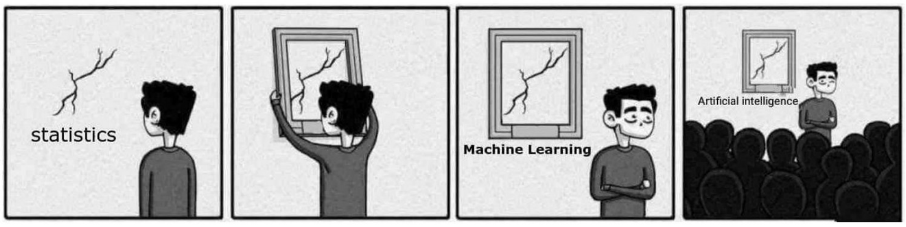
Naiv Bayes alapú spamszűrés

Feladat: Írjunk egy spamszűrő algoritmust, mely tetszőleges szöveges e-mailről eldönti, hogy spam vagy sem. A feladat megoldásához rendelkezésünkre áll egy e-mailekből álló adathalmaz, mely spam és nem spam e-maileket egyaránt tartalmaz. A feladatot valósítsuk meg Naiv Bayes osztályozó algoritmus segítségével.

1. Bevezető és egy pici terminológia

- A fenti feladat a **gépi tanulás** (**machine learning**, röviden **ML**) témaköréhez tartozik. A gépi tanulási módszerek alapvető jellemzője, hogy rendelkezésükre áll egy bizonyos mennyiségű adat/adathalmaz, melyet elemezve az adatokból információt nyernek ki, és ezeket felhasználva következtetéseket tesznek lehetővé. Így az ML gyökerei a statisztikai módszerekben találhatók, a „tanulás" szó pedig valójában azt az elvet fedi, miszerint a gép a kapott adatokból hasznos információkat tudhat meg, és minél több adat áll a rendelkezésére, a kinyert infók annál relevánsabbak statisztikailag.



- A feladat a **felügyelt tanulás** (**supervised learning**) kategóriájába tartozik, hiszen a rendelkezésünkre álló adathalmazban az e-mailek fel vannak **címkézve** (**labeled data**), azaz mindegyik e-mailről tudjuk, hogy spam vagy sem.
- A feladat tehát egy **osztályozási probléma**, melynek során egy e-mailt be kell sorolni a *SPAM* vagy *NEM SPAM*[1] kategóriák valamelyikébe. Mivel itt csak 2 kategória van, a feladatot **bináris osztályozási problémának** nevezik.
- Mivel szöveges állományokat kell osztályozni, a probléma megoldására egy klasszikus módszer a **naiv Bayes-osztályozó algoritmus**. Ez egy egyszerű statisztikai módszer, mely a **Bayes-tétel**en alapszik, és az e-mailekben megjelenő szavak alapján ki tudja számolni annak a valószínűségét, hogy egy adott e-mail *SPAM* (vagy *HAM*). Egy e-mail **jellemzőit**, **tulajdonságait** (angolul **feature**) tehát a mailben szereplő szavak adják meg.
- A **naiv** elnevezés abból adódik, hogy a modell megalkotása során feltételezzük, hogy az e-mail jellemzői, azaz az e-maileket alkotó szavak egymástól teljesen függetlenek[2]. Ez egy nagyon nyers

[1] A továbbiakban a *NEM SPAM* maileket *HAM*-nek fogjuk nevezni. Hogy miért fogjuk sonkának hívni őket? Ezért: `https://www.youtube.com/watch?v=Syr-oNr4IUQ`

[2] Ezt az elvet **bag of words** modellként is szokták emlegetni, ahol egy szöveget kizárólag a benne megjelenő szavak számossága (**hisztogram**ja) alapján jellemzünk, elhanyagolva a szöveg struktúráját vagy a szavak sorrendiségét.

egyszerűsítés, hiszen sok esetben az egymás melletti szavak között kapcsolat van, és a szavak sorrendisége is fontos. Ennek ellenére, a naiv Bayesen alapuló spamszűrő algoritmus egy jól bevált módszer, mely akár személyre is szabható, és megfelelően alacsony **false positive** aránnyal rendelkezik[3].

