Els atributs d'entrada són: la base d'entrada **x**, el vector d'etiquetes **y** corresponent a la base d'entrada **x**, un vector **ks** on indiqui les longituds de les subcadenes per les quals es vol fer la classificació, un vector numèric **ms** el qual indiqui el nombre màxim de desajustos, recordar que aquest paràmetre m ha de ser sempre inferior al paràmetre k, aquest vector és numèric per tant posició independent equival a 0 i posició dependent específica equival a 1, aquest vector es pot combinar de la forma que un prefereixi, percentatge de dades equivalent a la base de dades d'entrenament **prop**. Finalment l'últim atribut és un vector de reals **clist** corresponent als diferents costos pel qual es vol realitzar la classificació. Pel que fa al mismatch kernel no es pot definir posició, per tant tots els que es realitzaràn amb aquest kernel seràn de posició independent.

Aquesta funció proporciona com a resultat un gràfic de lines el qual indica quin ha estat l'error total de classificació de cada tipus de kernel per cada cost específicat. A continuació es poden veure els resultats que s'ha obtingut si s'especifiquen els següents

atributs:
Cas 1.

- **x**: Base de dades UniProtKB.
- **y**: Vector d'etiquetes corresponent a la base de dades UniProtKB.
- **ks**: longitud de la subcadena = {2,3}.
- **ms**: nombre màxim de desajustos = {1,2}.
- **clist**: {0.33, 1, 3, 9, 27, 81}.



5.2.3 Evaluació de la classificació

En l'apartat anterior s'ha realitzat una classificació de la base de dades que s'ha descrit en el principi d'aquest capítol i d'una manera bastant visual es pot veure quin mètode és el que obté un menor error i per tant el més adient per classificar amb millor precisió, de totes maneres tal i com s'ha vist en el quart capítol el paquet **kebabs** conté dues eines molt útils per poder veure quin és el model més adient per fer la classificació i la predicció de les proteïnes.
Per tant s'utilitzaran les funcions descrites en el capítol 4 per obtenir el millor model en *cross-validation* i *nested cross-validation*.

Grid search: *cross-validation*

Tal i com s'ha explicat en el principi d'aquesta secció, l'objectiu principal és trobar el millor mètode kernel amb els paràmetres i cost adients per tal d'aconseguir una classificació de la base de dades amb un error mínim. Aquest procediment es realitzarà mitjançant la funció implementada en el paquet kebabs **kbsvm** amb l'especificació de l'atribut **cross** que equival al nombre de *folds*.

Els atributs són els següents:

- **x**: Base de dades UniProtKB.
- **y**: Vector d'etiquetes corresponent a la base de dades UniProtKB.
- **kernel**:
  - Spectrum kernel $k = \{2, 3, 4, 5\}$ amb posició independent.
  - Spectrum kernel $k = \{2, 3, 4, 5\}$ amb posició dependent específica.
  - Gappy pair kernel $k = 1$, $m = \{1, 2\}$ amb posició independent.
  - Gappy pair kernel $k = 1$, $m = \{1, 2\}$ amb posició dependent específica.
  - Mismatch kernel $k = \{2, 3\}$, $m = 1$ amb posició independent.
  - Mismatch kernel $k = \{2, 3\}$, $m = 1$ amb posició dependent específica.
- **cross**: Nombre de *folds* = 10, per tant es fa servir una *k-fold cross-validation.*
- **cost**: vector de costos {0.33, 1, 3, 9, 27, 81}.
- **noCross**: en aquest cas no es farà cap repetició, per tant no cal implementar aquest atribut.

```
Grid search result object of class  "ModelSelectionResult"

cross                : 10
noCross              : 1
Smallest CV error    : 0.15

Grid Rows:
Kernel_1    SpectrumKernel: k=2
Kernel_2    SpectrumKernel: k=3
Kernel_3    SpectrumKernel: k=4
```

```
Kernel_4    SpectrumKernel: k=5
Kernel_5    SpectrumKernel: k=2, distWeight=1
Kernel_6    SpectrumKernel: k=3, distWeight=1
Kernel_7    SpectrumKernel: k=4, distWeight=1
Kernel_8    SpectrumKernel: k=5, distWeight=1
Kernel_9    GappyPairKernel: k=1, m=1
Kernel_10   GappyPairKernel: k=1, m=2
Kernel_11   GappyPairKernel: k=1, m=1, distWeight=1
Kernel_12   GappyPairKernel: k=1, m=2, distWeight=1
Kernel_13   MismatchKernel: k=2, m=1
Kernel_14   MismatchKernel: k=3, m=1

Grid Columns:
               cost
GridCol_1  0.3333333
GridCol_2  1.0000000
GridCol_3  3.0000000
GridCol_4  9.0000000
GridCol_5 27.0000000
GridCol_6 81.0000000

Grid Errors:

          GridCol_1 GridCol_2 GridCol_3 GridCol_4 GridCol_5 GridCol_6
Kernel_1       0.34      0.26      0.25      0.24      0.23      0.23
Kernel_2       0.34      0.30      0.31      0.29      0.29      0.31
Kernel_3       0.34      0.30      0.30      0.30      0.30      0.30
Kernel_4       0.34      0.30      0.30      0.30      0.30      0.30
Kernel_5       0.34      0.30      0.30      0.30      0.30      0.30
Kernel_6       0.34      0.30      0.30      0.32      0.30      0.30
Kernel_7       0.34      0.30      0.32      0.30      0.30      0.30
Kernel_8       0.34      0.30      0.30      0.32      0.30      0.30
Kernel_9       0.34      0.17      0.26      0.22      0.26      0.23
Kernel_10      0.33      0.15      0.26      0.23      0.24      0.30
Kernel_11      0.34      0.30      0.30      0.30      0.30      0.30
Kernel_12      0.34      0.30      0.30      0.32      0.30      0.30
Kernel_13      0.34      0.28      0.22      0.17      0.17      0.27
Kernel_14      0.34      0.18      0.15      0.31      0.25      0.29
```

```
Selected Grid Row:
GappyPairKernel: k=1, m=2

Selected Grid Column:
          cost
GridCol_2    1
```

sel · lecció del model: *nested cross-validation*

Un cop s'ha fet la *cross-validation* es realitza la *nested cross-validation*, la principal diferència que hi ha entre aquestes és que la *cross-validation* realitza una validació creuada interna i la *nested cross-validation* fa una validació creuada interne i externa, això s'explica més detalladament en el capítol 4.
Aquest procediment es realitzarà mitjançant la funció implementada en el paquet **kebabs** **kbsvm** amb l'especificació dels atributs, **cross** que equival al nombre de *folds* en la *cross-validation* interna i **nestedCross** que equival al nombre de *folds* en la *cross-validation* externa.

