Semestrální práce

KIV-SU

Strojové učení

Jakub Zíka - A15N0087P
zikaj@students.kiv.zcu.cz

3. února 2017

Obsah

1 Zadání

Navrhněte téma zadání semestrální práce související s oblastí strojového učení. Cílem práce je prohloubit znalosti studenta v oblasti kognititvních systémů pomocí nabytých zkušeností ze semetrální práce.

1.1 Vybrané zadání

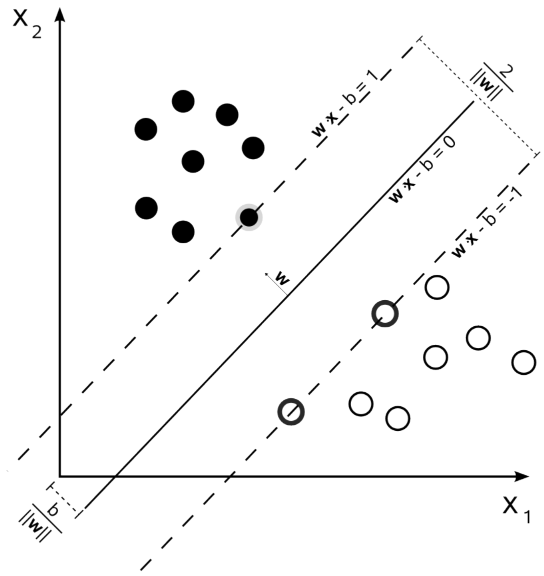
Sestrojte klasifikátor *Support Vector Machines*, dále už jen *SVM*, který bude skrze osobní údaje pasažérů lodi Titanic klasifikovat, zda daná osoba přežije či nepřežije potopení lodi. Data obsahují následující příznaky:

1. survived : přežil/nepřežil (1/0)
2. pclass : socio-ekonomická třída pasažéra (1=vyšší, 2=střední, 3=nižší)
3. name : jméno
4. sex : pohlaví
5. age : věk
6. sibsp : počet sourozenců, partnerů na palubě (příbuzní stejné generace)
7. parch : počet rodičů, dětí na palubě (příbuzní odlišné generace)
8. ticket : ID lístku
9. fare : cena lístku
10. cabin : číslo kajuty
11. embarked : přístav nalodění (C=Cherbourg, Q=Queenstown, S=Southampton)

2 Teoretický úvod

2.1 Support Vector Machines

Algoritmy strojového se skládají z *trénovací množiny* a *rozhodovací hranice*. Rozhodovací hranici můžeme též nazývat *hypotéza*. Algoritmy tedy dělíme na *učení s učitelem* (máme odpovědi) a *učení bez učitele* (nemáme odpovědi).



Obrázek 1: Maximální pás bez bodů trénovací množiny [4]

Každý vzorek $\vec{x}$ trénovací množiny je tvořen množinou příznaků

$$\vec{x} = [x_1, x_2, ...x_n],$$

kde n je počet příznaků. Hypotéza má tvar $h : x \rightarrow y$ což znamená, že hypotéza je zobrazení x do y. Hypotézu tvoří modelovací parametry

$$\vec{\Theta},$$

které nám umožňují nastavit rozumnou rozhodovací hranici. Jejich hodnoty předem neznáme a získáme je trénováním.[2],[1]

2.1.1 Lineární rozhodovací hranice

SVM je metoda strojového učení, která hledá v trénovací množině umístění optimální nadroviny. Tato nadrovina slouží k rozdělení bodů projekce na dvě třídy. V tomto rozdělení je požadováno aby minimum vzdáleností bodů od této nadroviny bylo co největší. Chceme tedy, aby nadrovina měla po obou stranách co nejširší pás bez bodů. K popisu těchto pásů slouží pomocné vektory (*Support Vectors*) (viz Obr.:1). [1]

2.1.2 Cenová funkce

Metoda SVM je vylepšenou verzí *logistické regrese.* Ovšem, na rozdíl od cenové funkce logistické regrese nám SVM nevrací pravděpodobnost, ale rovnou příslušnost klasifikovaného vzorku k třídě 1 nebo 0. Zde vidíme *cenovou funkci* logistické regrese:

$$\min_{\vec{\Theta}} \frac{1}{m}[\sum_{i=1}^{m} y^{(i)}(-\log h_{\vec{\Theta}}(x^{(i)})) + (1-y^{(i)})(-\log(1-h_{\vec{\Theta}}(x^{(i)})))] + \frac{\lambda}{2m}\sum_{j=1}^{n}\Theta_j^2.$$

