10. Jezgrene metode

Strojno učenje 1, UNIZG FER, ak. god. 2022./2023.

Jan Šnajder, predavanja, v1.9

Prošla dva puta bavili smo se strojem potpornih vektora. Objasnili smo model tog algoritma i optimizacijski postupak koji se temelji na ideji maksimalne margine. Na zadnjem predavanju to smo malo proširili i govorili smo o mekoj margini, koja dopušta malo pogrešnu klasifikaciju i efektivno time sprječava prenaučenost.

Danas ćemo pogledati nešto što je povezano s SVM-om, ali je i šire od toga: **jezgrene metode** (engl. *kernel methods*). Jezgrene metode općenit su i vrlo moćan radni okvir za modeliranje nelinearnosti. Vidjet ćemo da kezgrene metode dolaze u dva osnovna okusa: **jezgreni strojevi** i **jezgreni trik**. 1

1 Jezgrene funkcije

Do sada smo uvijek pretpostavljali da je primjer $\mathbf{x}$ **vektor** značajki, odnosno da primjeri žive u nekom n-dimenzijskome vektorskom prostoru, $\mathcal{X} = \mathbb{R}^n$. No, ponekad nije jasno kako primjer možemo prikazati kao vektor značajki. Npr., recimo da želimo napraviti klasifikator koji će klasificirati riječi (npr., za neku riječ hrvatskoga jezika želimo odrediti je li ta riječ latinizam). U tom slučaju naši primjeri $\mathbf{x}$ su riječi, tj. kao nizova znakova (stringovi). Nije očito kako bismo niz znakova prikazali kao vektor u nekom n-dimenzijskom prostoru. Ili, recimo da radimo klasifikaciju DNK nizova. Kako biste u tom slučaju prikazali značajke? Što ako radimo klasifikaciju grafova, np. klasificiramo strukture molekula koje su prikazane kao graf? Nije naš jasno kako bismo grafove prikazali kao vektor značajki. U takvim je slučajevima možda lakše računati **sličnost** između primjera, nego ih vektorizirati. Npr., lakše mi je izračunati sličnost između dviju riječi, dva DNK niza, ili čak dva grafa, nego svaki od njih prebaciti u prostor značajki. U to slučaju, umjesto vektorskog prostora $\mathcal{X} = \mathbb{R}^n$, imati ćemo neki **apstraktan prostor** $\mathcal{X}$. Taj prostor nije nužno vektorski, tj. možda ga ne znamo ga definirati pomoću značajki, ali znamo kako izračunati sličnost između primjera $\mathbf{x} \in \mathcal{X}$. 2

Općenito, sličnost dvaju primjera računamo pomoću jezgrene funkcije. **Jezgrena funkcija** (engl. *kernel function*) (ili samo **kernel** – koristit ćemo i taj naziv) je realna funkcija koja kao argumenate uzima dva primjera iz $\mathcal{X}$ i daje nam sličnost između tih primjera:

$$\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$$

Tipično vrijedi:

$$\kappa(\mathbf{x}, \mathbf{x}') \geqslant 0$$

i

$$\kappa(\mathbf{x}, \mathbf{x}') = \kappa(\mathbf{x}', \mathbf{x})$$

tj. jezgrena funkcija je nenegativna i simetrična. Ako je domena jezgrene funkcije dodatno ograničena na $[0, 1]$, dakle $\kappa : \mathcal{X} \times \mathcal{X} \to [0, 1]$ i ako $\kappa(\mathbf{x}, \mathbf{x}) = 1$, onda govorimo o **mjeri sličnosti** (engl. *similarity measure*). 3

Kao što smo već napomenuli, jezgrene funkcije imaju itekako smisla kada su podatci s kojima radimo na neki način strukturirani. Ako se radi o nizovima znakova, npr., riječima ili nizovima

DNK, onda koristimo **jezgre nizova znakova** ili (engl. *string kernele*), kojima uspoređujemo sličnost dvaju znakovnih nizova. Ako je struktura podataka složenija od nizova, onda idemo na **jezgre stabala** (engl. *tree kernels*) ili **jezgre grafova** (engl. *graph kernels*). Prve se često koriste u području obrade prirodnog jezika, gdje se uspoređuju sintaktički grafovi rečenica. Jezgre grafova još su općenitije: sličnost između grafova tipično računaju na temelju broja zajedničkih podgrafova ili broja zajedničkih šetnji u oba grafa, a mogu se definirati i za težinske ili usmjerene grafove. 4

▸ **PRIMJER**

Želimo raditi klasifikaciju riječi (koje se sastoje od slova) ili klasifikaciju nizova DNK (koja se sastoji od nukleotida A, T, G, C). Nije baš jasno kako bismo takve nizove najbolje priazali kao vektore. Doduše, možemo smisliti razne načine, ali lakše je definirati sličnost dvaju znakovnih nizova, odnosno definirati **jezgru znakovnih nizova**. Nju možemo definirati na različite načine, npr., možemo brojati koliko nizovi imaju podudarnih podnizova, ili možemo koristiti **Levenshteinovu udaljenost** (broj operacija umetanja i brisanja potrebih da bi se jedan niz sveo na drugi). Npr., *zagreb* i *zagrebački* imaju zajedničke podnizove *zagreb*, *zagre*, *zagr*, *zag*, *za*, *z*, *agreb*, *agre*, itd. Takvu jezgru možemo definirati ovako:

$$\kappa(\mathbf{x}, \mathbf{x}') = \sum_{s \in \mathcal{A}^*} c(s, x) c(s, x')$$

gdje je $\mathcal{A}^*$ skup svih nizova znakova abecede $\mathcal{A}$ (* je Kleeneov operator), a c brojač koliko puta se podniz s pojavio u nizu x.

Dakle, jedna jasna motivacija za korištenje jezgrenih funkcija je situacija kada primjeri nisu vekori i imaju neku internu strukturu. Međutim, jezgrene funkcije možemo koristiti i onda kada primjeri jesu vektori (bilo da su vektori u ulaznom prostoru ili da su vektori nakon preslikavanja u prostor značajki). Dapače, to se vrlo često radi, budući da jezgrene funkcije imaju i druge prednosti, kao što ćemo vidjeti.

Pogledajmo sada nekoliko primjera jezgrenih funkcija koji rade s primjerima koji se mogu prikazati kao vektori. Najjednostavnija je **linearna jezgra**, koja je jednostavno skalarni umnožak dvaju vektora primjera:

$$\kappa(\mathbf{x}, \mathbf{x}') = \mathbf{x}^{\mathrm{T}} \mathbf{x}'$$

Ova jezgra je simetrična, ali nije nenegativna niti je odozgo ograničena na 1. (Zapravo, mi smo na ovaj način već računali sličnost primjera kod SVM-a. To će reći da SVM zapravo koristi jezgru. Tako je, i na to ćemo se vratiti.)

Jedna moćna alternativa linearnoj jezgri je **Gaussova jezgra**:

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\big(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\big)$$

Ovo je isto kao gustoća vjerojatnosti normalne (Gaussove) distribucije centrirane u $\mathbf{x}$, samo što nema normalizacijskog izraza $1/\sqrt{2\pi\sigma^2}$ (kojim se osigurava da integral gustoće vjerojatnosti bude jednak jedinice). Prisjetimo se, $\|\mathbf{x} - \mathbf{x}'\|^2$ je kvadrat euklidske udaljenosti između $\mathbf{x}$ i $\mathbf{x}'$:

$$\|\mathbf{x} - \mathbf{x}'\|^2 = (\mathbf{x} - \mathbf{x}')^{\mathrm{T}}(\mathbf{x} - \mathbf{x}') = \sum_{j=1}^{n} (x_j - x'_j)^2$$

Ovo se lijepo može interpetirati upravo kao **sličnost** između $\mathbf{x}$ i $\mathbf{x}'$ jer će imati vrijednost 1 za $\mathbf{x} = \mathbf{x}'$, a vrijednost nula kako udaljenost između ta dva vektora ide prema beskonačno. Parametar σ^2 je **varijanca** normalne distribucije: što je varijanca veća, to je Gaussovo "zvono" šire. U ovom kontekstu σ^2 se također zove **širina pojasa** (engl. *bandwidth*): što je σ^2 veća, to će biti veća sličnost između primjera koji su udaljeniji. S druge strane, to što je σ^2 manja, to smo striktniji u pogledu što je slično a što nije. Nekad koristimo $\gamma = 1/2\sigma^2$, pa pišemo:

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\big(-\gamma\|\mathbf{x} - \mathbf{x}'\|^2\big)$$

gdje parametar $\gamma = 1/2\sigma^2$ nazivamo **preciznost** (inverz varijance). Zapravo, $1/\sigma^2$ zovemo preciznost, ali nema veza, razlika je u konstantnom faktoru. Koji je efekt parametra γ odnosno σ^2 na izračun sličnosti između primjera? Što je preciznost veća (odnosno σ^2 manja), to je distribucija uža i to bliže moraju biti primjeri da bismo ih smatrali sličnima. Suprotno, što je preciznost manja (odnosno σ^2 veća), to je distribucija šira i to su nam primjeri općenito međusobno sličniji.