Els atributs corresponents són els següents:

- **x**: Base de dades UniProtKB.
- **y**: Vector d'etiquetes corresponent a la base de dades UniProtKB.
- **kernel**:
  - Spectrum kernel $k = \{2, 3, 4, 5\}$ amb posició independent.
  - Spectrum kernel $k = \{2, 3, 4, 5\}$ amb posició dependent específica.
  - Gappy pair kernel $k = 1$, $m = \{1, 2\}$ amb posició independent.
  - Gappy pair kernel $k = 1$, $m = \{1, 2\}$ amb posició dependent específica.
  - Mismatch kernel $k = \{2, 3\}$, $m = 1$ amb posició independent.
  - Mismatch kernel $k = \{2, 3\}$, $m = 1$ amb posició dependent específica.
- **cross**: Nombre de *folds* en la *cross-validation* interna = 10, per tant es fa servir una *k-fold cross-validation*.
- **nestedCross**: Nombre de *folds* en la *cross-validation* externa = 4.
- **cost**: vector de costos {0.33, 1, 3, 9, 27, 81}.

- **noCross** i **noNestedCross** : en aquest cas no es farà cap repetició, per tant no cal implementar aquests atributs.

```
Model selection result object of class  "ModelSelectionResult"

cross                : 10
noCross              : 1
nestedCross          : 4
noNestedCross        : 1

Grid Rows:
Kernel_1    SpectrumKernel: k=2
Kernel_2    SpectrumKernel: k=3
Kernel_3    SpectrumKernel: k=4
Kernel_4    SpectrumKernel: k=5
Kernel_5    SpectrumKernel: k=2, distWeight=1
Kernel_6    SpectrumKernel: k=3, distWeight=1
Kernel_7    SpectrumKernel: k=4, distWeight=1
Kernel_8    SpectrumKernel: k=5, distWeight=1
Kernel_9    GappyPairKernel: k=1, m=1
Kernel_10   GappyPairKernel: k=1, m=2
Kernel_11   GappyPairKernel: k=1, m=1, distWeight=1
Kernel_12   GappyPairKernel: k=1, m=2, distWeight=1
Kernel_13   MismatchKernel: k=2, m=1
Kernel_14   MismatchKernel: k=3, m=1

Grid Columns:
               cost
GridCol_1  0.3333333
GridCol_2  1.0000000
GridCol_3  3.0000000
GridCol_4  9.0000000
GridCol_5 27.0000000
GridCol_6 81.0000000

Selected Grid Row:
1  GappyPairKernel: k=1, m=2
2  SpectrumKernel: k=2
3  GappyPairKernel: k=1, m=1
```

```
4  MismatchKernel: k=3, m=1

Selected Grid Column:
  selGridCol cost
1  GridCol_2    1
2  GridCol_6   81
3  GridCol_6   81
4  GridCol_3    3
```

A partir dels resultats anteriors es pot veure quin és el model més adequat per realitzar la classificació, per tant s'haura d'escollir el model que està en primera posició de la llista *selected grid row*, amb el corresponent error que és el que està en primera posició de la llista *selected grid column*. En el cas de la *nested cross-validation* és:

```
[[1]]
gappy pair kernel: k=1, m=2

[[2]]
[1] 1
```

Predicció Un cop s'ha sel · leccionat el model es realitzarà la predicció i es farà l'avaluació de la predicció del model que s'ha escollit anteriorment. Cal específicar que l'etiqueta 1 equival a localització sub · lular cloroplastica, l'etiqueta 2 eqival a localització sub · lular nucleotica. Per fer la predicció s'han fet servir el 70% de 5000 proteïnes sel · leccionades aleatoriament com a base de dades d'entrenament i la resta com a base de dades test.

```
     2   1
2 1069  97
1   18 316

Accuracy:              92.333% (1385 of 1500)
Balanced accuracy:     87.429% (1069 of 1087 and 316 of 413)
Matthews CC:            0.804

Sensitivity:           98.344% (1069 of 1087)
Specificity:           76.513% (316 of 413)
Precision:             91.681% (1069 of 1166)
```

5.3 Resultats

Pel que fa a l'apartat del paquet **kernlab** es pot veure que el model que ha aconseguit el mínim error en les prediccions és un kernel de tipus spectrum, amb $k = 2$ i cost 1,

cal destacar que es poden comparar els models perque tots han estat executats amb la mateixa mostra de dades, tant la mostra total com la base d'entrenament. A més a més cal comentar que si en els gràfics s'observen menys línies de les que hi ha a la llegenda és perquè diferents tipus de kernel presenten el mateix error per diferents costos.

Pel cas de l'apartat del paquet **kebabs** es pot veure en els gràfics que el model que presenta l'error mínim és el kernel de tipus spectrum amb posició no específica, amb hiperparàmetres $k = 2$ i amb cost igual a 9.

pel que fa a la *grid search* és pot observar que el model amb menys error és el kernel de tipus gappy pair amb hiperparàmetres $k = 1$ i $m = 2$ i cost 1, ja que es el que ha obtingut menys tant per cent d'error en la predicció, 15% d'error, llavors pel que fa a la sel · lecció del model o *nested cross-validation* es pot veure que el model amb un error mínim és kernel de tipus gappy pair amb hiperparàmetres $k = 1$ i $m = 2$ i cost 1. Es pot veure que tant per la *grid search* i la sel · lecció del model s'ha obtingut com a millor model, el mateix tipus de kernel, amb els mateixos hiperparàmetres i el mateix cost.

Finalment en l'ultim apartat on es realitza la predicció amb el model sel · leccionat a l'apartat de *grid search* i *nested cross-validation*, es pot veure que la predicció obté una *accuracy* del 92.33%, una *balanced accuracy* del 87.429%, un coeficient de correlació de Matthews igual a 0.804, això significa que la predicció és bastant bona ja que és bastant pròxima 1, una sensibilitat del 98.344% això vol dir que s'han identificat 1069 cadenes de tipus nucleotic de les 1087 totals, una especificitat del 76.513% això significa que s'han identificat 316 de les 413 cadenes de tipus cloroplastic i una precisió del 91.681%. Sembla ser que en la mostra de test hi ha moltes més cadenes de tipus nucleotic que de tipus cloroplastic. Aquests resultats estàn desbanlancejats.