Cenová funkce SVM pak vypadá následovně:

$$\min_{\vec{\Theta}} C\sum_{i=1}^{m}[y^{(i)}Cost_1(\vec{\Theta}^T x^{(i)}) + (1-y^{(i)})Cost_0(\vec{\Theta}^T x^{(i)})] + \frac{1}{2}\sum_{j=1}^{n}\Theta_j^2,$$

kde $Cost_1$ funkce pro y = 1 je

$$\log(\frac{1}{1+e^{-\Theta^T x}})$$

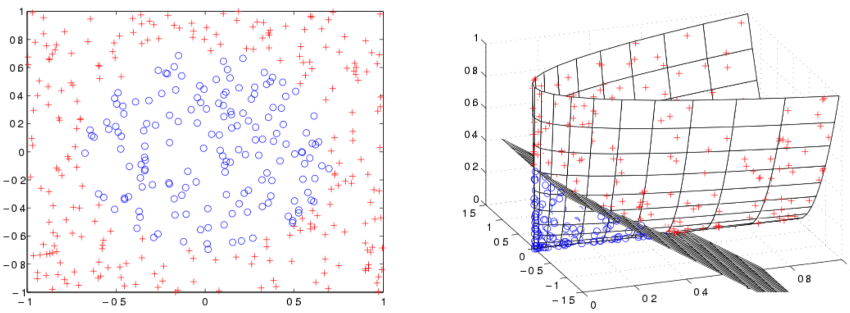
a funkce $Cost_0$ funkce pro y = 0 je

$$\log(1-\frac{1}{1+e^{-\Theta^T x}}).$$

Optimalizací cenové funkce získáme hodnoty parametrů $\vec{\Theta}$, které určují tvar hypotézy.

Hodnota C je regularizační faktor, který ovlivňuje výběr hypotézy. Rozumně vybraná hodnota pak umožní dělat ve výběru hypotézy kompromisy v extrémních případech rodělení tříd v trénovací množině:

1. C - vysoké = malá odchylka, velký rozptyl (malá λ), hrozí *overfitting*
2. C - malé = velká odchylka, malý rozptyl (velká λ), hrozí *undefitting*



Obrázek 2: Ukázka lineárně neseparabilních a separabilních dat.[4]

2.1.3 Nelineární rozhodovací hranice

Máme-li trénovací množinu, pro kterou je rozhodovací hranice nelineární (viz. Obr.:2), musíme použít metodu jader (*Kernels*). Zavedeme si i pomocných bodů tzv. *landmarky*. Každý landmark $l^{\overrightarrow{(i)}}$ nese hodnotu vzdálenosti od ostatních prvků trénovací množiny. Ke každému prvku $x^{\overrightarrow{(m)}}$ trénovací množiny spočteme podobnost f_i s každým landmarkem $l^{\overrightarrow{(i)}}$. Pro každý prvek trénovací množiny tak dostaneme vektor podobnosti $f^{\overrightarrow{(m)}}$. Podobnost počítáme následovně:

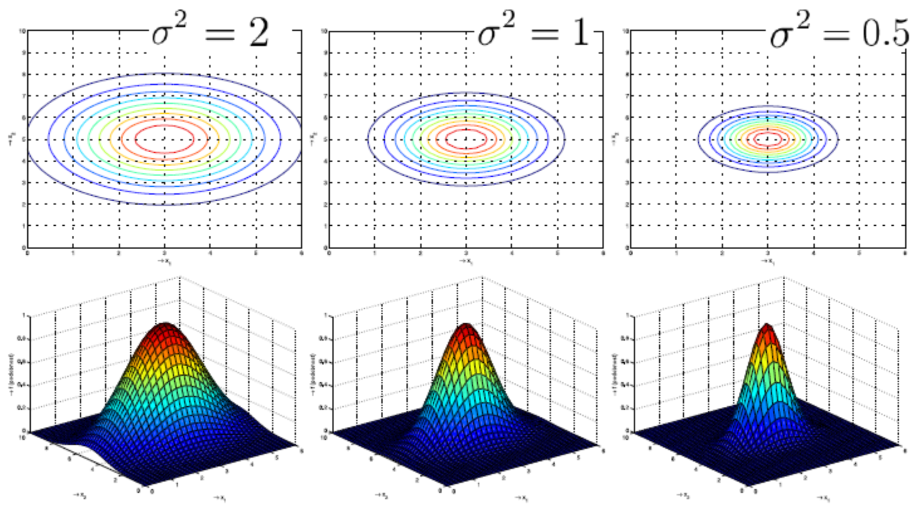
$$f_i(x^{\overrightarrow{(m)}}, l^{\overrightarrow{(i)}}) = exp(-\frac{||x^{\overrightarrow{(m)}} - l^{\overrightarrow{(i)}}||}{2\sigma^2})$$