Jezgre ovog tipa, koje su neka funkcija udaljenosti između primjera, $\|\mathbf{x} - \mathbf{x}'\|$, zovemo **radijalne bazne funkcije** (engl. *radial basis functions, RBF*). Dakle, jezgra RBF općenito je definirana kao:

$$\kappa(\mathbf{x}, \mathbf{x}') = f(\|\mathbf{x} - \mathbf{x}'\|)$$

gdje je f neka funkcija, npr., eksponencijalna funkcija kao Kod Gaussove jezgre. Gaussovoj jezgri slična je **eksponencijalna jezgra**, koja također koristi eksponencijalnu funkciju, ali ne kvadrira normu, tj.:

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\big(-\gamma\|\mathbf{x} - \mathbf{x}'\|\big)$$

Još jedna često korištena jezgra RBF je **inverzna kvadratna jezgra**:

$$\kappa(\mathbf{x}, \mathbf{x}') = \frac{1}{1 + \|\mathbf{x} - \mathbf{x}'\|^2}$$

Osim što jezgra RBF ne mora nužno biti Gaussova jezgra, ona ne mora koristiti ni euklidsku udaljenost. Problem s euklidskom udaljenošću je što sve značajke (odnosno sve dimenzije) tretira jednako važnima. Sjetimo se primjera s godinama i prihodima: ne želimo da prihodi dominiraju u izračunu udaljenosti samo zato što imaju veću mjernu skalu. Jedan način da to riješimo, koji smo spomenuli zadnji put, jest da normaliziramo značajke, pa tek onda računamo euklidsku udaljenost. Međutim, odnose između značajki možemo još bolje modelirati ako koristimo poopćenje euklidske udaljenosti, tzv. **Mahalanobisovu udaljenost**:

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\Big(-\frac{1}{2}(\mathbf{x} - \mathbf{x}')^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x} - \mathbf{x}')\Big)$$

gdje je $\boldsymbol{\Sigma}$ **kovarijacijska matrica**, koja kodira kovarijaciju između značajki (dimenzija). Kovarijanca nam kazuje koliko varijable zajedno variraju, odnosno koliko zajednički rastu ili padaju. Npr., može se pokazati da prihod pozitivno korelira s dobi više nego neki drugi par značajki. Onda želimo da, ako dvije osobe imaju slične prihode i sličnu dob, da to u izračunu sličnosti ima manju težinu nego ako imaju nešto drugo slično. To upravo možemo postići Mahalanobisovom udaljenošću. Ako kovarijacijsku matricu postavimo na jediničnu matricu, $\boldsymbol{\Sigma} = \mathbf{I}$, što znači da nema kovaricija između značajki i da je varijanca svake značajke jednaka, onda Mahalanobisova udaljenost degenerira na euklidsku udaljenost. 5

Izvrsno, sada znamo što su to jezgrene funkcije! Iduće pitanje je kako ih iskoristiti za klasifikaciju i regresiju. Tu imamo dvije mogućnosti: prva mogućnost su **jezgreni strojevi**, a druga mogućnost su **kernelizacija**, odnosno primjena tzv. **jezgrenog trika**. Pogledat ćemo obje mogućnosti. Krenimo najprije s jezgrenim strojevima.

2 Jezgreni strojevi

Jezgreni strojevi imaju mističan naziv, ali zapravo nisu ništa drugo nego jedna vrsta poopćenih linearnih modela. Pa, prisjetimo se najprije što je to poopćeni linearni model, pa ćemo polako doći do jezgrenih strojeva. **Poopćeni linearni model** definirali smo ovako:

$$h(\mathbf{x}) = f(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}))$$

gdje je $\boldsymbol{\phi}$ funkcija koja je preslikavala iz ulaznog prostora u prostor značajki:

$$\boldsymbol{\phi}(\mathbf{x}) = \big(1, \phi_1(\mathbf{x}), \ldots, \phi_m(\mathbf{x})\big)$$

gdje je

$$\{\phi_1, \phi_2, \ldots, \phi_m\}$$

skup od m **baznih funkcija** (nelinearnih funkcija ulaznih varijabli), svaka od kojih uzima cijeli primjer i vraća jedan broj, $\phi_j : \mathbb{R}^n \to \mathbb{R}$.

Problem s baznim funkcijama očito je taj da ih moramo nekako definirati, a da ne znamo unaprijed koje su dobre funkcije za naš problem, tj. koje nam bazne funkcije daju linearnu odvojivost u prostoru značajki. Taj smo problem već bili dotakli kada smo pričali o logističkoj regresiji. Tada smo opisali ideju **adaptivnih baznih funkcija**, tj. ideju da bazne funkcije automatski prilagodimo našim podatcima, umjesto da ih unaprijed ručno definiramo. Također smo spomenuli da to možemo napraviti na dva načina. Jedan način je da bazne funkcije parametriziramo te da te parametre učimo iz podataka, što nas je spektakularno dovelo do neuronskih mreža. Drugi je način da u prostoru primjera definiramo neke referentne primjere te da onda bazne funkcije definiramo tako da mjere sličnost između ulaznog primjera i dotičnih referentnih primjera. Drugim riječima, ideja je da za bazne funkcije koristimo **jezgrene funkcije**.

Danas se bavimo ovim drugim načinom. Dakle, jezgrene funkcije ϕ_j definirat ćemo tako da uspoređuju ulazni primjer s nekim unaprijed odabranim referentnim primjerima u ulaznom prostoru. Označimo te odabrane primjere sa $\boldsymbol{\mu}^j \in \mathcal{X}$, i neka ih ima ukupno m. Dakle, svaka bazna funkcija ϕ_j bit će definirana kao jezgrena funkcija koja mjeri sličnost u odnosu na primjer $\boldsymbol{\mu}^j$, tj. $\phi_j(\mathbf{x}) = \kappa(\mathbf{x}, \boldsymbol{\mu}_j)$.

S tako definiranim baznim funkcijama, imamo ovakvu funkciju preslikavanja:

$$\boldsymbol{\phi}(\mathbf{x}) = \big(1, \kappa(\mathbf{x}, \boldsymbol{\mu}^1), \kappa(\mathbf{x}, \boldsymbol{\mu}^2), \ldots, \kappa(\mathbf{x}, \boldsymbol{\mu}^m)\big)$$

I to je to! Poopćeni linearni model gdje vektor značajki ima upravo ovakav oblik nazivamo **jezgreni stroj** (engl. *kernel machine*).

Sad možemo jednostavno raditi *business as usual*, koristeći tako definirano preslikavanje, npr., u logističkoj regresiji:

$$h(\mathbf{x}; \mathbf{w}) = \sigma(\mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}))$$

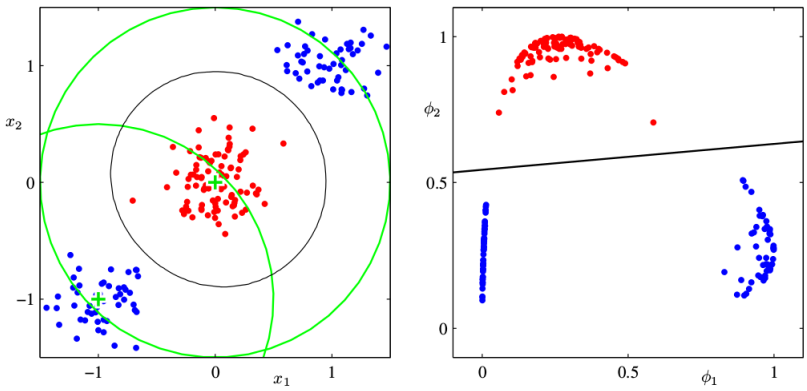
ili linearnoj regresiji:

$$h(\mathbf{x}; \mathbf{w}) = \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x})$$

ili SVM-u, koji ima isti model kao i linearna regresija. Pogledajmo nekoliko primjera.