Capítol 6

CONCLUSIONS

6.1 Resultats obtinguts

El present treball s'ha realitzat amb l'objectiu general d'obtenir una visió global de l'aplicaió de l'estadística al camp de la bioinformàtica.
Aquest objectiu s'ha volgut plantejar en els capítols dos i tres d'aquesta memòria. En el segon capítol s'ha volgut mostrar les idees bàsiques de la biologia mol · lecular, com per exemple la producció de proteïnes, el qual és un punt bastant important per poder endendre la importància de la seva classificació. En el segon capítol també es planteja que són les dades òmiques i la importància de la bioinformàtica per poder-les treballar més fàcilment.

En el tercer capítol es vol ensenyar la idea bàsica de l'aprenetatge automàtic i amb més profunditat els mètodes kernel per strings, així com la seva implementació en el software R. En aquest capítol s'ha pogut veure que aquest software presenta dos eines molt potetes pel que fa a l'aplicació dels mètodes kernel com són el paquet **kernlab** i el paquet **kebabs**, cal destacar que durant l'elaboració d'aquesta memòria s'ha pogut veure que el segon és més complet i amb més variants que el primer.

A partir de que s'ha obtingut una idea general de la bioinformàtica i de l'aprenentatge automàtic, aquesta memòria vol explicar les eines necessàries per fer una classificació, aquesta aplicació tè en compte les dues tècniques esmentades anteriorment, la bioiformàtica i l'aprenentatge automàtic.
En el quart capítol es pot veure que hi ha molts mètodes de classificació, però el que s'ha fet servir en aquesta memòria és el *support vector machine*.

Finalment, en últim capítol s'ha realitzat un cas pràctic amb una base proteòmica real, en aquest capítol s'ha pogut veure diferentes classificacions de proteïnes segons on es troben a la cèllula amb *support vector machine* per diferents tipus de kernels, diferents

paràmetres i diferents costos tant pel paquet **kernlab** com pel paquet **kebabs**. En aquest capítol també s'ha pogut veure quin és el millor model per realitzar la classificació i la seva predicció.

Com a conclusió general, es pot dir que la classificació de proteïnes segons on es troben a la cèl · lula és un tema molt extens, molt important pel quqe fa en el món de la biòlogia mollecular i que el software R és una eina molt resolutiva per aquest tipus de tasques, però al mateix temps poc efectives ja que les dades proteòmiques són caracteritzades per una dimensió molt gran i això fa que necessitin d'un temps molt elevat per obtenir resultats.

6.2 Possibles extencions de la memòria

Un cop finalitzat el projecte i revisant fins on s'ha arribat, sorgeixen algunes extencions que es podrien realitzar a partir del que s'ha fet:

- **Comparar els resultats amb altres tècniques de classificació**: tot i que des del començament es va determinar que la tècnica pricipal del treball seria el *support vector machine* també s'haguès pogut fer servir altres tècniques de classificació supervisada com poden ser les xarxes neuronals, *k-nearest neighbours* o els arbres de decisió.
- **Classificació de multiclasses**: en aquest treball només s'ha pogut fer classificació binària, això significa que les dades només tenien dos possibilitats de classificació, per tant una ampliació podria ser fer una classificació de multiclasses, això vol dir que les dades tindrien més de dos possibilitats de classificació.

6.3 Valoració personal

Un cop acabat el treball, puc dir que realment estic molt contenta dels resultats obtinguts. Tot i que al principi va ser bastant complicat, veure que s'han pogut assolir els objectius proposats es un gran motiu de satisfacció.

El Treball de fi de grau m'ha aportat principalment indepèndencia a l'hora de fer un projecte d'aquestes grans dimensions, ja que durant el grau si que es realitzen projectes però la majoria són amb grup. També m'agradaria destacar la gran quantitat i varietat de coneixements que m'ha permés assolir, com per exemple, la tècnica de l'aprenentatge automàtic per cadenes de caràcters, el mètode de *support vector machine* i nous paquets d'R.

A més a més m'agradaria senyalar que gran part d'aquest projecte s'ha realitzat durant el temps en que he estat d'erasmus, fet que suposa una dificultat afegida, ja que el

contacte amb el director ha estat basicament per correu electrònic.

Per últim m'agradaria agraïr al director d'aquest treball, Esteban Vegas, per la seva ajuda i suport en tot moment, ja que sense ell aquest resultat no hagués estat possible. I a la meva família i amics perquè és en aquests moments quan realment són importants.

Bibliografia

[1] Alegre, S. *Dimensionality reduction for omics data*, Vegas, E., 2014.

[2] Armañanazas, R., Bengoetxea, E., Calvo, B., Inza, I., Larrañaga, P. i Lozano, J. Machie Learning: An Idespensable Tool in Bioinformatics. En Matthiesen, R. (ed). *Bioinformatics Methods in Clinical Research*, 2010, p. 25-104.

[3] Bernardes, J. i Pedreira, C. A Review of Protein Function Prediction Under Machine Learning Perspective. *Recent Patents on Biotechnology*, 2013, vol. 7, núm 2, p. 122-141.

[4] Bodenhofer, U. i Palme, J. *KeBABS — An R Pakage for Kernel Based Analysis of Biological Sequences*. 2014. Disponible a http://www.bioinf.jku.at/software/kebabs/.

[5] Bogdanova, A. i Spirovska, K. *Multiple kernel learning methods ans theis application in yeast protein subcellular localitzation prediction*, 2012.

[6] Brodersen, K., Buhmann, J., Ong, C. i Stephan, K. *The balanced accuracy and its posterior distribution*, 2010.

[7] Crick, F. H. C. i Watson, J. D. Molecular structure of nucleic acids. *Nature*. 1953, vol. 171, núm, 4356, p. 737-738. Disponible a http://www.nature.com/nature/dna50/watsoncrick.pdf.

[8] Elías, J. *Análisis de datos biológico textuales*, 2010.