2. Naiv Bayes alapú spamszűrő

Rettentő leegyszerűsítve, a módszer alapgondolata az, hogy bizonyos szavak (például „FREE", „MILLION", „CLICK") megjelenése egy adott e-mailben arra enged következtetni, hogy az adott mail nagyobb valószínűséggel *SPAM* (vagy hasonló módon, *HAM*).

Hogy formalizálni tudjuk a dolgokat, vezessük be a következő jelöléseket:

- jelölje $\mathbf{d}_i$ az i-edik dokumentumot, és y_i ennek az állománynak a címkéjét (azaz *SPAM* vagy *HAM*).
- ekkor $\mathcal{D} = \{(\mathbf{d}_i, y_i) \mid i = 1, \ldots, \ell\}$ az adathalmaz, mely a rendelkezésünkre álló összes felcímkézett e-mailt jelöli.
- minden dokumentumot úgy fogunk fel, mint egymástól független szavak sorozatát (**bag of words**), vagyis az üzeneteket a következő módon jellemezhetjük:

  $$\mathbf{d}_i = \{(w_k, \operatorname{card}(w_k, \mathbf{d}_i)) \mid k = 1, \ldots, m\},$$

  ahol a dokumentumok szavait w_k szimbólummal jelöljük, és mindegyik szónak tudjuk az előfordulási számát egy-egy dokumentumon belül.

Ekkor egy adott $\mathbf{d}$ állomány esetén a következőket akarjuk kiszámítani/megbecsülni:

$$P(HAM|\mathbf{d}) \quad \text{és} \quad P(SPAM|\mathbf{d}),$$

azaz meg akarjuk határozni, mekkora a valószínűsége annak, hogy az adott mail *HAM* vagy *SPAM*.

Tekintsük például a $P(SPAM|\mathbf{d})$ valószínűséget. Ez a Bayes-tétel értelmében felírható úgy, mint

$$P(SPAM|\mathbf{d}) = \frac{P(\mathbf{d}|SPAM) \cdot P(SPAM)}{P(\mathbf{d})}, \tag{1}$$

ahol

- $P(SPAM)$ annak a valószínűsége, hogy az adathalmazból egy tetszőlegesen választott mail *SPAM*;
- $P(\mathbf{d})$ annak a valószínűsége, hogy egy tetszőleges dokumentum $\mathbf{d}$ alakú;
- $P(\mathbf{d}|SPAM)$ pedig egy feltételes valószínűség, ami megmondja, hogy egy *SPAM* e-mail mekkora eséllyel néz ki úgy, ahogy az illető $\mathbf{d}$ dokumentum. Azt pedig, hogy az adott dokumentum „hogy néz ki", az illető állomány jellemvonásai (feature-jei) adják meg, melyeket a bag of words elv értelmében a dokumentum szavai egyértelműen meghatároznak. Azaz szükségünk van a $w_k \in \mathbf{d}$ szavakra és ezek előfordulásának számára az illető mailben, ami $\operatorname{card}(w_k, \mathbf{d})$;

Egy adott w_k szó esetén jelölje $P(w_k|SPAM)$ annak a valószínűségét, hogy egy *SPAM* e-mail esetén mekkora eséllyel találkozunk ezzel a bizonyos w_k szóval. Most a modell „naivitását" felhasználva feltételezhetjük, hogy a dokumentumban megjelenő minden szó egymástól teljesen független feature, azaz:

$$P(\mathbf{d}|SPAM) = \prod_{w_k \in \mathbf{d}} P(w_k|SPAM)^{\operatorname{card}(w_k, \mathbf{d})}, \tag{2}$$

ami pontosan azt jelenti, hogy figyelembe vesszük, hogy az illető dokumentum szavai mekkora valószínűséggel jelenhetnek meg *SPAM* típusú üzenetekben (A hatványozás azért jelenik meg a fenti képletben, mert egy adott $w_k \in \mathbf{d}$ szót lehet, hogy többször is figyelembe kell vegyünk, igazából pontosan annyiszor, ahányszor a dokumentumban megjelenik.)