▸ **PRIMJER**

6



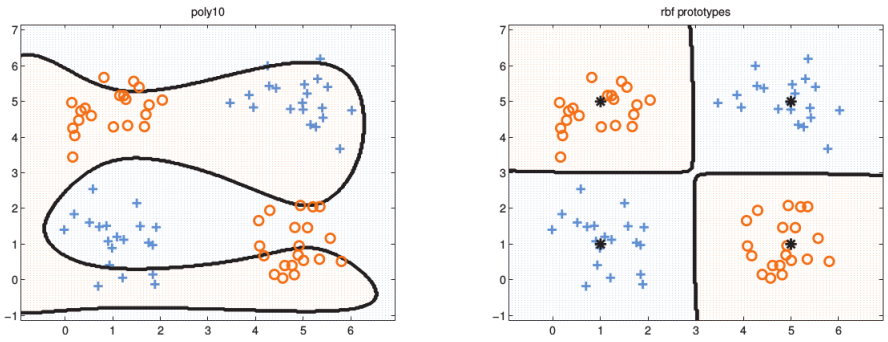
Na lijevoj slici prikazan je dvodimenzijski ulazni prostor ($n = 2$), u kojem imamo primjere iz dviju klasa (crvena i plava). Klase očito nisu linearno odvojive. Kako bismo ostvarili linearnu odvojivost, primjere ćemo preslikati u prostor značajki funkcijom

$$\boldsymbol{\phi}(\mathbf{x}) = (1, \phi_1(\mathbf{x}), \phi_2(\mathbf{x})) = (1, \kappa(x, \mu^1), \kappa(x, \mu^2))$$

gdje ćemo za bazne funkcije koristiti jezgre RBF. Preslikavanje ϕ_1 definirano je jezgrenom funkcijom sa središtem u $(-1, 1)$, a ϕ_2 je preslikavanje definirano jezgrenom funkcijom sa središtem u $(0, 0)$. Zeleni križići prikazuju središta tih dviju jezgrenih funkcija, a zelene kružnice prikazuju izokonture. Budući da imamo samo dvije jezgrene funkcije, preslikavamo u prostor značajki dimenzija $m = 2$ (ako zanemarimo težinu w_0). U prostoru značajki sada su na klase linearno odvojive (uzmite si vremena da shvatite zašto je to tako). Linearna granica (crni pravac) u prostoru značajki odgovara nelinarnoj granici (crnoj kružnici) u ulaznom prostoru. Primijetite da u ovom slučaju nismo preslikavali u prostor više dimenzije (imamo $n = m$), međutim preslikavanje je nelinearno, pa smo u prostoru značajki uspjeli ostvarili linearno odvajanje klasa.

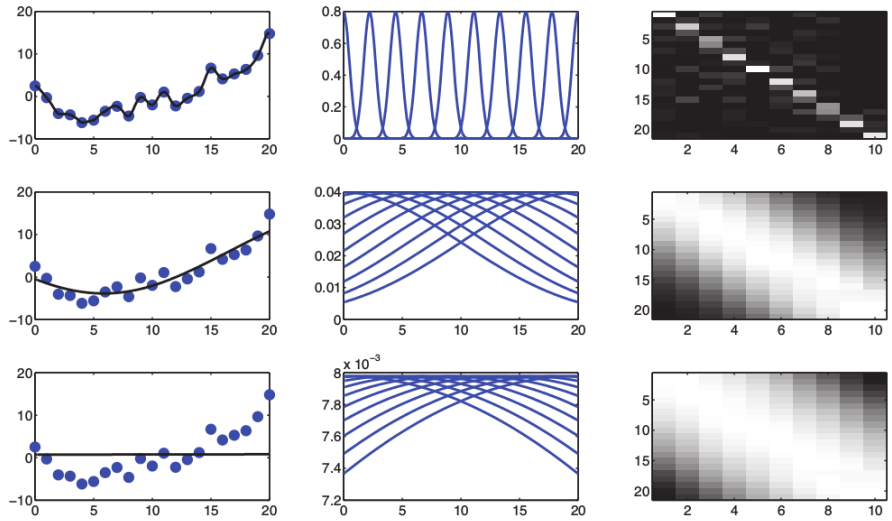
▸ PRIMJER

Ovaj je primjer sličan prethodom. Ponovno se radi o binarnom klasifikacijskom problemu u dvodimenzijskome ulaznome prostoru, koji nije linearno odvojiv (XOR-problem). Ako preslikavanje u prostor značajki definiramo kao jezgreni stroj sa četiri jezgre RBF, i središta tih jezgara postavimo u sredine svake od četiri klase u ulaznom prostoru, onda dobivamo nelinearnu granicu između klasa kakva je prikaza na desnoj slici. Na lijevoj je slici prikazana granica koju bismo dobili preslikavanjem polinomom 10. stupnja, koji za razliku od jezgre RBF, na ovom problemu ne ostvaruje savršeno točnu klasifikaciju.



▸ PRIMJER

Pogledajmo i jedan primjer s (jednostavnom) regresijom:



a Lijeva slika prikazuje ulazni prostor, u kojemu imamo 21 primjer. Ovdje se radi o jednostavnoj regresiji i naš ulazni prostor je jednodimenzijski. Za preslikavanje koristimo 10 Gaussovih jezgrenih funkcija. Srednja slika prikazuje te Gaussove jezgrene funkcije, koje su ulaznom prostoru ekvidistantno postavljene. Desna slika prikazuje matricu dizajna: retci su primjeri, a stupci su značajke. U ovom slučaju značajke su vrijednosti Gaussove jezgre, njih ukupno 10. Svjetlije ćelije odgovaraju većim vrijednostima jezgrene funkcije (tj. većoj sličnosti između primjera i referentnog primjera), a tamnije ćelije odgovaraju manjim vrijednostima jezgrene funkcije (tj. manjoj sličnosti između primjera i referentnog primjera). Drugim riječima, na toj slici vidimo koliko je svaki od 21 primjera blizu središta svake od 10 Gaussovih distribucija. Vidimo prenaučen (prvi red), dobar (srednji red) i podnaučen model (donji red).

Ovo sve je vrlo lijepo, no postoji jedan problem koji smo prešutjeli: kako definirati referentne primjere $\boldsymbol{\mu}_j$? Mogli bismo ih rasporediti uniformno (ekvidistantno) po prostoru, kao što je to bio slučaj u posljednja dva gornja primjera. To je izvedivo ako je prostor niske dimenzije, ali ako imamo visokodimenzijski prostor, to postaje problematično. Trebalo bi nam vrlo mnogo (eksponencijalno mnogo u broju dimenzija) takvih primjera, a da uniformno pokrijemo visokodimenzijski prostor kroz sve dimenzije. Što bi bila alternativa uniformnom raspoređivanju referentnih primjera? Nekako bismo htjeli da ti primjeri u prostoru budu postavljeni tamo gdje ima pravih primjera, a ne tamo gdje ih nema. Kako to ostvariti? Pa, upravo tako da za referentne primjere uzmemo primjere iz skupa za učenje. Onda nam se očito ne može dogoditi da tako definirani referentni primjeri budu u nekim dijelovima ulaznog prostora gdje nema pravih primjera. Ako, dakle, za referentne primjere odaberemo primjere iz skupa za učenje, onda za funkciju preslikavanja u prostor značajki imamo: 7

$$\boldsymbol{\phi}(\mathbf{x}) = \left(1, \kappa(\mathbf{x}, \mathbf{x}^{(1)}), \kappa(\mathbf{x}, \mathbf{x}^{(2)}), \ldots, \kappa(\mathbf{x}, \mathbf{x}^{(N)})\right)$$

Sada je dimenzija prostora značajki jednaka prostoru primjera, $m = N$, dakle imat ćemo onoliko parametara koliko imamo primjera u skupu za učenje.

Međutim, to bi opet moglo biti problematično, jer primjera može biti previše. Stoga se postavlja pitanje: kako možemo smanjiti broj referentnih primjera? Primijetite da je u ovom slučaju to je isto kao da smo pitali: kako možemo smanjiti broj značajki? Kako smo to radili do sada? Regularizacijom! Međutim, to onda mora biti L_1-**regularizacija**, kako bismo težine pritegnuli skroz na nulu. Dakle, ako primijenimo L1-regularizaciju, pritegnut ćemo neke težine (u prostoru značajki!) na nulu, što efektivno znači da smo neke od jezgri izbacili, a to je isto kao

da smo odabrali neki podskup primjera iz skupa za učenje da nam služe kao referentni primjeri. Takve primjere onda možemo smatrati **prototipnim primjerima**. Nadalje, takve jezgreni strojevi, koji kao referentne primjere za izračun jezgri koriste manji broj primjera iz skupa za učnje, nazivamo **rijetki jezgreni strojevi** (engl. *sparse kernel machines*) ili **rijetki vektorski strojevi** (engl. *sparse vector machines*). Konkretno, kada koristimo L_1-regularizaciju, to zovemo **L1VM**. Ako koristimo L_2-regularizaciju, onda je to **L2VM**, no primijetite da nam to neće dati rijetki model, iz razloga koji nam je već poznat (L_2-regularizacija ne potiskuje težine do nule).

Sve ovo vas možda malo podsjeća na SVM. Sjetite se da smo tamo upravo imali podskup primjera iz skupa za učenje, koje smo zvali **potporni vektori**, s kojima smo uspoređivali druge primjere. Međutim, kod SVM-a nismo baš koristili te primjere da bismo preslikali u prostor značajki na ovakav način, nego je cijela ta priča ispala nekako "automatski" kod prijelaza iz primarnog u dualni optimizacijski problem.

Sada je zgodan trenutak da se malo vratimo na SVM i pogledamo to u svjetlu ove današnje priče. Vrijeme je da pogledamo drugi način kako možemo iskoristiti jegrene funkcije. Vrijeme je za – **jezgreni trik**!

3 Jezgreni trik

3.1 Nelinearan SVM

Vratimo se na pitanje s kojim smo bili završili prošlotjedno predavanja: kako od SVM možemo dobiti **nelinearan model**, tj. model s nelinearnom granicom između klasa u ulaznome prostoru? Znamo, jer smo to već mnogo puta rekli, da to možemo lako ostvariti tako da primjere preslikamo u prostor više dimenzije pomoću funkcije $\boldsymbol{\phi}$. Nadam se da će, nakon preslikavanja, ti primjeri u prostoru značajki biti linearno odvojivi, ili barem blizu toga da budu linearno odvojivi – ne moraju biti savršeno linearno odvojivi jer imamo **meku marginu**. Za funkciju preslikavanja možemo koristiti uobičajene funkcije koje smo koristili do sada (npr., polinomijalno preslikavanje) ili možemo koristiti raznorazne jezgrene funkcije, kao što bismo to radili kod jezgrenih strojeva.