a vektor podobnosti pak bude vypadat:

$$f^{\overrightarrow{(m)}} = [f_1, f_2, f_3, ..., f_i].$$

Jádrem se nazývá funkce počítání podobnosti. V tomto případě je jádro *Gaussové*. Parametr *sigma* nám určuje míru podobnosti. Máme i jiné funkce jádra jako například *Polynomiální* nebo *Lineární*. Lineární jádro je předchozí případ lineárně separabilních dat.[5],[4]

Podobnost se určuje na grafu podobnosti (viz Obr.:3) tak, že čím je zkoumaný prvek blíže globálnímu maximu gaussovského klobouku, tím míra podobnosti narůstá. Jak moc je graf špičatý, či plochý závisí právě na parametru σ.



Obrázek 3: Míra podobnosti v závislosti na parametru σ.[1]

2.1.4 Cenová funkce s jádry

Při použití gaussového jádra máme předpočítaný vektor podobnosti pro každý prvek trénovací množiny. To znamená, že vektor podobnosti může reprezentovat daný prvek trénovací množiny. Cenouvou funkci tedy můžeme pozměnit do tvaru:

$$\min_{\vec{\Theta}} C \sum_{i=1}^{m} [y^{(i)} Cost_1(\Theta^T \vec{f^{(i)}}) + (1 - y^{(i)}) Cost_0(\Theta^T \vec{f^{(i)}})] + \frac{1}{2} \sum_{j=1}^{n} \Theta_j^2.$$

Pozor, gaussové jádro je citlivé na velké rozdíly hodnot mezi jednotlivými příznaky. Je dobré příznakový vektor nejdříve naškálovat a až poté počítat cenovou funkci.

2.2 Sequential minimal optimization

SMO je algoritmus objevený Johnem Plattem v roce 1998. Tento algoritmus se osvědčil v SVM když jím nahradíme klasickou cenovou funkci. Za pomoci *Lagrangoevých multiplikátorů* můžeme říct, že $\Theta(\alpha) = \sum_i \alpha_i y_i z_i$, kde α_i jsou Lagrangeovy multiplikátory a $z_i = \phi(x_i)$.

Platt toto vylepšení poté využije jako:

$$F_i = w(\alpha).z_i - y_i = \sum_j \alpha_j y_j k(x_i, x_j) - y_i,$$

kde $k(x_i, x_j)$ je jádro. Dostáváme tvar parciální derivace:

$$L = \frac{1}{2} w(\alpha).w(\alpha) - \sum_i \alpha_i - \sum_i \alpha_i \delta_i + \sum_i \mu_i(\alpha_i - C) - \beta \sum_i \alpha_i y_i,$$

$$\frac{\partial L}{\partial \alpha_i} = (F_i - \beta) y_i - \delta_i + \mu_i = 0.$$

Podle *Karush-Kuhn-Tucker (KKT)* dostaneme podmínky, které musí daný výraz splňovat:

$$(F_i - \beta) y_i \geq -\tau \Rightarrow \alpha_i = 0,$$

$$|(F_i - \beta)| \leq \tau \Rightarrow 0 < \alpha_i < C,$$

$$(F_i - \beta) y_i \leq \tau \Rightarrow \alpha_i = C.$$

Platt poté definuje výstupní chybu i-tého prvku jako:

$$E_i = F_i - \beta,$$

kde β je prahový paramter. V Plattově pseudokódu po určení chyby vybíráme dva indexy i_1 a i_2 pro výběr multiplikátorů α_{i_1} a α_{i_2}. Metody, jak správně vybrat indexy, jsou popsány v originálním článku z roku 1998. Celý algoritmus tedy funguje ve dvou smyčkách. Vnější smyčka vybírá index i_2 a pro index i_2 vybírá vnitřní smyčka index i_1.