Tehát annak a valószínűsége, hogy az adott $\mathbf{d}$ dokumentum *SPAM*, a következő:

$$P(SPAM|\mathbf{d}) = \frac{P(SPAM)}{P(\mathbf{d})} \cdot \prod_{w_k \in \mathbf{d}} P(w_k|SPAM)^{\operatorname{card}(w_k, \mathbf{d})}. \tag{3}$$

[3] „all models are wrong, but some are useful." (George Box statisztikus)

2.1. Bináris osztályozás

Bináris osztályozás esetén (amilyen a tartalom alapú spamszűrés is) elegendő, ha az

$$R := \frac{P(SPAM|\mathbf{d})}{P(HAM|\mathbf{d})} = \frac{P(SPAM)}{P(HAM)} \cdot \prod_{w_k \in \mathbf{d}} \left(\frac{P(w_k|SPAM)}{P(w_k|HAM)} \right)^{\mathrm{card}(w_k,\mathbf{d})} \tag{4}$$

arányt számoljuk ki, ezáltal megszabadulhatunk a $P(\mathbf{d})$ nevező meghatározásától. Így, ha ez az arány 1-nél nagyobb, a dokumentum 50%-nál nagyobb valószínűséggel *SPAM*, ellenkező esetben nagyobb valószínűséggel *HAM*.

Ugyanakkor, tudva, hogy a két kategória teljes eseményrendszert alkot (vagyis egy adott dokumentum vagy *SPAM*, vagy *HAM*), kapjuk, hogy

$$P(SPAM|\mathbf{d}) + P(HAM|\mathbf{d}) = 1, \quad \text{azaz} \quad R = \frac{P(SPAM|\mathbf{d})}{1 - P(SPAM|\mathbf{d})}, \tag{5}$$

és innen pontosan meghatározhatjuk rendre a $P(SPAM|\mathbf{d})$ és $P(HAM|\mathbf{d})$ valószínűségeket az R arány függvényében:

$$P(SPAM|\mathbf{d}) = \frac{R}{R+1} \quad \text{és} \quad P(HAM|\mathbf{d}) = \frac{1}{R+1}. \tag{6}$$

A munka nehezén igazából túl vagyunk, mert a (4) képlet jobb oldalán megjelenő értékeket mind meg tudjuk becsülni egy adott bemeneti adathalmaz alapján.

2.2. A paraméterek becslése

Jelölje C a két lehetséges kategória egyikét ($C \in \{SPAM, HAM\}$). A modell paramétereit a rendelkezésünkre álló adatok alapján a következőképpen tudjuk megbecsülni:

$$P(w_k|C) = \frac{\mathrm{card}(w_k, C)}{\sum_w \mathrm{card}(w, C)} \tag{7}$$

$$P(C) = \frac{C \text{ osztályú e-mailek száma}}{\text{dokumentumok száma}}, \tag{8}$$

ahol $\mathrm{card}(w_k, C)$ azt adja meg, hogy összesen hányszor fordult elő az illető w_k szó a C típusú dokumentumokban, $\sum_w \mathrm{card}(w, C)$ pedig a C kategóriájú dokumentumok össz szószámát jelöli.