Međutim, kod SVM-a možemo napraviti nešto puno bolje. Nema potrebe da prvo preslikavamo primjere u prostor značajki, pa onda primjenjujemo SVM-a. Umjesto toga, možemo direktno iskoristiti jezgrenu funkciju unutar samo algoritma (tj. unutar modela i optimizacijskog postupka). Dovoljno je primijetiti da SVM zapravo već koristi jezgru! Naime, prisjetimo se kako izgleda model SVM u dualnoj formulaciji:

$$h(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i y^{(i)} \mathbf{x}^{\mathrm{T}} \mathbf{x}^{(i)} + w_0$$

Ovaj skalarni umnožak $\mathbf{x}^{\mathrm{T}}\mathbf{x}^{(i)}$ nije ništa drugo nego **linearna jezgra**, koji računa sličnost primjera s potpornim vektorima! Ista se stvar javlja i kod treniranja SVM-a. Prisjetimo se, u dualnoj formulaciji, optimizacijski problem je ovaj:

$$\underset{\boldsymbol{\alpha}}{\operatorname{argmax}} \Big(\sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y^{(i)} y^{(j)} (\mathbf{x}^{(i)})^{\mathrm{T}} \mathbf{x}^{(j)} \Big)$$

uz određena ograničenja, koja ovdje više nećemo ponavljati. Vidimo da se i ovdje pojavljuje skalarni umnožak.

Štoviše, jedina mjesta gdje se u algoritmu SVM-a pojavljuju primjeri $\mathbf{x}$ zapravo ova dva mjesta gdje se oni pojavljuju u obliku skalarnog umnoška. Drugim riječima, vektori $\mathbf{x}$ nikada se ne pojavljuju sami, nego uvijek u skalarnom umnošku. To nam otvara jednu fantastičnu mogućnost. Zašto ne bismo skalarni umnožak na ta dva mjesta jednostavno zamijenili jezgrenom

funkcijom? Dakle, model bi bio:

$$h(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i y^{(i)} \kappa(\mathbf{x}, \mathbf{x}^{(i)}) + w_0$$

a ciljna funkcija kod optimizacije bila bi:

$$\sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y^{(i)} y^{(j)} \kappa(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$$

pri čemu $\kappa(\mathbf{x}, \mathbf{x}') = \mathbf{x}^{\mathrm{T}} \mathbf{x}'$, tj. koristimo **linearnu jezgru**. Što smo time dobili? Pa, iskreno, još ništa. Samo smo napravili zamjenu jednog izraza (skalarnog umnoška) ekvivalentnim izrazom (linearnom jezgrom). Međutim, zašto stati na linearnoj jezgri? To nam neće dati nelinarnost. Nelinearnost bismo dobili da smo primjere preslikali pomoću funkcije $\boldsymbol{\phi}$. U tom slučaju, umjesto skalarnog umnoška $\mathbf{x}^{\mathrm{T}} \mathbf{x}^{(i)}$, u izrazu za model i u izrazu za ciljnu funkciju imali bismo $\boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}^{(i)})$. Pa, dakle, ako želimo biti općenitiji, jezgru ćemo definirati da uključuje to preslikavanje, tj. ovako:

$$\kappa(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}')$$

Vidimo da ovako definirana jezgrena funkcija mjeri **sličnost** dvaju vektora nakon njihovog preslikavanja u **prostor značajki**, gdje je ta sličnost definirana konkretno pomoću **skalarnog umnoška**. Ako preslikavanja nema, tj. $\boldsymbol{\phi}(\mathbf{x}) = (\mathbf{x})$, onda smo nazad na linearnoj jezgri. 8

I to je zapravo taj famozni **jezgreni trik** (engl. *kernel trick*): jednostavno ćemo zamijeniti svako pojavljivanje ovog skalarnog umnoška u prostoru značajki jezgrenom funkcijom! No, zašto je to trik? Tu dolazimo do fantastičnog dijela. To je trik zato što mi nećemo stvarno izračunati tu jezgrenu funkciju kao skalarni umnožak u prostoru značajki. Naime, mi uopće nećemo preslikavati primjere u prostor značajki pomoću funkcije $\boldsymbol{\phi}$, nego ćemo direktno izračunati sličnost dvaju primjera pomoću jezgrene funkcije, i onda taj broj koji dobijemo umetnuti na mjesto skalarnog umnoška. Ono što je tu fantastično jest što možemo upotrijebiti (skoro pa) bilo koju jezgrenu funkciju, uključivo i one koje uopće nisu definirane kao skalarni umnožak vektora. Drugim riječima, tretirat ćemo jezgrenu funkciju kao "crnu kutiju": unutra ulaze dva primjera (bilo vektorska ili ne), van izlazi neki broj, a taj broj odgovara njihovoj sličnosti u prostoru značajki. Pritom nas uopće ne zanima da primjere doista preslikamo u prostor značajki, nego samo koliko bi iznosio njihov skalarni umnožak kada bismo ih doista preslikali. Poanta ovoga je da nema eksplicitnog preslikavanja, nego je preslikavanje **implicitno**! 9

Ovakav se pristup općenito naziva **inverzno oblikovanje**: umjesto da izravno preslikavamo primjere, mi izabiremo neku jezgrenu funkciju κ pa treniramo model s tom funkcijom, i istu tu funkciju koristimo kod predikcije. Inverzno oblikovanje ima tri vrlo važne prednosti nad preslikavanjem primjera u prostor značajki:

1. **Smanjenje računalne složenosti**: izračun jezgrene funkcije često je jednostavniji nego izračun dvaju preslikavanja pa zatim izračun skalarnog umnoška;

2. **Jednostavnost**: kao što smo već bili napomenuli, često je lakše definirati jezgrenu funkciju κ nego preslikavanje $\boldsymbol{\phi}$, a pored toga postoji niz standardnih jezgrenih funkcija. Ova prednost osobito dolazi do izražaja ako primjeri imaju nekakvu strukturu (znakovni nizovi, stabla, grafovi i sl.). Tada je puno lakše definirati sličnost nego značajke, a kamo li preslikavanje;

3. **Veća složenost modela**: Prostor značajki koji odgovara jezgrenoj funkciji može biti vrlo visokodimenzijski, potencijalno beskonačno dimenzijski! To ne možemo dobiti nikakvim preslikavanjem.

Sve ovo zvuči super! Jezgreni trik je super stvar, to je to! Međutim, ima jedna mala začkuljica. Pogledajmo još jednom kako smo definirali jezgrenu funkciju:

$$\kappa(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}')$$

Imajući na umu ovu definiciju, postavlja se pitanje: mogu li ovdje koristiti *baš svaku* jezgrenu funkciju? Odgovor je: ne mogu. Jezgrena funkcija mora odgovarati **skalarnom umnošku** primjera u prostoru značajki. Inače jezgreni trik ne funkcionira. Drugim riječima, skalarni umnožak $\boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}')$ smijem zamijeniti bilo kakvom jezgrenom funkcijom $\kappa(\mathbf{x}, \mathbf{x}')$, pod uvjetom da ta jezgrene funkcija odgovara onome što bih dobio kada bih doista preslikao vektore u prostor značajki i tamo ih skalarno pomnožio. S obzirom da jezgrenu funkciju možemo definirati kako god to želimo (sve dok ona zadovoljava osnovne uvjete mjere sličnosti: nenegativnost i simetričnost), očito je da neće svaka jezgrena funkcija zadovoljavati ovaj naš uvjet. Npr., jezgrena funkcija $\kappa(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|$ nije valjana jezgra jer ne postoji preslikavanje $\boldsymbol{\phi}$ takvo bi $\boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}')$ bilo jednako $\|\mathbf{x} - \mathbf{x}'\|$. 10

Srećom, postoji puno vrlo dobrih jezgri koje zadovoljavaju ovaj uvjet. Takve jezgre nazivamo **Mercerove jezgre** ili **pozitivno definitne jezgre** ili **valjane jezgre** (engl. *valid kernels*). 11 Usredotočimo se u nastavku na takve jezgre.