[9] Hornik, K., Karatzoglou, A. i Smola, A. *Package "kernlab"*. Disponible a http://cran.r-project.org/web/packages/kernlab/kernlab.pdf

[10] *International vocabulary of metrology — Basic and general concepts and associated terms (VIM)* JCGM, 2008.

[11] Lee, JK. Road to statistical bioinformatics. En Lee, JK. (ed). *Statistical Bioinformatics: A Guide for Life and Biomedical Science Researchers*. Wiley-Blackwell, 2010, p. 1-5.

[12] Powers, D. *Evaluation: From Precision, Recall and F-Factor to ROC Informedness, Markedness, Correlation*, 2007.

[13] Riba, L. *Integració de diferents fonts de dades òmiques i visualització de les variaables originals mitjançant tècniques de Machine Learning*, Vegas, E., 2014.

[14] Schölkopf, B., Tsuda, K. i Vert, J. *A primer on kernel methods*, 2004.

[15] Universitat de Barcelona, *Introduction to Molecular Biology*, 2014.

[16] The UniProt Knpwledgebase (UniProtKB). Disponible a http://www.uniprot.org/uniprot/.

[17] Vert, J. *Practical session: String kernels*.

[18] Woon, W. *Core Statistics for Bioinformatics*, 2003.

[19] *Accessexcellence — Codi genètic*. http://www.accessexcellence.org/RC/VL/GG/genetic.php/ , [últim accès el 27/05/2015].

[20] *Accessexcellence — Processament*. http://www.accessexcellence.org/RC/VL/GG/rna_synth.php , [últim accès el 28/06/2015]

[21] *Accessexcellence — Replicació*. http://www.accessexcellence.org/RC/VL/GG/collaboration.php/ , [últim accès el 27/05/2015].

[22] *Accessexcellence — Transcripció i traducció*. http://www.accessexcellence.org/RC/VL/GG/protein_synthesis.php/ , [últim accès el 27/05/2015].

[23] *Discriminative prediction of mammalian enhancers from DNA sequence*. http://www.beerlab.org/p300enhancer/ , [últim accès el 27/05/2015].

[24] Gene ontology. http://geneontology.org/ , [últim accès el 26/06/2015]

[25] *National Centre for Biothecnology Information*. Disponible a http://www.ncbi.nlm.nih.gov/ , [últim accès el 26/06/2015]

[26] *PrOCoil — Data Repository*. http://www.bioinf.jku.at/software/procoil/data.html , [últim accès el 26/06/2015].

[27] *The national health museum*. Disponible a http://www.accessexcellence.org/ , [últim accès el 26/06/2015]

[28] *Viquipèdia — Cross-validation*. https://es.wikipedia.org/wiki/Validaci%C3%B3n_cruzada , [últim accès el 28/06/2015].

[29] *Viquipèdia — Diferències entre ADN i ARN.* https://ca.wikipedia.org/wiki/%C3%80cid_ribonucleic , [últim accés 27/05/2015].

[30] *Viquipèdia — Proteïna.* https://ca.wikipedia.org/wiki/Prote%C3%AFna , [últim accés 27/05/2015].

[31] *Viquipèdia — Sensibilitat i especificitat.* https://en.wikipedia.org/wiki/Sensitivity_and_specificity , [últim accés 16/06/2015].

Annex A

CODI R

A.1 Capítol 3: Kernel per a *strings* amb R

A.1.1 Kernlab

```
> #Kernel entre dues paraules
> library(kernlab)
> sk <- stringdot(type="spectrum", length=2, normalized=F)
> sk('cabra','abracadabra')
> #es el producte escalar entre les dos paraules
> kernelMatrix(sk,c('abracadabra','cabra'))
> #crea la matriu kernel entre les dos paraules

> #kernel entre 4 cadenes d'aminoàcids tipus spectrum i no normalitzat
> (seq <- c("GACGAGGACCGA","AGTAGCGAGGT",
+           "ACGAGGTCTTT","GGACCGAGTCGAGG"))
> sk <- stringdot(type="spectrum", length=2, normalized=F)
> (km.sk <- kernelMatrix(sk, seq))

> #kernel entre 4 cadenes d'aminoàcids tipus spectrum i normalitzat
> skn <- stringdot(type="spectrum", length=2, normalized=T)
> (km.skn <- kernelMatrix(skn, seq))

> #kernel entre 4 cadenes d'aminoàcids tipus boundrange
> skb <- stringdot(type="boundrange", length=2, normalized=T)
> (km.skb <- kernelMatrix(skb, seq))

> #kernel entre 4 cadenes d'aminoàcids tipus constant
> skc <- stringdot(type="constant", length=2, normalized=T)
> (km.skc <- kernelMatrix(skc, seq))
```

```
> #kernel entre 4 cadenes d'aminoàcids tipus exponential
> ske <- stringdot(type="exponential", length=2,lambda = 2, normalized=T)
> (km.ske <- kernelMatrix(ske, seq))
```