Vnější smyčka jde přes všechny prvky trénovací množiny, které narušily podmínky optimality. Nejprve pouze přes ty, které nanerušili horní ani dolní hranici. V každém průběhu, je-li to vyžadováno, je chyba E_i aktualizována.[6],[2]

Dále je nutné si spočítat horní a dolní hranici. Pokud se y_{i_1} nerovná y_{i_2} pak:

$$L = max(0, \alpha_2 - \alpha_1); H = min(C, C + \alpha_2 - \alpha_1)$$

jinak:

$$L = max(0, \alpha_2 + \alpha_1 - C); H = min(C, \alpha_2 + \alpha_1).$$

Druhá derivace cílové funkce podél diagonály může být vyjádřena jako:

$$\eta = K(\vec{x_1}, \vec{x_1}) + K(\vec{x_2}, \vec{x_2}) - 2K(\vec{x_1}, \vec{x_2}).$$

Za normálních okolností bude cílová funkce pozitivně definitní, bude existovat minimum ve směru lineárního omezení rovnosti a η bude větší než nula. V tomto případě SMO vypočítá minimum ve směru omezení:

$$\alpha_2^{new,cliped} = \alpha_2 + \frac{y_2(E_1 - E_2)}{\eta}.$$

Teď si vyjádříme $s = y_1 y_2$. Potom se α_1 počítá:

$$\alpha_1^{new} = \alpha_1 + s(\alpha_2 - \alpha_2^{new,clipped}).$$

Dále si musíme vytvořit vzorce pro výpočet prahu. Práh β je počítán po každém provedeném vnitřním cyklu. Celkový práh je složen z β_1, který je ovlivněn chybou E_1, a β_2, který je ovlivněn chybou E_2. Následující práh β_1 je platný, pokud nová α_1 není v mezích, protože nutí výstup z SVM být y_1, když vstup je x_1:

$$\beta_1 = E_1 + y_1(\alpha_1^{new} - \alpha_1)K(\vec{x_1}, \vec{x_1}) + y_2(\alpha_2^{new} - \alpha_2)K(\vec{x_1}, \vec{x_2}) + \beta.$$

Předchozí trvrzení pro β_1 platí i pro β_2. Následující práh β_2 je platný, pokud nová α_2 není v mezích, protože nutí výstup z SVM být y_2, když vstup je x_2:

$$\beta_2 = E_2 + y_1(\alpha_1^{new} - \alpha_1)K(\vec{x_1}, \vec{x_2}) + y_2(\alpha_2^{new} - \alpha_2)K(\vec{x_2}, \vec{x_2}) + \beta.$$

Pokud oba prahy β_1 i β_2 jsou platné, jsou si rovny. Když oba nové Lagrangeovy multiplikátory jsou v mezích a pokud L není rovno H, pak interval mezi β_1 a β_2 je práh, který je v souladu s podmínkami KKT. SMO zvolí prahovou hodnotu jako polovinu vzdálenosti mezi β_1 a β_2.[7]

3 Implementace

Výsledný program je naprogramovaný v matematickém jazyce Octave. Tuto variantu jsem zvolil z důvodu velkého množství matematických operací v úloze. Jelikož výsledný program slouží pouze k vyzkoušení dané problematiky, je tento jazyk optimální pro jeho realizaci.

Program má hlavní část `main.m`. Ta je pak rozdělena na `settings.m, preprocessing.m`, `trainManual.m`, `predictManual.m` a `statistics.m`.

3.1 Příprava dat

Před načtením dat jsem analyzoval jednotlivé příznaky všech prvků trénovací množiny pomocí programu *Weka-3.8.1*.

