TiMBL – Tilburg Memory Based Learner

Zbyněk Matoušek

4. května 2000

Abstrakt

Timbl je software založený na algoritmech umělé inteligence. Hlavním algoritmem klasifikace je metoda K nejbližších sousedů. Jsou zde popsány možnosti nastavení metrik a algoritmů.

Další částí je popis experimentů s leteckými snímky, kde bylo úkolem rozhodnout, jaký druh povrchu je zachycen na jednotlivých pixelech. Pomocí Timblu bylo dosaženo přesnosti až 94 procent.

Obsah

1 Učení z instancí

Timbl je zkratka anglického názvu Tilburg Memory Based Learner[1]. Jedná se o software sloužící k získávání znalostí pomocí technik strojového učení. Tento konkrétní program je založen na tzv. Memory Based Learning metodě[2]. Ta spočívá v tom, že se systém z učící množiny, která je správně oklasifikovaná, zapamatuje jednotlivé instance. Každá další instance je oklasifikována na základě podobnosti s již známými příklady z učící množiny.

Memmory Based Learning má dvě základní části: *Učící složku*, která je celá uložena v paměti a *výkonnou složku*, která je založená na výpočtu podobnosti.

Učící složka jen ukládá instance, které jsou na vstupu do paměti. Někdy se tomuto přístupu také říká líný (lazy), neboť nijak neupravuje vstup (neprovádí žádné restrukturalizace). Instance je složena z vektoru n atributů a klasifikace.

Ve výkonné složce je používán výsledek učení k oklasifikování neznámých dat. Nový vektor X je zařazen do systému a jeho vzdálenost od prvku Y je spočítána pomocí tzv. *distance metric* (metriky vzdálenosti) $\Delta(X, Y)$. Jeho klasifikace se děje na základě nejčastějšího ohodnocení k nejbližších (vzhledem k Δ) sousedů.

Timbl má v sobě zabudováno několik přístupů jak počítat „podobnost“ a také jakým způsobem reprezentovat jednotlivé instance v paměti.

Timbl byl vytvořen především pro potřeby zpracování přirozeného jazyka. V této oblasti se pracuje s velkými objemy dat, takže byl mimo jiné kladen velký důraz na rychlost a nízkou paměťovou náročnost.

2 Instalace a ovládání

Program je k dispozici na adrese „http://ilk.kub.nl“. Každý uživatel se musí zaregistrovat (vyplnit formulář na webu), ale jinak je tento software pro vědecké účely volně šiřitelný. O podmínkách používání a šíření programu podstatně podrobněji hovoří licenční podmínky, které (v anglickém originále) najdete také na výše uvedené adrese.

Software je dodáván jako balík zdrojových textů s malým souborem zkušebních dat v podobě balíku *.tar.gz*. Na Unix platformách by neměly být s instalací problémy. Autoři tvrdí, že Timbl byl úspěšně zkompilován pro:

- počítače SPARC s SUN-OS (4.1.4) a Solaris (2.6)
- Intel platformy, běžící na Solaris (2.6 and 2.7)
- Intel platformy běžící na Linux (2.0)
- HP počítačích s HP-UX (10.20)

Po rozbalení archivu (`gunzip Timbl.tar.gz && tar xf Timbl.2.0.tar`) stačí zadat jen *make* a kompilace by měla proběhnout bez problémů. Jedna z voleb příkazu `make` umožňuje hned otestovat, zda vše dobře funguje.

Na školní počítačové síti jsem Timbl ve verzi 2.0 nainstaloval do modulu Timbl2.0. Tato verze je k dispozici na všech Unix platformách používaných u nás na fakultě (tj. Irix, SunOS, Linux).

V této zprávě budou popisovány výsledky, které jsem dosáhl s verzí 2.0. V současné době je k dispozici již verze 3.0, která je trochu rozšířená a liší se v syntaxi zadávaní přepínačů. Rozdílům se budu podrobněji věnovat v 5.

3 Parametry programu Timbl

3.1 Základní nastavení

Program *Timbl* se spouští z příkazové řádky, kde je mu možno předat mnoho parametrů. Pokud spustíte *Timbl* bez parametrů, vypíší se vám ve stručnosti nejpoužívanější přepínače i s jejich krátkým popisem:

> TiMBL Version 2.0 (c) ILK 1998, 1999. Tilburg Memory Based Learner Induction of Linguistic Knowledge Research Group, Tilburg University Tue Mar 21 23:43:35 2000
>
> usage: Timbl -f data-file -t test-file or see: Timbl -h for all possible options

Pokud se tedy spokojíte s implicitně nastavenými hodnotami stačí vám znát význam jen těchto přepínačů:
`-f <datový_soubor>`: zadáte jméno souboru s učícími daty

`-t <testovací_soubor>`: jméno souboru ve kterém se nachází testovací data
`-h`: vypíše krátký popis možných přepínačů
Pokud není zadán testovací soubor Timbl prevede jen první dvě fáze výpočtu:

1. Vyzkouší trénovací data a vypočítá na nich statistické charakteristiky (vypočítá váhy jednotlivých atributů, pořadí důležitosti, rozptyl atd.)
2. Příklady z učící množiny jsou uloženy do paměti.