3. Még pár tipp, hogy ez gyakorlatban is működjön...

Az első probléma, amit észrevehetünk az, hogy lehet, hogy létezik olyan szó, ami előfordul *SPAM* e-mailekben, viszont egyetlen *HAM* e-mailben sem jelenik meg. Ekkor a $P(w_k|HAM)$ valószínűségnek a paraméterek becslése során 0-át kapnánk (lásd (7) képlet), ez viszont a (4) összefüggésben egy nullával való osztást eredményezne. Hasonló módon, az sem egészséges, ha a számlálóban jelenik meg a nullás (csak amiért egy szó nem szerepelt a *SPAM* adatok között, nem kellene, hogy a végső arány értékét lenullázza). Erre a legegyszerűbb megoldás az, hogy mindegyik ilyen valószínűségi becslésre kiszabunk egy alsó korlátot, például $\lambda = 0.00000001$-et. Ekkor, ha $P(w_k|C) < \lambda$, akkor a $P(w_k|C)$ új értéke legyen $P(w_k|C) := \lambda$.

A másik probléma az alulcsordulásból adódhat. Valószínűségeket nem jó dolog szorozni, mivel nagyon kicsi szám lehet a végeredmény, amit nem tudunk megfelelő pontossággal ábrázolni. Ezért, ha lehetséges, jobb összegzést használni. Ezt megtehetjük úgy, hogy alkalmazunk egy logaritmus-függvényt (bármilyen $a > 1$ alapú logaritmus megteszi, hiszen ez esetben $\log_a$ monoton, szigorúan növekvő lesz). Mi itt most e alapú logaritmust fogunk használni. Ekkor a (4) képlet így alakul:

$$\ln R = \ln(P(SPAM)) - \ln(P(HAM)) + \sum_{w_k \in \mathbf{d}} \mathrm{card}(w_k, \mathbf{d}) \cdot \left[\ln(P(w_k|SPAM)) - \ln(P(w_k|HAM)) \right] \tag{9}$$

Ahogy korábban említettük, ha R értéke 1-nél nagyobb, a dokumentum valószínűleg *SPAM*, ellenkező esetben *HAM*. Így a fent meghatározott $L = \ln R$ kifejezés előjele fogja megadni a prediktált osztályt: ha L pozitív szám, a prediktált címke *SPAM*, ellenkező esetben *HAM*.

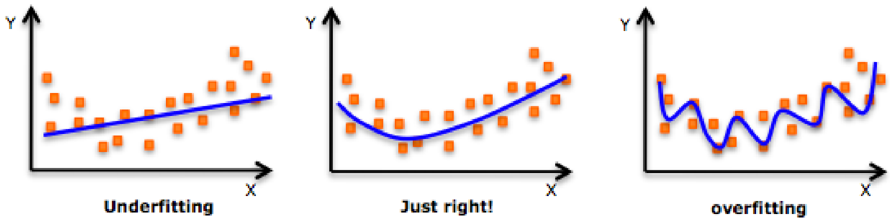
4. Tanulási és teszt adatok

Gépi tanulás során egy gyakran alkalmazott eljárás a rendelkezésre álló adatokat felbontani tanulási- és teszt-adathalmazokra. Az általunk kiválasztott modellt a tanulási adathalmazon tanítjuk, majd a teljesítményét a teszt adathalmazon (és néha a tanulási halmazon is) leellenőrizzük . A tanítási és teszt adathalmazokra való bontással próbáljuk megakadályozni az **overfitting** és **underfitting** jelenségeket, melyek minden ismert statisztikai modellt érintenek, ezek között a naiv Bayes-t is.

Amikor a modellünk nem illeszkedik helyesen az adatokra, vagyis egyáltalán nem vagy csak kis mértékben ismerte fel a különböző jellemvonásokat, tulajdonságokat, akkor a modell alultanultságáról beszélünk. Ez a jelenség az **underfitting**. Ilyenkor nem sikerült a modellnek azonosítani a tanulási adatok jellemző tulajdonságait, aminek következtében az új, eddig nem látott bemeneteket sem tudja majd helyesen osztályozni. Az underfittingnek több oka is lehet: tanulási adatok gyenge minősége vagy kis mennyisége, nem megfelelő modell használata (például lineáris modellel probálunk egy nem lineáris feladatot megoldani), stb.