3.2 Mercerove jezgre

U nastavku će nam biti korisno da jezgrenu funkciju prikažemo matrično. Primijetite, naime, da kod treniranja modela računamo vrijednost jezgrene funkcije između svih parova primjera iz skupa primjera $\mathcal{D}$. Te vrijednosti možemo izračunati unaprijed i pohraniti u simetičnu matricu dimenzija $N \times N$. Tu ćemo matricu nazvati **jezgrena matrica** (engl. *kernel matrix*) i označiti sa $\mathbf{K}$. Dakle, jezgrena matrica izgleda ovako:

$$\mathbf{K} = \begin{pmatrix} \kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(1)}) & \kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) & \dots & \kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(N)}) \\ \kappa(\mathbf{x}^{(2)}, \mathbf{x}^{(1)}) & \kappa(\mathbf{x}^{(2)}, \mathbf{x}^{(2)}) & \dots & \kappa(\mathbf{x}^{(2)}, \mathbf{x}^{(N)}) \\ \vdots & \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}^{(N)}, \mathbf{x}^{(1)}) & \kappa(\mathbf{x}^{(N)}, \mathbf{x}^{(2)}) & \dots & \kappa(\mathbf{x}^{(N)}, \mathbf{x}^{(N)}) \end{pmatrix}$$

Sada se pitamo: ako je matrica $\mathbf{K}$ doista rezultat skalarnih umnožaka u nekom vektorskom prostoru značajki, kakva ona mora biti, a da možemo reći da taj prostor stvarno postoji? Drugim riječima: kakva mora biti jezgrena matrica ako je jezgra Mecerova? Može se dokazati da je svaka matrica skalarnih umnožaka pozitivno definitna, i obrnuto, svaka pozitivno definitna matrica je matrica skalarnih umnožaka za neki skup vektora u nekom vektorskom prostoru (prisjetimo se, matrica $\mathbf{A}$ je pozitivno semidefinitna ako vrijedi $\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} \geqslant 0$). Takvu matricu zovemo **Gramova matrica** (taj smo naziv već susreli kada smo pričali u regresiji). Rekli smo već da je Mercerova 12 jezgra ona jezgra koja odgovara skalarnome umnošku vektora u prostoru značajki. Prema tome, ako je jezgra κ Mercerova, onda je matrica $\mathbf{K}$ nužno pozitivno semidefinitna, tj. jezgrena matrica je Gramova matrica, definirana na sljedeći način:

$$\mathbf{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}$$

gdje je $\boldsymbol{\Phi}$ je matrica dizajna

Sada kada znamo da Mercerova jezgra daje vrijednost skalarnog umnoška dvaju vektora u prostoru značajki, možemo se zapitati: kakav je taj prostor? Savim sigurno, to je **vektorski prostor**, koji očito mora imati definiranu **operaciju skalarnog umnoška**. Međutimm taj prostor može imati vrlo mnogo dimenzija, uključivo beskonačno mnogo dimenzija. Takav vektorski prostor, koji ima definiran **skalarni umnožak**, ali koji je proizvoljne dimenzije, uključivo beskonačnog broja dimenzija, naziva se **Hilbertov prostor** $\mathcal{H}$. To uključuje i konačno-dimenzijski višedimenzijski prostor $\mathbb{R}^n$, s kakvim smo do sada radili. Dakle, da bi jezgreni trik funkcionirao, jezgrena funkcija mora biti Mercerova jezgra, odnosno matrica $\mathbf{K}$ mora biti pozitivno semidefinitna, i onda će ona odgovarati skalarnom umnošku u nekom Hilbertovom prostoru.

3.3 Izgradnja jezgri

Toliko o teoriji. Okrenimo se sada praktičnim pitanjima. Osnovno pitanje očito glasi: kako izgraditi Mercerove jezgre? Možemo li izmisliti neku jezgrenu funkciju i onda dokazati da je Mercerova? Mogli bismo, ali to je općenito dosta zahtjevno i traži dosta znanja funkcijske analize. Alternativa je da izračunamo jezgrenu matricu $\mathbf{K}$ na našem skupu primjera $\mathcal{D}$ te da provjerimo je li to Gramova matrica, tj. je li ta matrica pozitivno semidefinitna. Očito, to nije formalan dokaz da je jezgra Mercerova, ali u praksi može biti dovoljno dobro, pogotovo ako je skup $\mathcal{D}$ razmjerno velik. 13

Međutim, postoji niz **standardnih jezgrenih funkcija** za koje se zna da su Mercerove jezgre. Na primjer:

- **Linearna jezgra**: $\kappa(\mathbf{x}, \mathbf{x}') = \mathbf{x}^{\mathrm{T}}\mathbf{x}'$, koja efektivno daje linearan model
- **Polinomijalna jezgra**: $\kappa(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^{\mathrm{T}}\mathbf{x}' + 1)^p$, koja uključuje linearne članove, kombinacije ulaznih varijabli (interakcije) i potencije do uključivo stupnja p
- **Gaussova RBF jezgra**, koju smo već vidjeli,
- **Jezgra znakovnih nizova**, o kojoj smo također već pričali.
- ... i mnoge druge. 14

Za linearnu jezgru nam je jasno da je Mercerova, jer očito odgovara skalarnom umnošku, i to izravno u ulaznom prostoru primjera. Što je s polinomijalnom jezgrom? Može se dokazati da je i ona Mercerova. No, umjesto toga, da dobijemo osjećaj da je to tako, pogledajmo jedan primjer.

▸ **PRIMJER**

Neka je jezgrena funkcija $\kappa(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^{\mathrm{T}}\mathbf{z})^2$. Provjerimo odgovara li ova jezgrena funkcija skalarnom umnošku u nekom prostoru značajki. To ćemo napraviti tako da ćemo raspisati izraz $(\mathbf{x}^{\mathrm{T}}\mathbf{z})^2$, i zatim provjeriti odgovara li on doista skalarnom umnošku neka dva vektora. Jednostavnosti radi, a ne smanjujući općenitost, ograničimo se na dvodimenzijski prostor, $n = 2$. Prvi vektor neka je $\mathbf{x} = (x_1, x_2)$, a drugi $\mathbf{z} = (z_1, z_2)$. Onda imamo:

$$\begin{aligned}\kappa(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^{\mathrm{T}}\mathbf{z})^2 &= ((x_1, x_2)^{\mathrm{T}}(z_1, z_2))^2 = (x_1z_1 + x_2z_2)^2 \\ &= (x_1z_1)^2 + 2(x_1z_1)(x_2z_2) + (x_2z_2)^2 = x_1^2z_1^2 + \sqrt{2}x_1x_2\sqrt{2}z_1z_2 + x_2^2z_2^2 \\ &= (x_1^2, \sqrt{2}x_1x_2, x_2^2)^{\mathrm{T}}(z_1^2, \sqrt{2}z_1z_2, z_2^2) = \boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{z})\end{aligned}$$

Vidimo da jezgrena funkcija $\kappa(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^{\mathrm{T}}\mathbf{z})^2$ doista odgovara skalarnom umnošku u prostoru značajki, i to s preslikavanjem definiranim kao $\boldsymbol{\phi}(\mathbf{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$.

Primijetimo također da je u ovom slučaju **računalno jednostavnije** izravno izračunati jezgrenu funkciju (ukupno 4 računske operacije) nego skalarni umnožak u prostoru značajki (2×4 operacija za preslikavanje i dodatnih 5 operacija za skalarni umnožak = 13 operacija).

3.4 Gaussova jezgra

Vrlo lijepo, linearna i polinomijalna jezgra su Mercerove. A što je sa Gaussovom jezgrom? Prisjetimo se, Gaussova jezgra definirana je ovako:

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\big(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\big) = \exp\big(-\gamma\|\mathbf{x} - \mathbf{x}'\|^2\big)$$

Može se dokazati da je Gaussova jezgra također Mercerova, no to ćemo ovdje preskočiti. Ono što je još zanimljivije jest pitanje kako izgleda prostor značajki u koji ona preslikava. Pogledajmo to na intuitivnoj razini. Ovdje ćemo reći dvije stvari. Prva se tiče **dimenzionalnosti** prostora 15

značajki. Može se pokazati da je Gramova matrica $\mathbf{K}$ za Gaussovu jezgru matrica punog ranga. To znači da su vektori $\boldsymbol{\phi}(\mathbf{x})$ linearno nezavisni. Efekt toga jest da je prostor značajki (prostor u koji preslikavamo) beskonačno dimenzijski! Točnije, taj prostor ima onoliko dimenzijski koliko imamo primjera, a na broj primjera nema gornje granice!

Druga stvar tiče se sličnosti između vektora u tom beskonačno dimenzijskom prostoru. Već znamo da Gaussova jezgra mjeri sličnost dvaju primjera temeljem njihove udaljenosti u vektorskom prostoru. Za slične primjere vrijedi $\kappa(\mathbf{x}, \mathbf{x}') \to 1$, pa su ti primjeri očito onda blizu u prostoru značajki, odnosno vektori su im slični nakon preslikavanja, jer je skalarni umnožak najviši mogući (ionako nije više od 1, kada koristimo Gaussovu jezgru). S druge strane, za vrlo različite primjere vrijedi $\kappa(\mathbf{x}, \mathbf{x}') \to 0$. Budući da je Gaussova jezgra Mercerova, to onda znači da je skalarni umnožak u prostoru značajki teži prema nuli. A što to onda znači? To znači da su primjeri u prostoru značajki međusobno ortogonalni (odnosno ortonormalni, jer su maksimalne duljine 1, budući da sličnost prema Gaussovoj jezgri može biti najviše jednaka 1). Pogledajmo parametar Gaussove jezgre σ^2 (širina pojasa) odnosno γ (preciznost). Taj parametar kontrolira kojom brzinom $\kappa(\mathbf{x}, \mathbf{x}')$ teži k nuli u ovisnosti o udaljenosti. Ako je γ malen, $\kappa(\mathbf{x}, \mathbf{x}') \to 1$ i primjeri su u prostoru značajki grupirani zajedno, što lako dovodi do **podnaučenosti**. S druge strane, ako je γ velik, onda $\kappa(\mathbf{x}, \mathbf{x}') \to 0$ (izuzev za $\mathbf{x} = \mathbf{x}'$), pa su sve točke u prostoru značajki međusobno ortogonalne, što pak lako dovodi do **prenaučenosti**. Sve u svemu, različite vrijednosti za γ daju različita preslikavanja, pa stoga je taj parametar vrlo važan. Intuitivno, vidimo da, ako odaberemo dovoljno velik γ, moći ćemo preslikati primjere tako da su oni međusobno okomiti.