A.1.2 Kebabs

```
> #Instalůlació del paquet kebabs
> ### baixar kebabs#####
> source("http://bioconductor.org/biocLite.R")
> biocLite("kebabs")
> library(kebabs)

> #Generació de 4 cadenes d'aminoàcids
> (seq1 <- AAStringSet(c("GACGAGGACCGA",
+                        "AGTAGCGAGGT",
+                        "ACGAGGTCTTT",
+                        "GGACCGAGTCGAGG")))

> #kernel entre 4 cadenes d'aminoàcids tipus spectrum, posició independent
> specK2 <- spectrumKernel(k=2,normalized = T)
> (km.spec <- getKernelMatrix(specK2, seq1))

> #kernel entre 4 cadenes d'aminoàcids tipus mismatch, posició independent
> mismK2M1 <- mismatchKernel(k=2, m=1,normalized = T)
> (km.mis <- getKernelMatrix(mismK2M1, seq1))

> #kernel entre 4 cadenes d'aminoàcids tipus gappy pair, posició independent
> gappyK1M3 <- gappyPairKernel(k=1, m=3,normalized = T)
> (km.gap <- getKernelMatrix(gappyK1M3 , seq1))

> #kernel entre 4 cadenes d'aminoàcids tipus motif, posició independent
> motifCollection1 <- c("A.G","[CT]")
> motif1 <- motifKernel(motifCollection1,normalized = T)
> (km.mot <- getKernelMatrix(motif1 , seq1))

> #kernel entre 4 cadenes d'aminoàcids tipus spectrum, posició específica
> specK2.esp <- spectrumKernel(k=2,normalized = T,distWeight = 1)
> (km.spec.esp <- getKernelMatrix(specK2.esp, seq1))

> #kernel entre 4 cadenes d'aminoàcids tipus spectrum,
> #posició específica amb alineació específica
> specK2.esp2 <- spectrumKernel(k=2,normalized = T,distWeight = 1)
> positionMetadata(seq1) <- c(3, 4, 2, 10) #vector d'iniciadors
> (km.spec.esp2 <- getKernelMatrix(specK2.esp2, seq1))
```

```
> #Gràfic de Predefined Distance Weighting Functions
> curve(linWeight(x, sigma=5), from=-25, to=25, xlab="p - q", ylab="weight",
+ main="Predefined Distance Weighting Functions", col="green")
> curve(expWeight(x, sigma=5), from=-25, to=25, col="blue", add=TRUE)
> curve(gaussWeight(x, sigma=5), from=-25, to=25, col="red", add=TRUE)
> curve(swdWeight(x), from=-25, to=25, col="orange", add=TRUE)
> legend('topright', inset=0.03, title="Weighting Functions",
+        c("linWeight", "expWeight", "gaussWeight", "swdWeight"),
+        fill=c("green", "blue", "red", "orange"))
> text(17, 0.5, "sigma = 5")

> #Gràfic amb les distàncies segons el paràmetre sigma
> par(mfrow=c(1,3))
> curve(linWeight(x, sigma=2), from=-25, to=25, xlab="p - q",
+       ylab="weight", main="linWeight", col="green")
> curve(linWeight(x, sigma=20), from=-25, to=25, col="blue", add=TRUE)
> curve(linWeight(x, sigma=50), from=-25, to=25, col="red", add=TRUE)
> curve(linWeight(x, sigma=100), from=-25, to=25, col="orange", add=TRUE)
> curve(expWeight(x, sigma=2), from=-25, to=25, xlab="p - q",
+       ylab="weight", main="expWeight", col="green")
> curve(expWeight(x, sigma=20), from=-25, to=25, col="blue", add=TRUE)
> curve(expWeight(x, sigma=50), from=-25, to=25, col="red", add=TRUE)
> curve(expWeight(x, sigma=100), from=-25, to=25, col="orange", add=TRUE)
> curve(gaussWeight(x, sigma=2), from=-25, to=25, xlab="p - q",
+       ylab="weight", main="gaussWeight", col="green")
> curve(gaussWeight(x, sigma=20), from=-25, to=25, col="blue", add=TRUE)
> curve(gaussWeight(x, sigma=50), from=-25, to=25, col="red", add=TRUE)
> curve(gaussWeight(x, sigma=100), from=-25, to=25, col="orange", add=TRUE)
> legend('bottomright', inset=0, c("s=2", "s=20", "s=50", "s=100"),
+        fill=c("green", "blue", "red", "orange"))
> par(mfrow=c(1,1))

> #kernel entre 4 cadenes d'aminoàcids tipus spectrum,
> #distància ponderada lineal
> specK2.esp.dist <- spectrumKernel(k=2,normalized = T,
+                                   distWeight = linWeight(sigma = 5))
> (km.spec.esp.dist <- getKernelMatrix(specK2.esp.dist, seq1))

> #kernel entre 4 cadenes d'aminoàcids tipus spectrum,
> #distància ponderada exponencial
```

```
> specK2.esp.dist2 <- spectrumKernel(k=2,normalized = T,
+                                    distWeight = expWeight(sigma = 5))
> (km.spec.esp.dist2 <- getKernelMatrix(specK2.esp.dist2, seq1))

> #kernel entre 4 cadenes d'aminoàcids tipus spectrum,
> #distància ponderada gaussiana
> specK2.esp.dist3 <- spectrumKernel(k=2,normalized = T,
+                                    distWeight = gaussWeight(sigma = 5))
> (km.spec.esp.dist3 <- getKernelMatrix(specK2.esp.dist3, seq1))

> #kernel entre 4 cadenes d'aminoàcids tipus spectrum,
> #distància ponderada the shifted degree weighted
> specK2.esp.dist4 <- spectrumKernel(k=2,normalized = T,
+                                    distWeight = swdWeight())
> (km.spec.esp.dist4 <- getKernelMatrix(specK2.esp.dist4, seq1))
```

A.2 Capítol 4: *support vector machine* amb R

A.2.1 Kernlab

```
> #Aplicació del support vector machine per les dades reuters
> #Descàrrega de les dades
> data(reuters)

> #crear el conjunt d'entrenament i el conjunt test
> numSamples <- length(reuters)
> trainingFraction <- 0.7
> train <- sample(1:numSamples, trainingFraction * numSamples)
> test <- c(1:numSamples)[-train]

> #específicar la funció kernel
> sk <- stringdot(type="spectrum", length=2, normalized=TRUE)

> #support vector machine
> svp <- ksvm(x=reuters[train],y=rlabels[train],type="C-svc",C=1,kernel=sk)
> svp

> #Fer la predicció
> ypred <- predict(object=svp,newdata=reuters[test])
> print(table(ypred,rlabels[test]))
```