3.1.1 Analýza a čištění

Příprava prostředí, načtení a čištění dat probíhá v částech settings.m a preprocessing.m. Po prozkoumání jednotlivých příznaků jsem zjistil, že sloupce *embarked, cabin* a *age* mají některá pole nevyplněná. V příznaku *embarked* chybí 2 hodnoty, proto je jednodušší tyto řádky smazat než vymýšlet postup nahrazení. Naopak příznaku *cabin* chybí 687 záznamů z 891, což je 77%. Pokud bychom chtěli tyto mezery v datech nahradit například střední hodnotou nebo generátorem náhodných čísel, výsledné zkreslení hypotézy by bylo příliš velké a stala by se tato operace spíše nevýhodou. Proto sloupec *cabin* odstraníme úplně. Posledním příznakem s chybějícími hodnotami je *age*. Zde chybí 177 záznamů z 891, což je 20%. To není zase tak mnoho, chybějící hodnoty nahradím střední hodnotou příznaku. Výslednou hypotézu mi to může zkreslit, ale myslím si že ne tolik, jako by se tomu stalo v případě příznaku *cabin*.

Příznak *cabin* bych odstranil i z jiného důvodu. Sice obsahuje pouze 147 hodnot, ale z toho je 101 unikátních. Pokud mám velké množství unikátních hodnot, výsledný sloupec mi poté slouží jako množina identifikátorů. Identifikátory do trénovací množiny nepřináší žádnou přidanou hodnotu a jsou tudíž zbytečné. Odstraním tedy i sloupec *name*, který je jednoznačným identifikátorem, protože má 100% unikátních hodnot.

3.1.2 Konverze

Jelikož nejsou všechny příznaky číselné hodnoty, je nutné je konvertovat. To platí pro sloupce *age*, *ticket* a *embarked*. U sloupců *ticket* a *embarked* nevím, jaké jiné hodnoty se zde mohou vyskytovat, takže přiřadit jednotlivým slovům unikátní číselný identifikátor nejde použít. Znaky každé buňky tedy převedu podle `ASCII` tabulky na celočíselnou hodnotu a sečtu. Náhled konverze pro sloupec *embraked*:

```
# preprocessing.m
# line 43
# EMBARKED to NUMBER
for i = 1:countRow
  data(i, 8) = sum(cell2mat (toascii(data(i, 8))));
endfor
```

Příznak *sex* lze konertovat na binární klasifikátor. Jelikož víme, že pohlaví u lidí jsou pouze dvě, převedeme hodnoty `male` a `female` na binární hodnotu, tedy na 0 a 1.

3.2 Trénování

3.2.1 Škálování a matice podobnosti

Před trénováním jsou data ještě škálována aby se srazili rozdíly mezi jednotlivými příznaky a došlo tak ke správnému natrénování. Ještě dříve jsou data rozdělena podle uživatelem zadaného poměru. Škálují se tedy dvě matice, jedna tránovací `scaleTrainingSet` a druhá validační `scaledValidationSet`. Příznaky se škálují odečtením maxima příznaku přes všechny vzorky trénovací množiny

$$s_n^i = x_n^i - \max_i x_n \, ; \, i \in < 1; m > .$$

Matice `scaledValidationSet` je škálována podle maximálních hodnot `scaleTrainingSet` matice. Kód škálování pak vypadá následovně:

```
# scale.m
# line 7
for i = 1:countColumn
  for j = 1:countRow
    scaledTrainingSet(j,i) = trainingSet(j,i) / maxFeature(i);
  endfor
endfor
```

Jako landmarky označím všechny prvky trénovací množiny. Matice podobnosti tedy bude obsahovat podobnost všech prvků trénovací množiny vůči ostatním prvkům. Matice `scaleTrainingSet` bude symetrická. Kód pro výpočet podobnosti je kód gaussového jádra.

```
# gaussianKernel.m
# line 5
function f = gaussianKernel(x1, x2, sigma)
  f = exp(-norm(x1 - x2)^2 / 2 * sigma^2);
end
```

3.2.2 Minimalizace cenové funkce

Samotné trénování probíhá v části `trainManual.m`. Na začátku proběhne nastavení proměnných a změna odpovědí učitele z 0/1 na -1/1. Dává nám to širší rozhodovací pásmo. Učení pak probíhá ve dvou cyklech, *vnitřním* (pracuje s j-tým prvkem) a *vnějším* (pracuje s i-tým prvkem). V obou cyklech se počítají výstupní chyby i/j-tého prvku. Ve vnějším se `i` generuje iteračně přes `for i = 1:m`, kde `m` je velikost trénovací množiny, ve vnitřním je výběr j-tého prvku realizován náhodou:

```
# trainManual.m
# line 55
j = ceil(m * rand());
while j == i,
  j = ceil(m * rand());
end
```