Zde celý algoritmus skončí. Pokud jsou definována testovací data (přepínač `-t`), tak na ně bude aplikován zvolený algoritmus klasifikace a výsledek bude zapsán do souboru. Název tohoto souboru je složen z názvu testovacího a z parametrů zadaných při spuštění Timblu. Zde je jedna z odlišností mezi verzemi 2.0 a 3.0, kde je možné si jméno výstupního souboru zvolit. Ve výstupním souboru jsou zopakována veškerá vstupní data (i s jejich správnou klasifikací) a je zde doplněn jeden atribut, což je klasifikace navržená programem Timbl. Nakonec Timbl ještě vypíše úspěšnost, to jest porovnání shody jeho výsledků s tím, co se nacházelo v testovacím souboru.

Pokud implicitní nastavení programu není vyhovující, je možné tato nastavení změnit.

3.2 Volba algoritmů a metrik

Program používá tři základní algoritmy klasifikace. Ty se liší jednak rychlostí a také filosofií přístupu k problému. Více v kapitole 4. Možná nastavení jsou:

- `-a <n>` – algoritmus
  - $n = 0$ – algoritmus IB1, anglicky nazývaný „k-nearest neighbor search“, tedy rozhodování podle k nejbližších sousedů.
  - $n = 1$ – IGTREE, jedná se o jistou verzi rozhodovacího stromu (podrobněji 4.4.1). Výhodou tohoto algoritmu je naproti IB1 především jeho rychlost a podstatně menší paměťová náročnost. Tato metoda je velice rychlá, ale nebere v úvahu vzájemné vzdálenosti instancí.
  - $n = 2$ – TRIBL, takto autoři nazvali třetí algoritmus, který je hybridem předcházejících dvou. Princip je ve stručnosti v tom, že prvních k (k lze zvolit) atributů je rozhodováno pomocí IGTREE a zbytek potom IB1. Tak se dosáhne kompromisu mezi rychlostí a přesností.
- `-m <n>` – výběr podobnostní metriky. Tento přepínač je možné použít jen spolu s algoritmem IB1 nebo TRIMBL.
  - $n = 0$ – V originále se tato metrika nazývá „Weighted Overlap Metric“. Způsob jejího výpočtu bude lépe vyjasněn v sekci 4.1. Základní princip je asi tento: Pokud jsou dvě hodnoty naprosto stejné, tak výsledek je 0, jinak je 1.
  - $n = 1$ – „Modified Value Difference Metric“ – česky by se to dalo nazvat jako modifikovaná metrika změny hodnoty. Více 4.3. Rozdíl mezi dvěma hodnotami atributu je rozdělen podle rozložení jeho podmíněné pravděpodobnosti. Rozdíly mezi všemi dvojicemi jsou spočítány již před testovací fází. Jedinou výjimkou je případ, kdy je hodnot velké množství.
- `-w <n>` – Váha jednotlivých atributů. Váhování je použito u algoritmů IGTREE a IB1. Jsou možné tyto hodnoty:
  - n=0 – Žádné váhy (No Weighting), jinak řečeno všechny atributy mají stejnou důležitost.
  - n=1 – Váha poměrného přírůstku (Gain Ratio). Implicitní nastavení.
  - n=2 – Váha přírůstku informace (Information Gain).
  - n=filename – Při tomto nastavení přečte Timbl nastavení vah jednotlivých atributů ze souboru. Každý řádek začíná číslem atributu a za ním následuje hodnota váhy jakou danému atributu chceme přiřadit. Pokud má být daný atribut ignorován, tak mu jako hodnotu dáme `Igmored`. Řádek komentáře začíná `#`.

- `-k <n>` – Počet sousedů braných v úvahu při klasifikaci. To je použitelné jen u IB1 a Trimblu. Normální nastavení je 1. Především u Mod. Value Diff. Metric je většinou velice užitečné toto číslo zvýšit. Je také dobré použít liché číslo, aby se předešlo uváznutí (dvě a více instancí májí stejnou vzdálenost).
- `-q <n>` – Tento přepínač má význam jen pro Trimbl. Udává atribut u kterého se přejde od IGTree k IB1. Základní nastavení je na 0.
- `-R <n>` – Řešení uváznutí pomocí generátoru náhodných čísel. Normálně je tato možnost vypnuta a uváznutí je řešeno pomocí priority atributů.
- `-t <@filename>` – Pokud název souboru za přepínačem „t“ začíná písmenem @, tak to znamená, že nastavení pro testovací fázi budou čtena ze souboru. V tomto souboru mohou být na každém jednotlivém řádku instrukce a příkazy z množiny:

  `-D -e -F -k -m -o -p -q -R -t -u -v -w -x -% --`

3.3 Vstupní nastavení

- `-F <format>` – Tímto říkáme Timblu v jakém formátu jsou testovací data. Jsou možné tyto hodnoty: `Compact`, `C4.5`, `ARFF`, `Columns`. Standardně se Timbl pokusí uhodnout formát z prvního řádku.

  Formát `Compact` znamená že jednotlivé atributy nejsou ve vstupním souboru odděleny. Musí tedy být všechny stejné délky (stejný počet znaků) a tato délka se zadává pomocí přepínače `-l`.