Az underfitting ellentéte az **overfitting**: ez a kifejezés a modellünk túltanultságát jelzi. Egy lehetséges hibaforrás, hogy a modell túlságosan ráilleszkedik a tanulási adathalmaz sajátos jellemvonásaira, és emiatt nem tud a későbbiekben megfelelően általánosítani. Ilyenkor a modell elvesz a részletekben, gyakran az adatokban levő zajt vagy kiugró sajátosságokat is jellemvonásnak titulálja, és emiatt nem tud megfelelően általános következtetéseket levonni az adathalmazról. Ezért az a tény, hogy magas teljesítményt érünk el a tanulási adatokon, nem vonja maga után azt, hogy a tesztadatokon (vagy más, eddig még nem látott bemenetek esetén) is jól fog teljesíteni az algoritmusunk.

A bemeneti adathalmaz felosztásakor ügyelnünk kell arra, hogy a tanulási és teszt halmazban az adatok eloszlása ugyanolyan legyen, mint az eredeti adathalmazban. Nem szeretnénk ugyanis azt, hogy az egyik halmazban szinte csak az egyik osztály tagjai szerepeljenek, míg a másik halmazban alig jelenjenek meg. Emiatt az egyes bemenetekről véletlenszerűen döntjük el, hogy a továbbiakban teszt vagy tanulási egyed legyen.



A fentiek értelmében a modellünk helyességének ellenőrzésére kétfajta hibát számolhatunk ki: tanulási hibát és teszt hibát.

- A **tanulási hiba** kiszámítása során a modellt a tanulási adatokon tanítjuk be, majd a kapott modellel újra felcímkézzük a tanulási adatokat, és ezeket a címkéket összehasonlítva a valós címkékkel kiszámítjuk, mennyire jól teljesít az algoritmus. Például, ha a kapott tanulási hiba nagy, **underfitting**-ről beszélünk.
- A **teszt hiba** kiszámítása során a modellt a tanulási adatokon tanítjuk be, és a teszt adatokon ellenőrizzük az algoritmus helyességét. Így például, ha a tanulási hiba kicsi, viszont a teszt hiba nagy, akkor ez **overfitting**-re enged következtetni.

5. Additív simítás (Additive/Laplace/Lidstone smoothing)

Az additív simítás lényege, hogy zérónál nagyobb valószínűségeket rendeljünk a tanulási halmazban nem látott szavakhoz, így jobb becslést kapva. Additív simítás esetén a paraméterekre vonatkozó (7) képlet helyett a következő kifejezést alkalmazzuk:

$$P(w_k|C) = \frac{\text{card}(w_k, C) + \alpha}{\alpha|V| + \sum_w \text{card}(w, C)}, \tag{10}$$

ahol $\alpha \in (0, 1]$ egy rögzített paraméter, $|V|$ pedig a tanulási adatok szótárának mérete (= az összes különböző szó száma, amely megjelent a tanulási adatokban - vigyázat, itt a multiplicitás nem számít).

A fent megjelenő α számot a modell hiperparaméterének is nevezik. Ellentétben a $P(w_k|C)$ paraméterekkel, ezt az értéket még a tanulási folyamat kezdete előtt lerögzítjük. Persze felmerül a kérdés, hogy milyen α értéket válasszunk? Az optimális paraméter meghatározására egy lehetséges módszer a **kereszt-validálás** (**cross validation**).

6. K-szoros kereszt-validálás

A K-szoros kereszt-validálás egy olyan újramintavételezési eljárás, amelyet a gépi tanulási algoritmusok használnak a modell optimális hiperparamétereinek becslésére, vagy a módszer kiértékelésére azokban az esetekben, amikor korlátos a rendelkezésre álló adatmennyiség.