Dále si podle vzorců z [7] spočítám hodnoty potřebné k výpočtu rozhodovací hranice. Hodnoty `L,H` jsou hranice vymezující pás podél rozhodovací hranice. Parametr `eta` obsahuje výpočet druhé derivace funkce. Hodnoty `b,b1,b2` slouží k výpočtu prahové hodnoty. Hodnoty vektoru `alphas` Lagrangeovy multiplikátory. Zastavovací podmínka se řídí parametrem `passes`. Pokud se po průběhum celým vnějším cyklem `for i = 1:m` nezmění žádný z parametrů `alphas`, zvedne se hodnota proměnné `num_changed_alphas` o 1. Jakmile je splněna podmínka $passes < max\_passes$, trénování končí a přejde se k zápisu výsledků trénování do proměnné `model`. Zápis výsledků:

```
# trainManual.m
# line 153
model.y = Y;
model.alphas = alphas;
model.w = ((alphas.*Y)'*X)';
```

V modelu hodnota `model.y` představuje odpovědi učitele. Hodnota `model.alphas` je podle teorie α_i, což jsou Lagrangeovi multiplikátory. Pole `model.w` jsou pro nás hodnoty modelovacích parametrů Θ.

3.3 Predikce

Predikce výsledků probíhá v modulu `predictManual.m`. Základem je mít data ve stejném stavu, jako je trénovací množina. Proto úprava validovaných dat probíhá společně s trénovací množinou. Ne vše jde ale dělat najednou. Validovaná data jsou škálována podle maximálních hodnot příznaků trénovací množiny. Výpočet podobnosti je u trénovací množiny mezi prvky samotnými. Pro validační data se počítá podobnost s prvky trénovací množiny. Validační matici vynásobím odpověďmi učitele s hodnotami -1/1. Nakonec matici ještě přenásobím Lagrangeovými multiplikátory, tzn. parametry $\vec{\alpha}$.

Dostanu desetinné číslo v rozsahu $< -1; 1 >$. SVM ale musí vracet jasnou odpověď 1 nebo 0. Je-li hodnota predikce větší rovna 0, vrátím jedničku. V opačném případě vracím 0.

```
# predictManual.m
# line 27
pred(p >= 0) = 1;
pred(p < 0) =  0;
```

4 Závěr

5 Uživatelská příručka

5.0.1 Požadavky

Aby bylo možné program spustit, je nutné mít nainstalovaný program GNU Octave ve verzi 4.2.0.

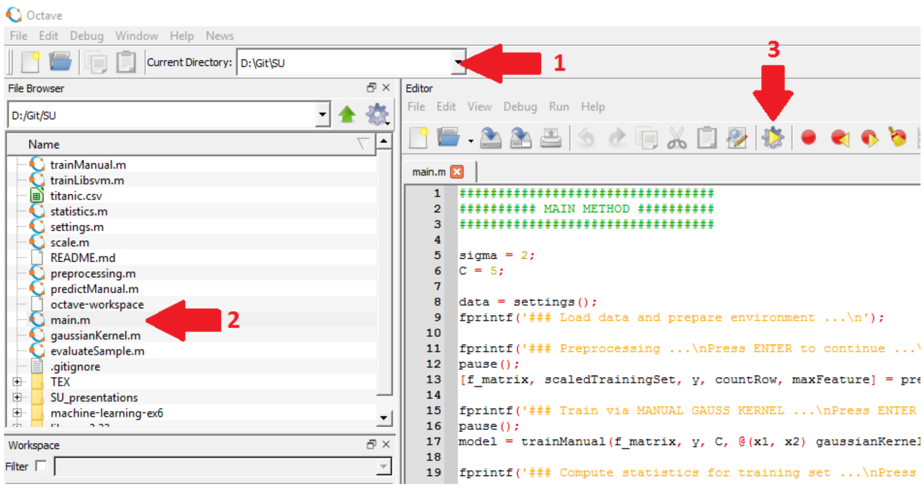
5.0.2 Načtení dat

Program na načítá data ze souboru *.csv*. Počítá také s tím, že data jsou ve stejné složce jako všechny potřebné moduly programu (*.m). Pokud jsou tedy data uloženy jinde, je nutné změnit cestu k souboru v modulu `settings.m` na řádce 25.

```
# settings.m
# line 25
data = csv2cell('titanic.csv');
```