  V `C4.5` je oddělovačem čárka a u `Columns` libovolný „bílý“ znak. `ARFF` je formát používaný softwarem WEKA.
- `-s` <$k_1, k_n, l-m$> – Přeskoč atributy k_1 a k_n a také l až m. Za přepínačem `s` je zadán řetězec, kde jsou jednotlivé atributy odděleny čárkou. Na ně poté nebude brán zřetel, jak při trénování, tak ani při testování dat. Toto nastavení má stejný efekt jako nastavit váhu atributu na `Ignored`. Tento postup má, na rozdíl od nastavení váhy na nulu, tu výhodu, že se zabrání některým vedlejším efektům, které by mohly vzniknout, neboť atribut s nulovou váhou je stále používán v interní reprezentaci učícího algoritmu (např. při použití alg. IGTree).
- `-N <k, ..., l-m>` – Říká Timblu, že s atributy $k \ldots l-m$ má zacházet jako s číselnými atributy.
- `-l <k>` – Délka atributu. Tento přepínač má význam jen při vstupních datech ve formátu Compact.
- `-i <treefile>` – Přeskoč první dvě fáze výpočtu (učení a optimalizování) a místo toho použij již dříve vygenerovaný a uložený soubor `treefile`

- -u <mvd_matrix_file> – Použij místo vypočítané matice podobnosti obsah souboru mvd_matrix_file
- -P <path> – Udává cestu, kde se budou hledat vstupní data. Tento argument je ignorován v případě, že cesta ke vstupním datům již obsahuje informace o adresáři.

3.4 Vnitřní reprezentace

- -T <k> – Uspořádání atributů. Jiné nastavení má za následek jinou paměťovou náročnost.
  - n=1 – Použije se stejné pořadí jako v učící množině.
  - n=2 – Použije se Gain Ratio (procento přírůstku) pro seřazení atributů. Toto nastavení je implicitní pro IGTree
  - n=3 – Je použito Information Gain (přírůstek informace) pro seřazení atributů.
  - n=4 – Srovnání podle velikosti výrazu
$$\frac{1}{pocet\ ruznych\ hodnot\ atributu}$$
  - n=5 – Podle
$$\frac{Gain\ Ratio}{pocet\ ruznych\ hodnot\ atributu}$$
  - n=6 – Podle
$$\frac{Inf.\ Gain}{pocet\ ruznych\ hodnot\ atributu}$$
- -x – Vypni používání zkratek pro přesně se shodující instance v IB1. Normální nastavení je na ON (což většinou bývá také podstatně rychlejší), ale pokud $k > 1$, zkracování dává odlišné výsledky od čistého algoritmu nejbližšího souseda.
- -- – Vypni používání optimalizace. Toto má rozdílný efekt v závislosti na použité metrice.

4 Integrované algoritmy a metriky

Společným znakem všech algoritmů použitých v Timblu je to, že si uchovávají nějakou reprezentaci učící množiny přímo v paměti. Během rozhodování jsou nové případy klasifikovány pomocí sousedů (instance s nejmenší vzdáleností), kteří se vyskytli v učící množině. Hlavní rozdíly v algoritmech spočívají:

- v definici *vzdálenosti*
- ve způsobu uložení instancí v paměti (struktuře)
- ve způsobu prohledávání paměti (přesněji prohledávání učící množiny)

4.1 Přesahující metrika (Overlap Metric)

V anglickém názvosloví se tomu říká Overlap metric, ale také Manhattan metric, Hamming distance nebo L1 metric. Je vyjádřena těmito vzorci:

$$\Delta(X,Y) = \sum_{i=1}^{n} \delta(x_i, y_i)$$

kde pro číselné atributy se počítá jako

$$\delta(x_i, y_i) = \frac{x_i - y_i}{max_i - min_i}$$

a pro diskrétní

$$\delta(x_i, y_i) = \begin{cases} 0, \ pokud \ x_i = y_i \\ 1, \ pokud \ x_i \neq y_i \end{cases}$$

Vzdálenost $\Delta(x_i, y_i)$ je tedy pouhým součtem vzdáleností jednotlivých atributů. kNN s použitím této metriky se nazývá IB1. Implicitně je k nastaveno na 1.

V Timblu došlo k menším úpravám IB1. Především pokud je $k > 1$, Timbl použije klasifikaci nejčastěji se vyskytující v nejbližším okolí. Pokud dojde k tomu, že mají dvě klasifikace stejnou četnost, potom se vybere ta, která byla častěji zastoupena v učící množině. Poslední úprava je spíše syntaktického rázu. Hodnota k neznačí počet okolních příkladů, ale počet různých vzdáleností, které se budou zahrnovat do výpočtu. To je významné především u výčtových (nespojitých) atributů.

4.2 Váha podle přínosu informace (Information Gain)

Metrika $\delta(x, y)$ se počítá podle stejného vzorce jako pro Overlap metric, což znamená, že se pouze spočítá počet stejných a rozdílných hodnot mezi dvěma instancemi. Toto se dá použít především pokud nám chybí informace o důležitosti jednotlivých atributů.

Information Gain (IG) bere jeden atribut po druhém a snaží se zjistit jeho přínos k určení do třídy. IG na i-tém atributu je zjištěno tak, že se spočítá rozdíl v nepřesnosti mezi určením bez hodnoty i-tého atributu a s ní. Matematicky se to dá vyjádřit takto:

$$w_i = H(C) - \sum_{v \in V_i} P(v) \times H(C|v)$$

Kde C je množina možných klasifikací, V_i je množina hodnot pro i-tý atribut a $H(C) = -\sum_{c \in C} P(c) log_2 P(c)$ je nepřesnost určení do třídy. Pravděpodobnosti jsou odhadnuty z relativních četností v trénovací množině. Pokud se jedná o spojitý atribut, tak je použitý interval rozdělen na několik stejných podintervalů (std. nastavení je na 20). Dále jsou tyto intervaly používány stejně jako by to byly výčtové typy. Toto rozdělení se však nevztahuje na výpočet metriky vzdálenosti.