A.2.2 Kebabs

```
> #Aplicació del support vector machine per les dades TFBS
> data(TFBS)#Descàrrega de les dades
> enhancerFB

> #crear el conjunt d'entrenament i el conjunt test
> train <- sample(1:length(enhancerFB), length(enhancerFB) * 0.7)
> test <- c(1:length(enhancerFB))[-train]

> #específicar la funció kernel
> specK2 <- spectrumKernel(k=2,normalized=T)

> #support vector machine
> model <- kbsvm(x=enhancerFB[train], y=yFB[train], kernel=specK2,
+                pkg="kernlab", svm="C-svc", C=10, explicit="no")
> model

> #predicció i evaluació de la predicció
> pred <- predict(model, enhancerFB[test])
> evaluatePrediction(pred, yFB[test], allLabels=unique(yFB))

> #Cross-validation k-fold cross-validation
> #Descàrrega de CCoil data
> data(CCoil)
> x <- ccseq
> y <- as.numeric(yCC)

> #específicar la funció kernel
> gappyK1M6 <- gappyPairKernel(k=1, m=6)

> #support vector machine amb el paràmetre cross
> model <- kbsvm(x=x, y=y, kernel=gappyK1M6,
+                pkg="kernlab", svm="C-svc", cost=30, cross=3)

> #Resultats
> cvResult(model)

> #Boxplot amb els errors corresponents a la cross-validation
> plot(cvResult(model))

> #Cross-validation Grouped cross-validation
> #Grups
> head(ccgroups)
```

```
> #support vector machine amb el paràmetre cross i groupBy
> model2 <- kbsvm(x=x, y=y, kernel=gappyK1M6,
+                 pkg="kernlab", svm="C-svc", cost=30, cross=3, groupBy=ccgroups)

> #Resultats
> cvResult(model2)

> #Boxplot amb els errors corresponents a la cross-validation
> plot(cvResult(model2))

> #Grid search
> #Descàrrega de CCoil data
> data(CCoil)
> x <- ccseq
> y <- as.numeric(yCC)

> #específicar les funcions kernel i els costos corresponents
> specK24 <- spectrumKernel(k=2:4)
> gappyK1M24 <- gappyPairKernel(k=1, m=2:4)
> gridKernels <- c(specK24, gappyK1M24)
> cost <- c(1,10, 100, 1000, 10000)

> #support vector machine amb el paràmetre cross
> model <- kbsvm(x=x, y=y, kernel=gridKernels,
+                 pkg="kernlab", svm="C-svc", cost=cost, cross=3)

> #Resultats
> modelSelResult(model)

> #Selůlecció del model
> #Descàrrega de CCoil data
> data(CCoil)
> x <- ccseq
> y <- as.numeric(yCC)

> #específicar les funcions kernel i els costos corresponents
> specK24 <- spectrumKernel(k=2:4)
> gappyK1M24 <- gappyPairKernel(k=1, m=2:4)
> gridKernels <- c(specK24, gappyK1M24)
> cost <- c(10, 50, 100)
```

```
> #support vector machine amb els paràmetres corss i nestedCross
> model <- kbsvm(x=x, y=y, kernel=gridKernels,
+                pkg="kernlab", svm="C-svc", cost=cost, cross=3, nestedCross=4)

> #Resultats
> modelSelResult(model)

> #Millor model
> c(selGridRow(modelSelResult(model))[[1]],
+   selGridCol(modelSelResult(model))[1,])
```

A.3 Capítol 5: Aplicació del *support vector machine*

A.3.1 Kernlab

```
> #Es necessita la llibreria seqinr per poder llegir els fitxers en format fasta
> library(seqinr)
> #Descàrrega de les dades
> protdata1 <- read.fasta("C:/Users/Aurea/Desktop/Subcel.location Uniprot_fasta/
+                         chloroplast.fasta",  seqtype="AA", as.string=TRUE)
> protdata2 <- read.fasta("C:/Users/Aurea/Desktop/Subcel.location Uniprot_fasta/
+                         Nucleus.fasta",  seqtype="AA", as.string=T)

> #Creació del vector de cadenes x i el vector d'etiquetes y
> x1 <- unlist(getSequence(protdata1,as.string=TRUE),recursive=FALSE)
> y1 <- rep(1,length(x1))
> x2 <- unlist(getSequence(protdata2,as.string=TRUE),recursive=FALSE)
> y2 <- rep(2,length(x2))
> x <- c(c(x1,x2))
> y <- c(c(y1,y2))

> #Agafar 100 cadenes a l'atzar
> data <- sample(1:length(x),100)
> x <- x[data]
> y <- y[data]

> #Funció que et calcula l'error de predicció
> #per diferents tipus de kernel i diferents costos
> kernlab <- function(x,y,ty,ks,ls,prop,clist){
+   numSamples <- length(x)
+   trainingFraction <- prop
```

```
+   train <- sample(1:numSamples, trainingFraction * numSamples)
+   test <- c(1:numSamples)[-train]
+   if(ls[1]==0){leg <- rep('A',length(ty)*length(ks))}
+   else{leg <- rep('A',(length(ty)-1)*length(ks)+length(ls)*length(ks))}
+   e <- 1
+   err <- matrix(0,length(leg),length(clist))
+   for (it in 1:length(ty)){
+     t <- ty[it]
+     #cat('type=',t)
+     #cat('\n')
+     for (ik in 1:length(ks)) {
+       k <- ks[ik]
+       #cat('k=',k)
+       if(t=="exponential"){
+         for(il in 1:length(ls)){
+           l <- ls[il]
+           #cat('l=',l)
+           sk <- stringdot(type=t, length=k, lambda = l,normalized=TRUE)
+           for (ic in 1:length(clist)) {
+             #cat('.')
+
+             # Train a SVM on training set
+             svp <- ksvm(x[train],y[train],type="C-svc",C=clist[ic],kernel=sk)
+
+             # Predict on test set
+             ypred <- predict(object=svp,newdata=x[test])
+
+             # Store accuracy
+             err[e,ic] <- sum(ypred != y[test])
+
+           }
+           leg[e] <- paste(t,":","l=",l,"k=",k)
+           e <- e+1
+           #cat('\n')
+         }
+
+       }
+       else{sk <- stringdot(type=t, length=k, normalized=TRUE)
+
```

```
+       for (ic in 1:length(clist)) {
+         #cat('.')
+
+         # Train a SVM on training set
+         svp <- ksvm(x[train],y[train],type="C-svc",C=clist[ic],kernel=sk)
+
+         # Predict on test set
+         ypred <- predict(object=svp,newdata=x[test])
+
+         # Store accuracy
+         err[e,ic] <- sum(ypred != y[test])
+
+       }
+       leg[e] <- paste(t,":","k=",k)
+       e <- e+1
+       #cat('\n')
+     }
+     }
+   }
+     err <- err/length(test)
+     plot(c(min(clist),max(clist)),c(0,1),log="x",type='n',
+          xlab="C",ylab="Error",main="Kernlab")
+     for (ik in 1:length(leg)) {
+       lines(clist,err[ik,],col=ik,lwd=2)
+     }
+     legend("topright",paste(leg),col=1:length(leg),lwd=rep(2,length(ks)))
+     grid(col='darkgray')
+ }

> #Aplicació de la funció
> kernlab(x,y,c("spectrum","constant","boundrange"),c(2,3),c(0),0.7,3^seq(-1,4))
> kernlab(x,y,c("exponential"),c(2,3,4),c(2,3),0.7,3^seq(-1,4))
```