5.0.3 Spuštění

Máme dvě možnosti spuštění:



Obrázek 4: 1. Nastavení cesty do složky programu, 2. Hlavní program (rozkliknout do záložky), 3. Spuštění programu

1. - Octave (GUI) : Máme-li Octave ve verzi s GUI, tak po spuštění programu navedeme Octave do správné složky s programem a spustíme modul `main.m` (viz Obr.:4)

2. - Octave (CLI) : Spouštíme-li Octave v příkazové řádce, je nutné se přesunout do složky programu s moduly a zadat pouze příkaz `main` (viz Obr.:5.

```
GNU Octave, version 4.2.0
Copyright (C) 2016 John W. Eaton and others.
This is free software; see the source code for copying conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type 'warranty'.

Octave was configured for "x86_64-w64-mingw32".

Additional information about Octave is available at http://www.octave.org.

Please contribute if you find this software useful.
For more information, visit http://www.octave.org/get-involved.html

Read http://www.octave.org/bugs.html to learn how to submit bug reports.
For information about changes from previous versions, type 'news'.

octave:1> cd D:\Git\SU
octave:2> main
```

Obrázek 5: Spuštění Octave (CLI)

Dále už průběh pokračuje v obou prostředí stejně. Uživatel stiskem klávesy `ENTER` spustí načtení a úpravu dat. Jakmile jsou data připravena, spustí se výpočet matice podobnosti indikovaný hláškou `Similarity ....` Dokončení výpočtu je značeno hláškou `... Done!`. Následovně je uživateli vyzván k potvrzení trénování hypotézy. Trénování začíná v okamžiku vyskočení hlášky `Training ...` a dokončení opět poznáme přes `... Done!`.

Program tedy přejde k výpočtu statistik a zobrazí uživatel v procentech, jak moc hypotéza natrénovaná.

Reference

[1] EKŠTEIN, Kamil. *Support Vector Machines* [online]. Plzeň, 2012 [cit. 2017-01-31]. Dostupné z: https://portal.zcu.cz/CoursewarePortlets2/DownloadDokumentu?id=123920. Přednášky k předmětu Strojové učení. Západočeská univerzita v Plzni.

[2] NG, Andrew. *CS229 Lecture notes - SVM* [online]. Standford, 2016 [cit. 2017-01-31]. Dostupné z: http://cs229.stanford.edu/notes/cs229-notes3.pdf. Přednášky k předmětu Machine Learning - CS229. Stanford University.

[3] NG, Andrew. *Support Vector Machines* [online]. Standford, 2016 [cit. 2017-01-31]. Dostupné z: https://d3c33hcgiwev3.cloudfront.net/_246c2a4e4c249f94c895f607ea1e6407_Lecture12.pdf?Expires=1485993600&Signature=MGhCSj6vnfSMVWpDERGUz8fc2312duxtjEpe2o4R0vhRA9KQQuPnZOPulQy5I0mCrICpxj3M0efelTXkmFFQsKB8&Key-Pair-Id=APKAJLTNE6QMUY6HBC5A. Přednášky k předmětu Machine Learning. Stanford University.

[4] Support vector machine. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2016 [cit. 2017-01-31]. Dostupné z: https://cs.wikipedia.org/wiki/Support_vector_machines.

[5] ZISSERMAN, Andrew. In: *SVM dual, kernels and regression* [online]. Oxford, 2015 [cit. 2017-01-31]. Dostupné z: http://www.robots.ox.ac.uk/~az/lectures/ml/lect3.pdf.

[6] KEERTHI,S.S.;SHEVADE,S. K.;BHATTACHARYYA,C.;MURTHY,K.R.K.. *Improvements to Platt's SMO Algorithm for SVM Classifier Design* [online]. Singapore, 2000 [cit. 2017-02-03]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.115.5266&rep=rep1&type=pdf.

[7] PLATT, John C.. *Sequential Minimal Optimization* [online]. Microsoft Research, 1998 [cit. 2017-02-03]. Dostupné z: https://pdfs.semanticscholar.org/8b5e/

`ab2c9fefe2fb1cc15e755cf7382ffc638f7c.pdf`. A Fast Algorithm for Training Support Vector Machines - Microsoft Research.