Metrika IG má sklony přeceňovat atributy s velkým počtem různých hodnot. Např. pokud si představíme data z nemocnice, kde u každého záznamu je číslo pacienta, tak právě tento atribut bude považován za dosti podstatný při určování klasifikace, ale přitom nám nenese žádnou zobecňující informaci. Toto se pokouší odstranit Gain Ratio Metric, což je IG dělená informací o rozložení. Počítá se takto:

$$w_i = \frac{H(C) - \sum_{v \in V_i} P(v) \times H(C|)}{si(i)}$$

kde

$$si(i) = -\sum_{v \in V_i} P(v) \log_2(P(v)$$

Vzdálenost je potom vypočtena jako

$$\Delta(X, Y) = \sum_{i=1}^{n} w_i \delta(x_i, y_i)$$

4.3 Metrika změny hodnoty (Modiffied Value Metric)

MVDM metric je způsob jak postihnout to, že podobné hodnoty v atributu znamenají klasifikaci do stejné třídy. Předchozí metriky dávaly velký důraz na výskyt stejných hodnot. Tady je pro výpočet vzdálenosti dvou instancí V_1, V_2 použito vzorce:

$$\delta(V_1, V_2) = \sum_{i=1}^{n} |P(C_i|V_1) - P(C_i|V_2)|$$

V Timblu jsou v zájmu urychlení předpočítány jednotlivé dvojice vzdáleností. Pro atributy s velkým přínosem informace bude většinou $\delta(V_1, V_2)$ velmi vysoká. Naopak pro atributy s malým množstvím informace bude klasifikace do třídy vcelku vyrovnaná.

MVDM dává při použití kNN dosti odlišné výsledky než přesahující metrika. Např. pokud dojde u Overlap Metric k nerozhodnému výsledku a instance se liší od ostatních v učící množině jen na jednom atributu, tak do množiny k-nearest budou vybírány všechny takové, které se shodují na všech atributech a na tom jednom nevyhovujícím mají jakoukoliv hodnotu. Při použití MVDM budou vybrány jen ty, co mají nejmenší $\delta(V_1, V_2)$ na místě špatně určeného atributu. Pokud by takovýchto instancí bylo málo, vezmou se ještě ty instance, které mají celkově nejmenší vzdálenost. To znamená, že se množiny instancí použité při kNN budou značně lišit.

4.4 Reprezentace instancí v paměti

Z důvodů efektivity prohledávání a nízké paměťové náročnosti jsou k reprezentaci instancí použity stromové struktury. Struktura je navržena tak, že pokud mají instance stejné hodnoty atributů, tak se nalézají na stejné cestě stromem. Pokud se instance liší na k-tém atributu, tak sdílejí společný prefix a dále se větví.

To má dva významné přínosy pro k-nearest neigbour Především je velice rychlé srovnání na úplnou shodu. Také je možné používat tzv. zkratky. To znamená, že pokud se všechny instance shodují na k až l-tých atributech, tak není potřeba srovnávat každý zvlášť, ale je možné je porovnat v jednom kroku. Druhým přínosem je, že počítání vzdáleností může částečně využívat shodnost prefixu, neboť pokud jdeme po stejné cestě, znamená to že se i instance shodují.

Dalším prvkem paměťové reprezentace je invertovaný index. Jeho využití je založeno na skutečnosti, že při počítání vzdáleností se používají i ty příklady, které nemají s danou instancí společný žádný atribut. Ale tyto příklady mají vždy nulovou podobnost. A právě tomuto zabrání *invertovaný index*. Konstruuje se jako záznam pro každý typ hodnoty všech atributů. V záznamu jsou uloženy instance, ve kterých se tato hodnota atributu vyskytla.

Při klasifikaci testovací množiny vezmeme jen seznam jejich atributů a vyhledáme v inverzním indexu instance, které se shodují. Tak velice rychle získáme k nejbližších sousedů.

Nevýhodou tohoto přístupu je to, že musíme v paměti uchovávat hned dvě struktury: strom a invertovaný index.

4.4.1 IGTree

Dobré výsledky Information Gain metriky (přínos informace) přiměly autory k navržení nové implementace paměťové struktury. IGTree obsahuje stejné informace jako struktury uvedené výše, ale v komprimované formě. Je to vlastně jistá forma rozhodovacího stromu. Instance jsou uchovávány jako cesta mezi uzly, které obsahují informaci o klasifikaci. Spojnice určují hodnoty jednotlivých atributů. Information Gain určuje pořadí atributů v jakém se budou do stromu přidávat.

Důvodem k této reprezentaci bylo to, že pokud jeden atribut dostatečně rozhoduje o klasifikaci dané instance, rozhodnutí se redukuje na porovnání s instancí, která byla již uložena v paměti. Pokud jeden (nejvýznamnější) atribut nestačí, použije se rozhodování na dalším (méně významném) atributu. Není tedy nutné indexovat všechny instance v paměti dvěmi strukturami, ale stačí reprezentace pomocí jednoho rozhodovacího stromu. To je významné z hlediska úspory paměti.