A módszer lényege, hogy a rendelkezésre álló adathalmazt felosztjuk K darab megközelítőleg egyforma méretű diszjunkt részhalmazra, majd sorban kiválasztjuk a részhalmazokat, mint validációs halmazt (teszt halmazt), és a megmaradt $K - 1$ halmaz egyesítése lesz a tanulási halmaz. Az így kialakult tanulási halmazok mindegyikén sorra betanítjuk a modellünket, és a megfelelő validálási halmazokon sorra kiértékeljük a modell teljesítményét. Végül, a kiértékelések eredményeit összesítjük (lásd ALG 1 algoritmus).

ALG 1 K-szoros kereszt-validálás.

Véletlenszerűen keverd össze az adathalmazt
Bontsd K darab megközelítőleg egyforma méretű diszjunkt halmazra
for $\mathbf{h_i} \in$ diszjunkt részhalmazok **do**
    Válaszd ki a $\mathbf{h_i}$ halmazt, mint validációs halmaz
    Összesítsd a megmaradt $K - 1$ halmazt, mint tanulási halmazt
    Tanítsd a modellt a tanulási halmazon
    Értékeld ki a frissen tanított modellt a $\mathbf{h_i}$ validációs halmazon
    $\mathbf{S_i} \leftarrow$ a modell teljesítménye
end for
Számítsd ki a modell összesített teljesítményét: $\frac{1}{K} \sum_{i=1}^{K} \mathbf{S_i}$

A kereszt-validáció azért is előnyös, mert az eredeti adathalmaz egyenletes mintavételezésére ad lehetőséget, ennek következtében pedig egy robusztusabb kiértékelést biztosít.

Abban az esetben, ha a kereszt-validálást a modell hiperparamétereinek meghatározására alkalmazzuk, a kereszt-validált hibákat ki kell számítanunk különböző paraméterek esetén, majd azt a paramétert választjuk, melyre a kapott hiba minimális.

7. Félig felügyelt tanulás naiv Bayes-szel

A félig felügyelt tanulás alapötlete az, hogy ha vannak címke nélküli adataink (amiből általában több van, mint címkézettekből, hiszen könnyebben beszerezhetőek), akkor használjuk fel ezeket is, javítva ezáltal a predikciókat. A kérdés az, hogy hogyan is tudjuk ezeket felhasználni?

Naiv Bayes esetén az egyik alkalmazható módszer a következő: tanítsuk be (azaz határozzuk meg a paramétereket) a címkézett tanulási adatok alapján, majd határozzuk meg a címkézetlen tanulási adatok osztályait. Ha eléggé biztos a döntés (azaz $P_{\text{nagyobb}}/P_{\text{kisebb}} \geq \theta$, ahol θ egy általunk rögzített paraméter), akkor adjuk az illető dokumentumot a prediktált címkéjével együtt a tanulási adathalmazhoz. Ezután pedig tanítsuk újra a naiv Bayes osztályozót az új, kibővült tanulási halmaz alapján. Így a címkézett tanulási adatok halmaza nő, és a predikcióink pontosabbak lesznek.

Az eljárást addig folytatjuk, amíg a paraméterek értékeinél nem észlelünk több változást (azaz nem találunk több olyan adatot, mely biztosan felcímkézhető). A pszeudokód az ALG 2 algoritmusban látható.

ALG 2 Félig felügyelt naiv Bayes.