Zbrojimo li ova dva efekta – **beskonačnu dimenzionalnost** i **ortonormiranost** svih vektora iz skupa primjera za učenje (za dovoljno visok γ) – dobivamo to da je svaki primjer u svojoj dimenziji i da je skroz drugačiji od drugih. To efektivno znači da je svaki primjer "poslan u svoj vrh" duž ortogonalne osi. U takvom prostoru uvijek možemo linearno odvojiti primjere kako god oni bili označeni, a da klasifikacija bude savršeno točna. 16

Što to znači? To znači da (za dovoljno veliku γ), možemo dobiti **savršenu linearnu odvojivost** u prostoru značajki. Prostor značajki će biti onoliko dimenzijski koliko imamo primjera.

3.5 Izgradnja složenijih jezgara

Ovo je opet vrlo lijepo, i zapravo ne možemo ni tražiti više od beskonačnodimenzijskog prostora. No, što ako ipak trebamo složenije jezgre, zato jer su naši primjeri složeniji? Pritom mislimo na situacije kada naši primjeri nisu vektori, nego imaju složeniju strukturu. Npr., zamislimo da se svaki naš primjeri sastoje od jednog znakovnog niza i od jednog vektora. Ako želimo pomoću jezgrene funkcije računati sličnost između takvih primjera, očito nam treba složenija jezgrena funkcija, koja istovremeno može raditi i sa znakovnim nizom i sa vektorom. K tome još želimo da ta jezgra bude Mercerova.

Dobra vijest je da, koristeći standardne jezgrene funkcije koje smo gore naveli kao gradivne blokove, možemo konstruirati složenije jezgrene funkcije. Pritom za kombiniranje Mercerovih jezgri možemo koristiti niz operacija za koje je dokazano da daju jezgre koje su opet Mercerove (engl. *Mercer-preserving operations*). Na primjer, ako su κ_1 i κ_2 Mercerove jezgre, onda će to

biti i jezgra κ dobivena pomoću sljedećih operacija:

$$\kappa(\mathbf{x}, \mathbf{x}') = \alpha\kappa_1(\mathbf{x}, \mathbf{x}') \qquad \alpha > 0$$
$$\kappa(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})\kappa_1(\mathbf{x}, \mathbf{x}')f(\mathbf{x}') \qquad f \text{ – bilo koja funkcija}$$
$$\kappa(\mathbf{x}, \mathbf{x}') = q\big(\kappa_1(\mathbf{x}, \mathbf{x}')\big) \qquad q \text{ – polinom s pozitivnim koeficijentima}$$
$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\big(\kappa_1(\mathbf{x}, \mathbf{x}')\big)$$
$$\kappa(\mathbf{x}, \mathbf{x}') = \kappa_1\big(\boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x})\big) \qquad \boldsymbol{\phi} : \mathbb{R}^n \to \mathbb{R}^{m+1}$$
$$\kappa(\mathbf{x}, \mathbf{x}') = \kappa_1(\mathbf{x}, \mathbf{x}') + \kappa_2(\mathbf{x}, \mathbf{x}')$$
$$\kappa(\mathbf{x}, \mathbf{x}') = \kappa_1(\mathbf{x}, \mathbf{x}')\kappa_2(\mathbf{x}, \mathbf{x}')$$

Kombiniranje više jezgri zove naziva se **učenje s više jezgara** (engl. *multiple kernel learning, MKL*). Najjednostavnije je zbrojiti dvije jezgre (i lako je pokazati da zbroj dviju jezgri zapravo odgovara konkatenaciji dvaju vektora u prostoru značajki, što je općenito najjednostavniji način kombiniranja značajki). 17

4 Napomene

4.1 SVM

Prisjetimo se da SVM ima **hiperparametar** C koji određuje njegovu složenost. Manja vrijednost za C znači da će model dozvoljavati više pogrešaka, tj. bit će jednostavniji. Obrnuto, veća vrijednost za C znači da će model više kažnjavati pogreške, pa će biti složeniji. Taj hiperparametar tipično optimiramo unakrsnom provjerom.

Ako koristimo (nelinearnu) jezgrenu funkciju, onda trebamo odabrati i **hiperparametar jezgrene funkcije** (npr., stupanj polinoma p ili preciznost γ). Ovi su parametri povezani s hiperparametrom C. Npr., ako odaberemo visoku vrijednost za γ, dobit ćemo složeniji model, pa će trebati pojačati regularizaciju odabirom manje vrijednosti za C. To znači da unakrsnom provjerom trebamo optimirati dva hiperparametra istovremeno, što tipično činimo iscrpnim pretraživanjem u unaprijed definiranom rasponu, tzv. **pretraživanjem po rešetci** (engl. *grid search*).

4.2 Linearna jezgra

Bismo li ikada želeli koristiti linearnu jezgru? Da, ako želimo **rijetki model**: bit će lakše hiperravninu prikazati s nekoliko potpornih vektora, nego s hrpom težina. Ta prednost naravno dolazi do izražaja kod predikcije. Kod učenja ionako moramo izračunati jezgrenu funkciju između svih parova primjera.

Druga prednost bi mogla biti da nam se više isplati raditi optimizaciju u dualu, koju algoritmom SMO možemo napraviti sa složenošću $O(N^2)$, dok bi u primarnoj formulaciji morali koristiti generički rješavač kvadratnog programa, pa bi složenost bila $O(n^3)$ (doduše, možemo koristiti gradijentni spust sa pogreškom definiranom na temelju gubitka zglobnice, pa tada nemamo taj problem).

4.3 Aproksimacija jezgrenih funkcija

Jezgreni trik je izvrsna ideja, ali će izračun jezrene matrice biti dosta skup kada imamo puno primjera. S druge strane, optimizacija gradijentnim spustom vrlo je računalno jeftina, međutim tamo ne možemo koristiti jezgre Ako želimo najbolje od dva svijeta, možemo upotrijebiti **aproksimaciju jezgre**. To znači da ne koristimo jezgreni trik, nego doista primjere stvarno preslikamo u prostor značajki, ali to preslikavanja radimo približno. U tom prostoru onda radimo stohastički gradijentni spust.

4.4 Ujezgravanje algoritama

Jezgreni trik nije primjenjiv samo na SVM. Doduše, kod njega je očit, jer se skalarni umnožak automatski javio kod dualne formulacije optimizacijskog problema. Međutim, kod mnogih drugih modela moguće je također primijeniti jezgreni trik, ako se model preformulira tako da se vektori primjera javljaju kod predikcije (u definiciji modela) i kod optimizacije isključivo u obliku skalarnog umnoška. Ako to uspijemo napraviti, onda znači da možemo **ujezgriti (kernelizirti** (engl. *kernelize*) algoritam. Npr., i linearna i logistička regresija se mogu vrlo jednostavno ujezgriti. [18]

Sažetak

- **Jezgrene funkcije** mjere sličnost dvaju primjera
- Primjeri mogu biti vektori ili neke druge strukture (npr. stringovi, stabla, grafovi)
- Postoji niz standardnih jezgrenih funkcija, npr. **linearna**, **polinomijalna** i **Gaussova jezgra**, te operacija za izgradnju složenijih jezgara
- **Jezgreni strojevi** su poopćeni linearni modeli s jezgrenim funkcijama kao baznim funkcijama
- **Jezgreni trik**: skalarni umnožak vektora u prostoru značajki mijenjamo Mercerovom jezgrenom funkcijom. Trik je primjenjiv kod SVM-a, ali i nekih drugih algoritama
- Jezgreni trik omogućava **inverzno oblikovanje**: učinkovitije, jednostavnije, ekspresivnije
- **Gaussova jezgra** je Mercerova i preslikava u beskonačnodimenzijski prostor značajki

Bilješke

[1] **Jezgrene metode** (engl. *kernel methods*) jedno su od osnovnih područje strojnog učenja, od velikog teorijskog i praktičnog značaja. Standardni uvodni tekstovi u ovo područje su (Scholkopf and Smola, 2018), (Shawe-Taylor et al., 2004) i (Hofmann et al., 2008).
Što se naziva "jezgra" tiče, on dolazi iz funkcijske analize s početka 20. stoljeća. Isti se naziv počeo koristiti u statistici tek u sedamdesetim godina prošlog stoljeća u kontekstu neparametarskih metoda, konkretno **jezgreni procjenitelji gustoće vjerojatnosti** (engl. *kernel density estimators*). Više ovdje: `https://stats.stackexchange.com/a/341842/93766`