V každém uzlu stromu je také uchovávána informace o nejpravděpodobnější nebo implicitní klasifikaci těch instancí, které se dostaly až do tohoto uzlu. pomoc9 toho je možné rozhodovat i neznámá data. Pokud se totiž dostanou do uzlu, který není ještě listem, ale žádná cesta není vyhovující, tak se prostě oklasifikuje tou hodnotou, která je uložena v uzlu.

5 Rozdíly mezi verzí 2.0 a 3.0

Nedávno vyšla nová, vylepšená verze programu Timbl s označením 3.0. Zde se pokusím popsat ty nejpodstatnější rozdíly.

Základní rozdíly:

- Architektura client-server. Je zde možnost načítat testovací data z portu specifikovaného pomocí přepínače `-S portnumber`. Jednoduchý klient je součástí distribuce. Pokud jsou testována malá data v krátkých časových intervalech, tak toto rozšíření urychlí práci, neboť se nemusí stále dokola nahrávat vlastní program a nemusí se ani stále dokola učit. Je zde také možnost současného klasifikování více dat.
- Pomocí přepínače `-t leave_one_out` lze použít učící data i k odhadu chyby. Funguje to tak, že se Timbl pokusí klasifikovat každou instanci z učících dat (ale bez informace o ní samotné). Výhodou je to, že člověk nemusí vytvářet speciální testovací soubor.
- Podpora řídkých binárních dat. Pokud zpracováváme soubor s velkým počtem atributů, které mají jen dvě hodnoty, je možné ve vstupním souboru vypsat jen ty atributy, které jsou u dané instance nastaveny na pozitivní. Podrobnější popis formátu je v manuálu dodávaném s Timblem.
- Součtová metrika. Byly přidáno chi-kvadrát a sdílená variance jako další možnosti jak spočítat váhy jednotlivých atributů. Tato metrika je určena především pro atributy s velkým množstvím rozdílných hodnot.
- Možnost na každý atribut použít jinou metriku.
- Přepracovaná syntaxe zadávání atributů.
  - Metrika se již nezadává jako jedno číslo společné všem atributům, ale jako vektor.
  - Veškerý výstup lze již ovlivnit nastavením atributu `+v`
  - Nové přepínače `-v, +v` vypínají a zapínají výpisy informací.
  - Přepínač `+v` lze nastavovat i pomocí slovních nastavení. Samozřejmě je stále možné používat i čísla (jako ve verzi 2.0).
- Z důvodu optimalizace již není implicitně vytvářen inverzní index. To lze zapnout pomocí přepínače `+-`
- Je možné určit jméno výstupního souboru.

6 Experimenty s Timblem

Jak jsem již v úvodu řekl, Timbl byl vytvořen jako nástroj pro zpracování přirozeného jazyka. Já jsem jej však použil na datech z úplně jiného oboru.

Jedná se o úkol rozpoznat druh povrchu (město, pole atd.) zachyceného na jednom pixelu leteckého snímku. Celý snímek má rozměry 100×100 pixelů a zachycuje část Brněnské přehrady (`soubor sixs`). Každý pixel je popsán 6 hodnotami.

Možné klasifikace byly tyto:

```
poleb - pole bez vegetace
polev - pole s~nějakou vegetací
lesj  - les listnatý
lesj  - les jehličnatý
mesto
voda
```

Ještě jsem měl k dispozici čistá učící data o velikosti vždy 7×7 pixelů z každé oblasti (`all.data`). Dalším souborem bylo oklasifikování tohoto obrázku 100×100 pomocí statistické metody Maximum Likelihood Method (MLM)[3], kterou používají geografové k určení povrchu (soubor `maxlike.img`).

Mým úkolem bylo vyzkoušet jak se budou lišit výsledky statistické metody a metod použitých Timblem.

6.1 Dataset1

6.1.1 Data bez okolí

Použil jsem údaje ze `sixs` rozšířené o klasifikaci pomocí MLM metody. Data jsem rozdělil na 10 stejných dílů tak, že jsem vždy dával 2.-9. řádek do učící množiny 1 a 1. řádek do testovací. Získal jsem tím 10 souborů dat, které se navzájem lišily.

Při použití Timblu bez jakýchkoliv dodatečných nastavení parametrů, což vlastně znamená zavolat `Timlib -a0 -m0 -w1 -k1` kde:

- `-a 0` – znamená, že se použije metoda IB1 (nearest neigbor search)
- `-m 0` – pro výpočet se použije Weighted overlap matric
- `-w 1` – za váhu se bere Information Gain
- `-k 1` – počet sousedů vzhledem ke kterým se počítá klasifikace,

jsem obdržel výsledky uvedené v tabulce 1 (v tabulce je uveden počet špatně oklasifikovaných pixelů). Jsou tam již zachyceny mé pokusy s různým nastavením parametru k.

Z předcházející tabulky je naprosto zřejmé, že pouhé zvýšení parametru k nezvýší výrazně přesnost.

Další nastavení:

```
-m 1 -> použije se Modified Value Difference Metric pro výpočet váhy
Weighting   : No Weighting
```

Výsledky jsou v tabulce 2.