$\mathcal{D}_0$ = címkézett tanulási adatok
$\mathcal{D}_1$ = címkézetlen tanulási adatok
while nem változnak a paraméterek **do**
    Tanítsuk be, azaz számoljuk ki a naiv Bayes paramétereit $\mathcal{D}_0$ alapján.
    $\mathcal{D}_2 = \emptyset$
    for $\mathbf{d} \in \mathcal{D}_1$ **do**
        if $P_{\text{nagyobb}}(\mathbf{d})/P_{\text{kisebb}}(\mathbf{d}) \geq \theta$ **then**
            $\mathcal{D}_2 = \mathcal{D}_2 \cup \{(\mathbf{d}, \text{prediktált címke})\}$
        end if
    end for
    $\mathcal{D}_0 = \mathcal{D}_0 \cup \mathcal{D}_2$
    $\mathcal{D}_1 = \mathcal{D}_1 \setminus \mathcal{D}_2$
end while

8. Útmutatás a laborfeladat implementálásához lépésekben

1. Adatok előfeldolgozása:
   Minden e-mail esetén:

   - felosztjuk az e-mailt tokenek (írásjelek és szavak) sorozatára (split), majd kisbetűsítünk minden szót;
   - a tokenek halmazából kiszűrünk minden felesleges információt, például az írásjeleket (",", ".", ":", stb.), a "subject:" címkét az üzenet legelejéről, illetve a stopszavakat (ilyenek a kötőszavak, névelők, névmások, stb. - ehhez használjuk a stopwords.txt és stopwords2.txt segédállományokat);
   - a megmaradt szavak alapján elkészítjük a dokumentum hisztogramját

     $$\mathbf{d} = \{ \text{ szó : szó előfordulásának száma az illető dokumentumban } \}$$

     alakban;

2. A naiv Bayes alapú modell betanítása:

   - a tanulási adatok száma alapján meghatározzuk a $P(SPAM)$ és $P(HAM)$ priori valószínűségeket (lásd (8) képlet);
   - a feldolgozott tanulási adatok alapján létrehozzuk a tanulási üzenetekben megjelenő szavak szótárát (V);
   - a (7) (additív simítás esetén pedig a (10)) képletet és a tanulási halmaz e-mailjeinek hisztogramjait felhasználva meghatározzuk a modell paramétereit, ekkor a tanulási adatok szótára

     $$V = \{ \text{ szó : } P(\text{ szó } | SPAM), P(\text{ szó } | HAM) \}$$

     alakú lesz.

3. A modell tesztelése:

   - Tanulási hiba kiszámítása: a tanulási adatokon betanított modellt felhasználva osztályozunk minden **tanulási adatot** ((9) képlet), majd hibaszázalékot számolunk, összehasonlítva az adatok valós címkéit az általunk becsült címkékkel.
   - Teszt hiba kiszámítása: a tesztadatok előfeldolgozása után, a tanulási adatokon betanított modellt felhasználva osztályozunk minden **tesztadatot** ((9) képlet), majd hibaszázalékot számolunk, összehasonlítva az adatok valós címkéit az általunk becsült címkékkel.

9. Annak, aki egy picit többet szeretne tudni :)

9.1. Naiv Bayes klasszifikáló algoritmus több osztály esetén

A naiv Bayes alapú klasszifikáló módszer tetszőleges számú osztály esetén alkalmazható. Ekkor egy adat címkéjét a

$$C^* = \arg\max_i P(C_i|\mathbf{d}) \tag{11}$$

képlettel határozhatjuk meg, ahol C_i végigfut az összes lehetséges kategórián (azaz az adatot azzal a címkével látjuk el, melynek valószínűsége a legnagyobb).

Ez a Bayes-tétel értelmében felírható úgy, mint

$$C^* = \arg\max_i \frac{P(\mathbf{d}|C_i)P(C_i)}{P(\mathbf{d})}, \tag{12}$$

ahonnan – a maximum függvény miatt – a nevező ismét elhagyható, mivel rögzített dokumentum esetén a $P(\mathbf{d})$ érték ugyanaz lesz minden osztályra. Tehát a következő kifejezéssel dolgozhatunk tovább:

$$C^* = \arg\max_i P(\mathbf{d}|C_i)P(C_i). \tag{13}$$

A fenti képletben megjelenő paramétereket a tanulási adatok és az illető **d** adat jellemzőit felhasználva becsüljük meg úgy, ahogy azt a 2. részben leírtuk.