[2] Naravno, nije nemoguće riječi jezika na neki smislen način prikazati kao vektor. Npr., riječi bismo mogi prikazati u prostoru razapetom slovima abecede, gdje bi svaka značajka odgovarala broju pojavljivanja određenog slova u toj riječi. U konkretnom primjeru s prepoznavanjem tuđica, mogli bismo smisliti i mnogo pametnije značajke od toga, npr. prisustvo određenih sufikasa ili općenito morfema i slično. U **području obrade prirodnog jezika** (engl. *natural language processing*) riječi se doista prikazuju kao vektori u semantičkom vektorskom prostoru, tako da ti vektori odgovaraju značenju riječi u smislu da semantički slične i srodne riječi imaju bliske vektore (v. Almeida and Xexéo (2019)). Ipak, u nekim je situacijama mnogo lakše razmišljati izravno o sličnosti između riječi, nego o tome kako ih preslikati u vektore. Ovo zapažanje vrijedi općenito: premda je načelno moguće bilo koji apstraktni objekt $\mathbf{x} \in \mathcal{X}$ prikazati kao vektor konačnih dimenzija, nije uvijek jasno kako to najbolje napraviti, i tada je često lakše formalizirati koncept sličnosti između tih objekata pomoću jezgrene funkcije.

[3] Nemojte brkati **mjeru sličnosti** (engl. *similarity measure*) s mjerom različitosti odnosno s funkcijom udaljenosti. Objasnimo razlike između ovih pojmova, krenuvši od ovog posljednjeg, koji je najspecifičniji. **Funkcija udaljenosti** (engl. *distance function*) ili **udaljenosti** ili **metrika** je funkcija $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_0^+$ za koju vrijede sljedeća svojstva:

(1) $d(\mathbf{x}^a, \mathbf{x}^b) \geqslant 0$ (nenegativnost)

(2) $d(\mathbf{x}^a, \mathbf{x}^b) = 0$ ako i samo ako $\mathbf{x}^a = \mathbf{x}^b$ (strogost)

(3) $d(\mathbf{x}^a, \mathbf{x}^b) = d(\mathbf{x}^b, \mathbf{x}^a)$ (simetričnost)

(4) $d(\mathbf{x}^a, \mathbf{x}^b) + d(\mathbf{x}^b, \mathbf{x}^c) \geqslant d(\mathbf{x}^a, \mathbf{x}^c)$ (nejednakost trokuta)

Svojstva (1) i (2) zajedno nazivaju se pozitivna definitnost. Strikto gledano, svojstvo (1) nije aksiomatsko, tj. može se izvesti iz preostala tri svojstva. Postoji niz standardnih metrika za vektorske prostore $\mathcal{X}$, npr. euklidska udaljenost ili Manhattanova udaljenost, odnosno općenitije Minkowskijeva udaljenost, koju smo već bili spominjali. Metrika, međutim, nije ograničena isključivo na vektorske prostore. Npr., metriku možemo definirati nad grafovima, nizovima znakova (npr. Levenshteinova udaljenost) ili vjerojatnosnim distribucijama (npr. Wassersteinova metrika).
Općenitiji pojam od udaljenosti je **mjera sličnosti** (engl. *similarity measure*), odnosno njoj komplementarna **mjera različitosti** (engl. *dissimilarity measure*). Udaljenost se može tumačiti kao geometrijska interpretacija sličnosti, odnosno različitosti. Bitna razlika jest u tome što mjera sličnosti (odnosno mjera različitosti) nije metrika. Mjera sličnosti je funkcija $s : \mathcal{X} \times \mathcal{X} \to [0, 1]$ za koju se obično podrazumijeva da zadovoljava sljedeće:

(1) $s(\mathbf{x}, \mathbf{x}) = 1$

(2) $0 \leqslant s(\mathbf{x}, \mathbf{x}') \leqslant 1$

(3) $s(\mathbf{x}, \mathbf{x}') = s(\mathbf{x}', \mathbf{x})$

Iz navedenih je svojstava očigledno da mjera udaljenosti i mjera sličnosti nisu suprotne u smislu da je jedna komplement druge, već su suprotne u smislu da je jednu moguće preslikati u drugu nekom monotono padajućom funkcijom. Na primjer, za preslikavanje udaljenosti d u sličnost s često se koristi:

$$s(\mathbf{x}, \mathbf{x}') = \frac{1}{1 + d(\mathbf{x}, \mathbf{x}')} .$$

Obrnuti smjer, pretvorba sličnosti u udaljenosti, nešto je teži zbog uvjeta nejednakosti trokuta. U većini slučajeva, udaljenosti i sličnosti (odnosno različitosti) mogu se koristiti jednako. U nekim slučajevima međutim sličnosti pružaju dodatnu fleksibilnost.

4 Više o jezgrama za grafove možete naći u (Gärtner, 2003) i (Kriege et al., 2020).

5 Mahalnobisova udaljenost svodi se dakle na euklidsku udaljenost za $\mathbf{\Sigma} = \mathbf{I}$. Međutim, možemo pojednostaviti Mahalanobisovu udlajenost, a ne svesti je odmah na euklidsku. Ako kovarijacijsku matricu definiramo kao dijagonalnu matricu, $\mathbf{\Sigma} = \text{diag}(\sigma_j^2)$, to znači da između značajki nema kovarijacije, ali dopuštamo da su varijance značajki različite. S dijagonalnom kovacijacijskom matricom dobivamo ovakvu jezgrenu funkciju:

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\Big(-\frac{1}{2} \sum_{j=1}^{n} \frac{1}{\sigma_j^2} (x_j - x_j')^2 \Big)$$

koja zapravo samo skalira svaku dimenziju ovisno o varijanci te dimenzije, čime se upravo anuliraju razlike u skalama između dimenzija. Takva jezgru nazivamo **jezgra ARD** (engl. *automatic relevance determination*). Primijetite da je jezgra ARD istovjetna Gaussovoj jezgri s euklidskom udaljenošću između primjera nad kojima je prethodno provedena standardizacija.

6 Prvi primjer preuzet je iz (Bishop, 2006) (poglavlje 4.3.1), a iduća dva iz (Murphy, 2012) (poglavlje 14.3.1).

7 Pored toga, u visokodimenzijskim prostorima javlja se problem **prokletstva dimenzionalnosti** (engl. *curse of dimensionality*): podatci postaju rijetki i jednako udaljeni jedan od drugih, tj. sve sličniji jedni drugima. Više o prokletstvu dimenzionalnosti idući put.

8 Pokoji pedantan čitatelj ovdje će primijetiti malu nekonzistentnost u notaciji: ako preslikavanja nema, onda bi funkcija $\boldsymbol{\phi}$ trebala biti definirana kao $\phi(\mathbf{x}) = (1, \mathbf{x})$, tako da uključi i "dummy"

jedinicu koja množi težinu w_0. Međutim, kod SVM-a to nije slučaj, budući da je težina w_0 izuzeta i tretiramo ju posebno.

9 **Jezgreni trik** zapravo radi po principu "što bi bilo kad bi bilo": nećemo doista preslikati vektore u prostor značajki, ali možemo reći koliko bi iznosio njihov skalarni umnožak kada bismo to doista učinili. U neku ruku, jezgreni je trik prečica koja nam omogućava da dođemo do informacije koja nas zanima, a da si pritom uskratimo muku izračuna. Ovdje do izražaja dolazi razlika između **ekstenzionalne** i **intenzionalne jednakosti**, koju smo već bili spomenuli u kontekstu jednakosti funkcija hipoteza. Naime, budući da $\kappa(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}')$, to su $\kappa(\mathbf{x}, \mathbf{x}')$ i $\phi(\mathbf{x})^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}')$ ekstenzionalno jednaki, međutim nisu intenzionalno jednaki, budući da $\kappa(\mathbf{x}, \mathbf{x}')$ možda možemo izračunati na drugačiji način nego da doista preslikamo vektore i skalarno ih pomnožim. U računarstvu je razlika između intenzionalne i ekstenzionalne jednakosti itekako važna, jer nam nije svejedno na koji način izračunavam funkciju: neki su načini skuplji, a neki jeftiniji. Jednostavan primjer koji ilustrira drastičnost te razlike je suma beskonačnog geometrijskog niza: geometrijski red kovergira i ima konačnu sumu ako je kvocijent manji od jedinice, međutim očito tu sumu nećemo moći izračunati rekurzivnim generiranjem i zbrajanjem svih elemenata niza.