Ostatní nastavení už jen ve zkratce:

```
Timbl -m1 -w1 -k$i

Weighting   : GainRatio
```

	k=1	k=2	k=5	k=10
soubor cislo 0:	161	166	188	193
soubor cislo 1:	158	155	155	173
soubor cislo 2:	169	172	160	175
soubor cislo 3:	194	196	172	195
soubor cislo 4:	179	177	185	203
soubor cislo 5:	198	189	187	210
soubor cislo 6:	189	181	194	194
soubor cislo 7:	181	175	170	194
soubor cislo 8:	182	182	173	198
soubor cislo 9:	179	184	185	199
průměr různých:	179	177.7	176.9	193.4
prum přesnost:	82.1%	82.2%	82.3%	80.6%

Tabulka 1: nastavení: IB1, WO metric, IG weight

```
Timbl -m1 -w2 -k$i

Weighting   : InfoGain
```

výsledek viz. tabulka 3. Je to průměr počtu špatně klasifikovaných případů.

Pomocí algoritmu "nearest neighbor search" se mi podařilo dosáhnout přesnosti více než 93.4 procenta. Z výsledků je vidět, že počet sousedů (parametr `-k`) ovlivňuje výsledek vcelku minimálně a podle mého názoru zpomalení výpočtu není naprosto adekvátní jeho zpřesnění.

Ostatní metody implementované v Timblu nepřinesly žádné výrazné zvýšení přesnosti. Možná nastavení:

`-a 1` – Použití algoritmu IGTree, což je reprezentace učící množiny pomocí rozhodovacího stromu (podrobněji air95.ps v distribuci Timblu).

Výsledky, které jsem dosáhl touto metodou však nebyly nijak uspokojující. S použitím IGTree se mi nepodařilo dosáhnout větší přesnosti než je 80 procent. Ale výpočet je velice rychlý.

```
Timbl -a 1 -> Algorithm   : IG -> prum pocet chyb =  199
-> prumerna presnost = 80.1%
```

Kombinace rychlosti a vyhledávání sousedů využívá algoritmus Tribl. Jedná se o hybrid mezi IB1 a IGTree. Výsledky jsou v tabulce 4: (do posledního sloupce jsem přidal sloupec výsledky získané čistým IGTree)

Jak sem psal již výše, je rozhodování pomocí IB1 proti IGTree podstatně pomalejší. Je zde ale patrný jistý rozdíl mezi Trimblem s parametrem `q0` a a IB1 algoritmem. I přes to, že se jedná v podstatě o stejný výpočet, klasifikace pomocí Trimbu trvala 9.16 s, a IB1 ta stejná data zvládl za 8.76 s.

Dá se tedy říct, že Timbl dává v porovnání s výsledky MLM akceptovatelnou přesnost výsledků při použití klasického Nearest Neighbor Search algoritmu

	k=1	k=2	k=5	k=10
soubor cislo 0:	82	84	66	66
soubor cislo 1:	58	66	54	62
soubor cislo 2:	86	66	62	57
soubor cislo 3:	107	96	89	79
soubor cislo 4:	91	96	76	86
soubor cislo 5:	90	92	84	75
soubor cislo 6:	88	86	87	84
soubor cislo 7:	67	81	70	76
soubor cislo 8:	79	74	64	72
soubor cislo 9:	91	90	72	62
průměr	75.7	74.7	65.8	65.3
prům. přesnost:	92.43%	92.53%	93.42%	93.47%

Tabulka 2: nastavení: IB1, MVD metric, No Weighting

	k1	k2	k5	k10	k20
-w1	84.6	85.3	72.1	75.3	74.8
-w2	79.0	80.4	69.8	72.0	
úspěšnost					
-w1	91.1%	91.4%	92.79%	92.47%	92.52%
-w2	92.1%	91.9%	93.0%	92.8%	

Tabulka 3: Nastavení: IB1, MVD metric, Gain Ratio nebo Information Gain

spolu s metriku tzv. Modified Value Difference Metric, kde lze dosáhnout až 93% přesnost.

Pokud bychom se spokojili s menší přesností (do 90%), ale vyžadujeme vyšší rychlost, dá se použít Tribl algoritmus s nastaveným parametrem q na 1.

6.1.2 Rozšíření o okolí

Hlavní motivací k tomuto experimentu byl názor, že pokud rozšíříme informaci o pixelu ještě o informaci o klasifikaci okolních bodů, tak dostaneme větší přesnost. Jako základní jednotka pro učení okolí byl čtverec 3×3 pixely.

První nápad byl, že by mohlo stačit použít místo vektoru 6 hodnot pro každý bod, vektor 54 atributů (9 bodů $\times$ 6 atributů). Na takto zpracovaných datech výpočet trval velice dlouhou dobu a výsledná přesnost nedosahovala ani 70 procent. Asi došlo k přeurčení dat.