9.2. Bayes tétele kicsit másképp

Bayes tételének legegyszerűbb változatát már mindenki kívülről tudja: ha adott az A és B esemény, akkor

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}.$$

De nézzük meg, pontosan hogyan is használjuk ezt mi. Valójában az A esemény egy hipotézisnek, egy ismeretlen állapotnak felel meg (pl. „a dokumentum egy *SPAM*”), aminek a valószínűségére vagyunk kíváncsiak úgy, hogy rendelkezésünkre áll a B megfigyelhető esemény, mely kapcsolatban áll az A eseménnyel (pl. „a dokumentum tartalmazza a MILLION szót”):

$$P(\text{hipotézis}|\text{megfigyelés}) = \frac{P(\text{megfigyelés}|\text{hipotézis}) \cdot P(\text{hipotézis})}{P(\text{megfigyelés})}. \tag{14}$$

Ekkor ez az összefüggés valójában azt adja meg, hogyan növeli vagy csökkenti a megfigyelt esemény a hipotézis valószínűségét.

- A $P(\text{hipotézis})$ értéket az esemény **priori valószínűség**ének nevezik. Például: tegyük fel, hogy adott egy 10000 e-mailből álló tanulási adathalmaz, melyben 7320 üzenet *SPAM*, 2680 *HAM*. Ekkor megírhatom 2 sorban a világ legegyszerűbb spamszűrőjét, ami 73.2%-os valószínűséggel fog egy mailt *SPAM*-nek minősíteni. Ha a mintavételem elég nagy, ez nem is fog olyan rosszul működni. Ekkor a **priori valószínűség** $P(SPAM) = 73.2\%$ (megfigyelés előtti valószínűség).
- Most tegyük fel, hogy javítani szeretném az előző spamszűrőm, úgyhogy meg is nyitom az üzenetet, mielőtt 73.2%-os valószínűséggel *SPAM*-nek nyilvánítanám. Amint az üzenetet megfigyeltem, plusz információ birtokába jutok, ami befolyásolni fogja a korábbi 73.2%-os becslésemet. A $P(\text{hipotézis}|\text{megfigyelés})$ értéket emiatt **posteriori valószínűség**nek nevezik (azaz megfigyelés utáni valószínűség).
- a $P(\text{megfigyelés}|\text{hipotézis})$ számot **likelihood**nak nevezzük, ami kifejezi, mekkora eséllyel észlelhetjük az adott megfigyelést abban az esetben, ha a hipotézis fennáll.
- $P(\text{megfigyelés})$ a normalizációs tag, mely biztosítja, hogy a kapott poszteriori becslés egy tényleges valószínűség, azaz egy $[0, 1]$ közötti szám legyen. A $P(\text{megfigyelés})$ értéket általában nehéz meghatározni. Szerencsére, az alkalmazások során erre az értékre az esetek nagy részében nincs szükség,

hiszen minket csak a $P(\text{hipotézis}|\text{megfigyelés})$ **posteriori valószínűség** maximuma érdekel, így a nevezőt elhanyagolhatjuk. Figyeld meg: ezért volt az, hogy a bináris osztályozás esetén a (4) képletben a $\frac{P(SPAM|\mathbf{d})}{P(HAM|\mathbf{d})}$ törtet számítottuk ki, és több osztály esetén pedig elegendő volt a (12) képlet helyett a (13) összefüggést használni.

- A fentiek alapján Bayes-tételét gyakran a

$$\textbf{posterior} \propto \textbf{likelihood} \cdot \textbf{prior}$$

egyszerűsített formában írják fel, ahol a $\propto$ szimbólum egyenesen arányosságot jelent.

A Bayes-tétel alkalmazását, és azt, hogy miként befolyásolja a megfigyelésünk a priori becslésünket, nagyon jól szemlélteti a híres **Monty Hall paradoxon**:
`https://www.youtube.com/watch?v=ugbWqWCcxrg`