A.3.2 Kebabs

```
> #Descàrrega de les dades
> protdata1 <- readAAStringSet("C:/Users/Aurea/Desktop/
+                              Subcel.location Uniprot_fasta/chloroplast.fasta")
> protdata2 <- readAAStringSet("C:/Users/Aurea/Desktop/
+                              Subcel.location Uniprot_fasta/Nucleus.fasta")
```

```
> #Creació del vector de cadenes x i el vector d'etiquetes y
> y1 <- rep(1,length(protdata1))
> y2 <- rep(2,length(protdata2))
> x <- c(c(protdata1,protdata2))
> y <- c(c(y1,y2))
> x <- x[data] #es fa servir la mateixa mostra que abans per poder comparar
> y <- y[data]

> #Funció que et calcula l'error de predicció
> #pel kernel de tipus spectrum i diferents costos
> keb.spec <- function(x,y,ks,pos,clist){
+   numSamples <- length(x)
+   trainingFraction <- 0.7
+   train <- sample(1:numSamples, trainingFraction * numSamples)
+   test <- c(1:numSamples)[-train]
+   if((sum(pos)==0 & length(pos)==1) | (sum(pos)==1 & length(pos)==1))
+   {leg <- rep('A',length(ks))}
+   else{leg <- rep('A',length(ks)*2)}
+   e <- 1
+   err <- matrix(0,length(leg),length(clist))
+   for(ie in 1:length(pos)){
+     if(pos[ie]==0){
+       #cat('no esp')
+       for (ik in 1:length(ks)) {
+         k <- ks[ik]
+         #cat('k=',k)
+
+         # kernel definition
+         sk <- spectrumKernel(k = k, normalized=TRUE)
+
+         # Loop over C
+         for (ic in 1:length(clist)) {
+           #cat('.')
+
+           # Train a SVM on training set
+           svp <- kbsvm(x=x[train],y=y[train],svm="C-svc",
+                        C=clist[ic],pkg="kernlab",kernel=sk)
+
+           # Predict on test set
```

```
+           ypred <- predict(svp,x[test])
+
+           # Store accuracy
+           err[e,ic] <- sum(ypred != y[test])
+         }
+         leg[e] <- paste("no esp","k=",k)
+         e <- e+1
+         #cat('\n')
+       }
+     }
+     else{
+        # cat('esp')
+         for (ik in 1:length(ks)) {
+           k <- ks[ik]
+           #cat('k=',k)
+
+           # kernel definition
+           sk <- spectrumKernel(k = k, normalized=TRUE,distWeight=1)
+
+           # Loop over C
+           for (ic in 1:length(clist)) {
+             #cat('.')
+
+             # Train a SVM on training set
+             svp <- kbsvm(x=x[train],y=y[train],svm="C-svc",C=clist[ic]
+                          ,pkg="kernlab",kernel=sk)
+
+             # Predict on test set
+             ypred <- predict(svp,x[test])
+
+             # Store accuracy
+             err[e,ic] <- sum(ypred != y[test])
+           }
+           leg[e] <- paste("esp","k=",k)
+           e <- e+1
+           #cat('\n')
+         }
+       }
+   }
```

```
+   err <- err/length(test)
+   plot(c(min(clist),max(clist)),c(0,1),log="x",type='n',
+        xlab="C",ylab="Error",main="Spectrum kernel")
+   for (ik in 1:length(leg)) {
+     lines(clist,err[ik,],col=ik,lwd=2)
+   }
+   legend("topright",paste(leg),col=1:length(leg),lwd=rep(2,length(ks)))
+   grid(col='darkgray')
+ }

> #Aplicació de la funció
> keb.spec(x,y,c(2,3,4),c(0,1),3^seq(-1,4))

> #Funció que et calcula l'error de predicció
> #pel kernel de tipus gappy pair i diferents costos
> keb.gappy <- function(x,y,ks,ms,pos,prop,clist){
+   numSamples <- length(x)
+   trainingFraction <- prop
+   train <- sample(1:numSamples, trainingFraction * numSamples)
+   test <- c(1:numSamples)[-train]
+   if((sum(pos)==0 & length(pos)==1) | (sum(pos)==1 & length(pos)==1))
+   {leg <- rep('A',length(ks)*length(ms))}
+   else{leg <- rep('A',length(ks)*length(ms)*2)}
+   e <- 1
+   err <- matrix(0,length(leg),length(clist))
+   for(ie in 1:length(pos)){
+     if(pos[ie]==0){
+       #cat('no esp')
+       for(im in 1:length(ms)){
+         m <- ms[im]
+         #cat('m=',m)
+       for (ik in 1:length(ks)) {
+         k <- ks[ik]
+         #cat('k=',k)
+
+         # kernel definition
+         sk <- gappyPairKernel(k=k,m=m,normalized=T)
+
+         # Loop over C
+         for (ic in 1:length(clist)) {
```

```
+           #cat('.')
+
+           # Train a SVM on training set
+           svp <- kbsvm(x=x[train],y=y[train],svm="C-svc",
+                        C=clist[ic],pkg="kernlab",kernel=sk)
+
+           # Predict on test set
+           ypred <- predict(svp,x[test])
+
+           # Store accuracy
+           err[e,ic] <- sum(ypred != y[test])
+         }
+         leg[e] <- paste("no esp","k=",k,'m=',m)
+         e <- e+1
+         #cat('\n')
+       }
+       }
+     }
+     else{
+       #cat('esp')
+       for(im in 1:length(ms)){
+         m <- ms[im]
+         #cat('m=',m)
+       for (ik in 1:length(ks)) {
+         k <- ks[ik]
+         #cat('k=',k)
+
+         # kernel definition
+         sk <- gappyPairKernel(k=k,m=m,normalized=T,distWeight=1)
+
+         # Loop over C
+         for (ic in 1:length(clist)) {
+           #cat('.')
+
+           # Train a SVM on training set
+           svp <- kbsvm(x=x[train],y=y[train],svm="C-svc",
+                        C=clist[ic],pkg="kernlab",kernel=sk)
+
+           # Predict on test set
```

```
+           ypred <- predict(svp,x[test])
+
+           # Store accuracy