6.1.3 Podle klasifikace okolních bodů

V dalším kroku jsem rozšířil `sixs` o okolí získané z `maxlike`. Takže soubor vypadal asi takto:

```
77,29,25,70,53,16,pb,ll,pb,ll,ll,ll,ll,ll,lesl
```

	q=0	q=1	q=2	q=3	a=1
soubor cislo 0:	ruzne 66	85	136	233	179
soubor cislo 1:	ruzne 54	76	119	218	189
soubor cislo 2:	ruzne 62	74	111	206	190
soubor cislo 3:	ruzne 89	97	144	230	208
soubor cislo 4:	ruzne 76	91	134	241	199
soubor cislo 5:	ruzne 84	103	150	247	203
soubor cislo 6:	ruzne 87	100	148	251	211
soubor cislo 7:	ruzne 70	85	137	231	213
soubor cislo 8:	ruzne 64	85	113	219	189
soubor cislo 9:	ruzne 72	91	146	244	209
průměr	72.4	88.7	133.8	232	199
prům přesnost	92.76%	91.13%	86.2%	76.8%	80.1%
čas výpočtu	9.616	2.953	1.917	1.73	1.65

Tabulka 4: Algoritmus IGTree a Trimbl

kde 7. – 14. atribut jsou klasifikace okolních pixelů a poslední atribut je klasifikace centrálního pixelu. Všechny tyto údaje jsem bral z maxlike.img.

Je zajímavé, že při std. nastavení dával Timbl největší váhu atributům určujícím okolí. Výsledky byly však podstatně horší, než bez použití informace o okolí, viz. tabulka 5.

	různé	přesnost
soubor cislo 0:	351	81,74%
soubor cislo 1:	352	81,67%
soubor cislo 2:	365	80,99%
soubor cislo 3:	360	81,27%
soubor cislo 4:	372	80,63%
soubor cislo 5:	356	81,46%
soubor cislo 6:	358	81,38%
soubor cislo 7:	386	79,91%
soubor cislo 8:	338	82,40%
soubor cislo 9:	386	79,91%
průměr	362.4	81,14%

Tabulka 5: Pomocí klasifikace okolí

U minulého experimentu nejlepší přesnost dávalo nastaveni "m 1". Výsledek na souboru s okolím při tomto nastavení je:

```
průměr špatně klas. 237.9, prům přesnost 87,62
```

což také není to, co by jsem čekal. Výpočet Timblu se oproti datům s 6 atributy silně zpomalil (asi tak 20 x).

Z výsledků pro různě nastavený parametr k se nedá vyčíst žádná tendence vývoje přesnosti. V tabulkách jsou uvedené výsledky pro $k = 1$, pro 2 přesnost klesne, ale při nastavení k na 5, se výsledky pohybují kolem 90 procent.

Většinou se úspěšnost pro tento vstupní soubor pohybovala kolem 85 procent v porovnání s MLM.

6.1.4 Podle počtu okolních bodů

Dalším souborem byla data získaná tak, že jsem za základních 6 atributů dodal počty různých druhů povrchu, které se vyskytují v okolí 3 x 3. Formát byl takovýto:

```
6 x spektrum,#poleb,#lesj,#lesl,#mesto,#polev,#voda,centalní pixel
```

Výsledky byly trochu lepší než v předchozím případě, ale stále horší, než u dat bez okolí. Dostal jsem se na přesnost 91,8% při nastavení Modiffied Value Metric (`-m1`) spolu s Information Gain weighting (`-w2`).

Závěrem se dá říct to, že rozšíření o okolí nijak nepomohlo. Nejpřesnější výsledky jsem dostal ze souboru, který obsahoval pouze 6 atributů a to při nastavení `-m1 -w0 -k10`, kde byla úspěšnost 93.47 procent. Rozšíření o počet okolních bodů se tomuto výsledku sice přiblížilo, ale nepřekonalo ho.

6.2 Dataset2

Podobné experimenty se prováděly na druhém snímku. Hlavní rozdíl spočíval v tom, že u tohoto snímku měly být lépe vybrány učící data. V minulém případě totiž byla dodaná učící data vybraná z naprosto šumu-prostých oblastí, což mělo za následek nepřesnost při rozhodování bodů, které se nacházely na přechodu mezi dvěma prostředími. Toto mělo být v druhém snímku odstraněno.

Když jsem naučil Timbl na čistých datech a poté otestoval na celém obrázku, tak výsledky dopadly takto:

	mvd			wo		
	nw	gr	ig	nw	gr	ig
různých	3571	3653	3634	3878	4001	3930
prům. přesnost	64.29	63.47	63.66	61.22	59.99	60.70

Tabulka 6: Výsledky Timblu pro učící množinu a IB1 algoritmus

Kde zkratky jsou tvořeny z parametrů s kterými byl volán Timbl. Znamenají toto:

- wo = Weighted Overlap Metric
- mvd = Modified Value Difference Metric

	mvd			wo		
	nw	gr	ig	nw	gr	ig
různých	4691	5002	4574	4691	5002	4574
prům. přesnost	53.09	49.98	54.26	53.09	49.98	54.26

Tabulka 7: Výsledky Timblu pro učící množinu a IGTree algoritmus

- nw = No Weighting
- gr = Gain Ratio weighting
- ig = Information Gain weighting

Pro vyšší počet sousedů se výsledná přesnost spíše snižovala, takže jsem ji do výpisu ani nedával. Pokud tyto výsledky srovnám s tím, co mi dával Timbl na prvních datech:

```
soubor velky.data.mbl.wo.gr.k1.out: různě klasifikovaných 4332, přesnost 56.68]\\
```

tak se změnou učící množiny nic výrazného nestalo.