10 Jezgra $\kappa(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|$ nije valjana (Mercerova) jezgra. Pretpostavimo suprotno: da postoji preslikavanje $\boldsymbol{\phi} : \mathbb{R}^n \to \mathbb{R}^m$ takvo da $\kappa(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}')$. Prema definiciji ove jezgre vrijedi $\kappa(\mathbf{x}, \mathbf{x}) = 0$, a to povlači $\boldsymbol{\phi}(\mathbf{x}) = \mathbf{0}$, budući da jedino tako skalarni umnožak svakog vektora sa samim sobom može biti jednak nuli. No, ako $\boldsymbol{\phi}(\mathbf{x}) = \mathbf{0}$, onda $\kappa(\mathbf{x}, \mathbf{x}') = 0$ za bilo koji par vektora $\mathbf{x}$ i $\mathbf{x}'$, a to je u kontradikciji s definicijom funkcije κ. Stoga ne postoji $\boldsymbol{\phi}$ takvo $\boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|$, pa zaključujemo da $\kappa(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|$ nije valjana jezgra.

11 Naziv **Mercerove jezgre** dolazi od **Mercerovog teorema**, koji je jedan od najvažnijih rezultata funkcijske analize za teoriju računalnog učenja. Teorem je 1909. godine dokazao britanski matematičar James Mercer. Opis i dokaz možete pronaći ovdje: http://homes.cs.washington.edu/~thickstn/docs/mercer.pdf

12 **Gramova matrica** je matrica skalarnih umnožaka vektora. Ta je matrica simetrična i pozitivno semidefinitna. Dokaz da je Gramova matrica pozitivno semidefinitna i da je svaka pozitivno semidefinitna matrica Gramova matrica je jednostavan i možete ga pogledati ovdje: https://en.wikipedia.org/wiki/Gramian_matrix#Positive-semidefiniteness.
U ovom predmetu Gramovu matricu susrećemo na dva mjesta, i svaki put je definiramo drugačije, što može dovesti do zabune. Prvo mjesto gdje smo susreli Gramovu matricu je kod regresije. Tamo smo Gramovu matrica definirali kao $\mathbf{X}^{\mathrm{T}}\mathbf{X}$, gdje je $\mathbf{X}$ matrica dizajna dimenzija $N \times (n+1)$, odnosno kao $\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}$, gdje je $\boldsymbol{\Phi}$ matrica dizajna u prostoru značajki dimenzija $N \times (m+1)$. Tako definirana Gramova matrica odgovara skalarnome umnošku vektor-stupaca, koji pak odgovaraju značajkama. Dimenzija te Gramove matrice je $(n+1) \times (n+1)$, odnosno $(m+1) \times (m+1)$. U statistici, tu matricu također nazivamo **matrica raspršenja** (engl. *scatter matrix*). Drugo mjesto gdje susrećemo Gramovu matricu jest ova naša današnja tema: naime, jezgrena matrica **Mercerove jezgre** je Gramova matrica. U ovom slučaju, vektori čiji skalarni umnožak računamo su pojedinačni primjeri, dakle vektor-retci matrice dizajna, a ne vektor-stupci kao kod regresije. Gramova je matrica u tom slučaju definirana kao $\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}$, i njezina je dimenzija $N \times N$. To su, dakle, skalarni umnošci između svih parova primjera iz skupa od N primjera.
Gramova matrica nazvana je po danskom matematičaru s kraja 18. i početka 19. stoljeća, Jorgenu Pedersenu Gramu. Gram se bavio teorijom vjerojatnošću, numeričkom analizom, teorijom brojeva i aktuaristikom (modeliranjem financijskog rizika), a osnovao je i svoju osiguravateljsku tvrtku. Poginuo je 1916. godine kada je na njega u Kopenhagenu naletio bicikl. Ferovcima je Gram vjerojatno poznat po Gram-Schmidtovom postupku ortogonalizacije u linearnoj algebri.

13 Provjeru je li matrica $\mathbf{K}$ pozitivno semidefinitna možemo napraviti tako da izračunamo njezine **svojstvene vrijednosti**. Naime, matrica je pozitivno semidefinitna ako i samo ako su sve njezine svojstvene vrijednosti nenegativne. Za nalaženje svojstvenih vrijednosti matrice u numeričkoj analizi postoji niz algoritama; v. https://en.wikipedia.org/wiki/Eigenvalue_algorithm. Tipično se koristi ili rastav matrice na singularne vrijednosti (SVD) ili neki iterativan algoritam za izračun svojstvenih vrijednosti, npr., Arnoldijev algoritam, koji se temelji na gore spomenutoj Gram-Schmidtovoj ortogonalizaciji. Standardna referenca za numeričke algoritme nad matricama je (Van Loan and Go-

lub, 1983).

14 Još primjera Mercerovih jezgri možete vidjeti ovdje: https://en.wikipedia.org/wiki/Positive-definite_kernel#Examples_of_p.d._kernels

15 Dokaz da je **Gaussova jezgra** Mercerova jezgra:

$$\begin{aligned}\kappa(\mathbf{x},\mathbf{x}') &= \exp(-\gamma\|\mathbf{x}-\mathbf{x}'\|^2)\\ &= \exp\big(-\gamma(\mathbf{x}-\mathbf{x}')^{\mathrm{T}}(\mathbf{x}-\mathbf{x}')\big)\\ &= \exp\big(-\gamma(\mathbf{x}^{\mathrm{T}}\mathbf{x}'-2\mathbf{x}^{\mathrm{T}}\mathbf{x}'+(\mathbf{x}')^{\mathrm{T}}\mathbf{x}')\big)\\ &= \exp(-\gamma\mathbf{x}^{\mathrm{T}}\mathbf{x}')\exp(2\gamma\mathbf{x}^{\mathrm{T}}\mathbf{x})\exp(-\gamma(\mathbf{x}')^{\mathrm{T}}\mathbf{x}')\\ &= f(\mathbf{x})\exp\big(\alpha\kappa'(\mathbf{x},\mathbf{x}')\big)f(\mathbf{x}')\end{aligned}$$

Dobili smo izraz koji je umnožak funkcije koja ovisi samo o vektoru $\mathbf{x}$, funkcije koje ovisi samo vektoru $\mathbf{x}'$, te eksponcijalne funkcije od skalarnog umnoška (koji jest Mercerova jezgra) pomnoženog s nekim skalarom $\alpha = 2\gamma$. Na ovom mjestu možemo se pozvati na operacije nad Mercerovim jezgrama koje zadržavaju svojstvo Mercerove jezgre (v. glavni tekst u nastavku) te zaključiti da je dobiven izraz Mercerova jezgra.

16 Detaljnije u http://stats.stackexchange.com/a/80405/93766, https://math.stackexchange.com/q/130554/273854, i http://pages.cs.wisc.edu/~matthewb/pages/notes/pdf/svms/RBFKernel.pdf.

17 Pokažimo da zbroj dviju jezgri odgovara **konkatenaciji vektora** u prostoru značajki. Krenimo od konkatenacije vektora. Neka su

$$\begin{aligned}\boldsymbol{\phi}(\mathbf{x}) &= [\boldsymbol{\phi}_1(\mathbf{x}),\boldsymbol{\phi}_2(\mathbf{x})]\\ \boldsymbol{\phi}(\mathbf{z}) &= [\boldsymbol{\phi}_1(\mathbf{z}),\boldsymbol{\phi}_2(\mathbf{z})]\end{aligned}$$

dva vektora u prostoru značajki, svaki dobiven konkatenacijem dvaju vektora. Mercerova jezgra primijenjena na vektore $\mathbf{x}$ i $\mathbf{z}$ je:

$$\begin{aligned}\kappa(\mathbf{x},\mathbf{z}) &= \boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{z})\\ &= [\boldsymbol{\phi}_1(\mathbf{x}),\boldsymbol{\phi}_2(\mathbf{x})]^{\mathrm{T}}[\boldsymbol{\phi}_1(\mathbf{z}),\boldsymbol{\phi}_2(\mathbf{z})]\\ &= \boldsymbol{\phi}_1(\mathbf{x})^{\mathrm{T}}\boldsymbol{\phi}_1(\mathbf{z})+\boldsymbol{\phi}_2(\mathbf{x})^{\mathrm{T}}\boldsymbol{\phi}_2(\mathbf{z})\\ &= \kappa_1(\mathbf{x},\mathbf{z})+\kappa_2(\mathbf{x},\mathbf{z})\end{aligned}$$

18 Kako izvesti ujezgrenu inačicu L2-regularizirane linearne možete pročitati u (Murphy, 2012) (poglavlje 14.4.3).

Literatura

F. Almeida and G. Xexéo. Word embeddings: A survey. *arXiv preprint arXiv:1901.09069*, 2019.

C. M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

T. Gärtner. A survey of kernels for structured data. *ACM SIGKDD Explorations Newsletter*, 5 (1):49–58, 2003.

T. Hofmann, B. Schölkopf, and A. J. Smola. Kernel methods in machine learning. *The annals of statistics*, pages 1171–1220, 2008.

N. M. Kriege, F. D. Johansson, and C. Morris. A survey on graph kernels. *Applied Network Science*, 5(1):1–42, 2020.

K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

B. Scholkopf and A. J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. Adaptive Computation and Machine Learning series, 2018.

J. Shawe-Taylor, N. Cristianini, et al. *Kernel methods for pattern analysis*. Cambridge university press, 2004.

C. F. Van Loan and G. H. Golub. *Matrix computations*. Johns Hopkins University Press Baltimore, 1983.