+           err[e,ic] <- sum(ypred != y[test])
+         }
+         leg[e] <- paste("esp","k=",k,'m=',m)
+         e <- e+1
+         #cat('\n')
+       }
+       }
+     }
+   }
+   err <- err/length(test)
+   plot(c(min(clist),max(clist)),c(0,1),log="x",type='n',
+        xlab="C",ylab="Error",main="Gappy pair kernel")
+   for (ik in 1:length(leg)) {
+     lines(clist,err[ik,],col=ik,lwd=2)
+   }
+   legend("topright",paste(leg),col=1:length(leg),
+          lwd=rep(2,length(ks)))
+   grid(col='darkgray')
+ }

> #Aplicació de la funció
> keb.gappy(x,y,c(1),c(1,2),c(0,1),0.7,3^seq(-1,4))

> #Funció que et calcula l'error de predicció
> #pel kernel de tipus mismatch i diferents costos
> keb.mismatch <- function(x,y,ks,ms,prop,clist){
+     numSamples <- length(x)
+     trainingFraction <- prop
+     train <- sample(1:numSamples, trainingFraction * numSamples)
+     test <- c(1:numSamples)[-train]
+     r <- 0
+     for (i in 1:length(ms)) {
+       for(j in 1:length(ks)){
+         if( ms[i]<ks[j]){
+           r <- r+1
+         }
+       }
```

```
+     }
+     leg <- rep('A',r)
+     e <- 1
+     err <- matrix(0,length(leg),length(clist))
+     #cat('no esp')
+     for(im in 1:length(ms)){
+       m <- ms[im]
+       #cat('m=',m)
+       for (ik in 1:length(ks)) {
+         if(ms[im]<ks[ik]){
+           k <- ks[ik]
+           #cat('k=',k)
+
+           # kernel definition
+           sk <- mismatchKernel(k=k,m=m,normalized=T)
+
+           # Loop over C
+           for (ic in 1:length(clist)) {
+             #cat('.')
+
+             # Train a SVM on training set
+             svp <- kbsvm(x=x[train],y=y[train],svm="C-svc",
+                          C=clist[ic],pkg="kernlab",kernel=sk)
+
+             # Predict on test set
+             ypred <- predict(svp,x[test])
+
+             # Store accuracy
+             err[e,ic] <- sum(ypred != y[test])
+           }
+           leg[e] <- paste("k=",k,'m=',m)
+           e <- e+1
+           #cat('\n')
+         }
+       }
+     }
+     err <- err/length(test)
+     plot(c(min(clist),max(clist)),c(0,1),log="x",type='n',
+          xlab="C",ylab="Error",main="Mismatch kernel")
```

```
+     for (ik in 1:length(leg)) {
+       lines(clist,err[ik,],col=ik,lwd=2)
+     }
+     legend("topright",paste(leg),col=1:length(leg),lwd=rep(2,length(ks)))
+     grid(col='darkgray')
+   }

> #Aplicació de la funció
> keb.mismatch(x,y,c(2,3),c(1),0.7,3^seq(-1,4))

> #Grid search
> #Especificació de les funcions kernel
> specK25i <- spectrumKernel(k=2:5)
> specK25d <- spectrumKernel(k=2:5, distWeight=1)
> gappyK1M12i <- gappyPairKernel(k=1, m=1:2)
> gappyK1M12d <- gappyPairKernel(k=1, m=1:2, distWeight=1)
> mismK2M1 <- mismatchKernel(k=2, m=1)
> mismK3M1 <- mismatchKernel(k=3, m=1)
> gridKernels <- c(specK25i,specK25d ,gappyK1M12i,
+                  gappyK1M12d ,mismK2M1,mismK3M1)

> #Especificació dels costos
> cost <- 3^seq(-1,4)

> #Support vector machine
> model1 <- kbsvm(x=x, y=y, kernel=gridKernels,
+                 pkg="kernlab", svm="C-svc", cost=cost, cross=10)

> #Resultats
> modelSelResult(model1)

> #Selůlecció del model
> #Especificació de les funcions kernel
> specK25i <- spectrumKernel(k=2:5)
> specK25d <- spectrumKernel(k=2:5, distWeight=1)
> gappyK1M12i <- gappyPairKernel(k=1, m=1:2)
> gappyK1M12d <- gappyPairKernel(k=1, m=1:2, distWeight=1)
> mismK2M1 <- mismatchKernel(k=2, m=1)
> mismK3M1 <- mismatchKernel(k=3, m=1)
> gridKernels <- c(specK25i,specK25d ,gappyK1M12i,
+                  gappyK1M12d ,mismK2M1,mismK3M1)
```

```
> #Especificació dels costos
> cost <- 3^seq(-1,4)

> #Support vector machine
> model2 <- kbsvm(x=x, y=y, kernel=gridKernels,
+                 pkg="kernlab", svm="C-svc", cost=cost, cross=10, nestedCross=4)

> #Resultats
> modelSelResult(model2)

> #Millor model
> (r <- c(selGridRow(modelSelResult(model2))[[1]],
+         selGridCol(modelSelResult(model2))[1,2]))

> #Predicció i evaluació de la predicció
> #Descàrrega de les dades
> library(seqinr)
> # Load localization data from http://www.psort.org/dataset/dataset1_0.txt
> protdata1 <- readAAStringSet("C:/Users/Aurea/Desktop/
+                               Subcel.location Uniprot_fasta/chloroplast.fasta")
> protdata2 <- readAAStringSet("C:/Users/Aurea/Desktop/
+                               Subcel.location Uniprot_fasta/Nucleus.fasta")

> #Creació del vector de cadenes x i el vector d'etiquetes y
> y1 <- rep(1,length(protdata1))
> y2 <- rep(2,length(protdata2))
> x <- c(c(protdata1,protdata2))
> y <- c(c(y1,y2))

> #s'agafen 5000 cadenes a l'atzar amb les etiquetes corresponents
> data <- sample(1:length(x),5000)
> x <- x[data]
> y <- y[data]

> #Implementació de la base d'entrenament i test
> trainingFraction <- 0.7
> train <- sample(1:length(x), trainingFraction * length(x))
> test <- c(1:length(x))[-train]

> #Support vector machine
> model3 <- kbsvm(x=x[train], y=y[train], kernel=r[[1]],
+                  pkg="kernlab", svm="C-svc", cost=r[[2]])
```

```
> #Predicció
> pred <- predict(model3, x[test])

> #Evaluació de la predicció
> evaluatePrediction(pred, y[test],allLabels=unique(y))
```