Zkusil jsem vytvořit svou vlastní učící množinu, která bude mít podobné zastoupení jednotlivých kategorií. Naprosto náhodně jsem ze souboru `newsixs.data` (obrázek 100×100) vybral učící množinu která obsahovala:

	moje	dodaná	maxlike
beton	35	66	609
hole	85	140	1543
veg1	199	54	3533
veg2	35	49	541
pole2	76	136	1518
pole3	125	91	2256
celkem	555	536	10000

Zajímavé bylo, že s takto vybranou učící množinou jsem měl mnohem vyšší přesnost:

```
soubor ucmnozina.mbl.mvd.nw.k1: ruzně klasifikovaných 1197, přesnost 87.32
```

Proti výsledkům z Dataset1 se nový výběr učící množiny neprojevil lepšími výsledky. Je zde ale vidět silná závislost na tom, jaká se použije učící množina, protože pro náhodný vzorek (o přibližně stejném obsahu) dává Timbl výrazně lepší výsledky.

	průměrná přesnost v %			
nastavení	k=1	k=2	k=5	k=10
mvd.gr	93.42	93.01	93.99	93.63
mvd.ig	93.55	93.09	94.11	93.42
mvd.nw	93.63	93.08	94.09	93.77
wo.gr	82.43	82.13	82.54	80.20
wo.ig	82.62	82.33	82.45	79.95
wo.nw	84.94	82.71	69.97	69.97

Tabulka 8: Výsledky IB1 algoritmu na Dataset2

6.2.1 Učení ze 100×100

Zopakoval jsem experimenty, které jsem dělal nad souborem Dataset1. Rozdělil jsem data z obrázku na 80% učící a 20% testovací. Šlo mi především o to zjistit, jestli pro stejná nastavení dostanu výsledky podobné s Dataset1.

A opravdu, Timbl se chová velice podobně jako na předchozích datech, viz tabulka 8.

Nejlepší výsledek zase dostaneme při nastavení `-m1`. Na tomto konkrétním případě je skoro jedno, jestli za váhu atributů zvolíme No Wrighting nebo InfoGain. Celkově jsou výsledky skoro totožné s tím, co dával Timbl i na Dataset1.

Rozdíl v přesnosti mezi učením z čistých dat (max 65 procent) a výše uvedenými výsledky mě inspiroval k experimentu s výměnou učících a testovacích dat (20 procent učící / 80 procent testovací). V tom případě bude v učící množině 2000 instancí, což je kompromis mezi 536 (v dodaných) a 8000 (v předchozím experimentu). Výsledek z deseti průchodů byl takovýto:

```
prumerný počet špatně určených 1695.8, prům přesnos 0.7880
```

Z těchto výsledků je vidět, že výsledek je závislý na velikosti testovací množiny. Pokles přesnosti ale není nijak velký (proti poklesu velikosti učící množiny), takže si troufám odhadovat, že pokud by byl Timbl naučen na datech, která mu vyhovují, je možné dosáhnout i z malé učící množiny přesnost kolem 80 procent.

Asi nejnázorněji je vidět rozdíl mezi jednotlivými výsledky při grafické reprezentaci snímku. V příloze jsou obrázky z Dataset2 vygenerované:

a) Maximum Likelihood metodou

b) Timblem pomocí učení na čistých datech

c) Timblem naučeným na mých náhodně vybraných instancích

7 Závěr

Timbl byl vytvořen pro potřeby zpracování přirozeného jazyka, kde se většinou pracuje s diskrétními hodnotami atributů. Data na kterých byl testován měla všechny atributy spojité, což mohlo také ovlivnit výsledek. Ne nepodstatnou roli hraje i to, že není jak posoudit správnost rozhodnutí Timblu, neboť metoda Maximum Likelihood není také úplně přesná. Asi nejnázornější je porovnat jednotlivé obrázky z přílohy a rozhodnout, který z nich odpovídá nejvíce realitě.

Na dodaných datech se mi podařilo dosáhnout přesnosti 94 procent (za předpokladu, že učící množina zahrnovala 80 procent všech dat). Pro spojité atributy je nejvýhodnější metrika Value Difference (`-m1`). Rozšiřování informace o pixelu nevedlo k výraznému zlepšení. Z toho se dá usuzovat, že jednotlivé body jsou dostatečně určeny již hodnotami 6 atributů.

Program Timbl je vcelku snadno ovladatelný a stabilní. Kromě dat, která byla opravdu dosti rozsáhlá (54 atributů) dával výsledky řádově v sekundách. I po stránce dokumentace je vcelku zdařilý. Součástí distribuce je i popis jednotlivých objektů, takže je možné jej i rozšiřovat a zdokonalovat.

Reference

[1] Daelemans W., Zavrel J., Van der Sloot K., Van den Bosch.: TiMBL: Tilburg memory based learner. TR 99-01, ILK, 1999. `http://ilk.kub.nl/ ilk/papers/ilk9901.ps.gz`

[2] Mitchell, T.M.: Machine Learning. McGraw Hill, New York, 1997.

[3] Ripley B.D.: Pattern Recognition and Neural Networks. Cambridge University Press, 1996.

[4] UCI Machine Learning repository. `http://www.ics.uci.edu/AI/ML/Machine-Learning.html`

A Obrázky

Vysvetlivky:

```
beton = 'b'
hole  = ' '
veg1  = 'v'
veg2  = '^'
pole2 = '"'
pole3 = '~'
```

Obrázek 1: Výsledek MLM

Obrázek 2: IB1 s učícími daty

Obrázek 3: IB1 s naučený na mých datech