Fakulta matematiky, fyziky a informatiky Univerzity Komenského, Bratislava

Projekt z metód voľnej optimalizácie

Logistická regresia pomocou kvázinewtonovských metód – predikcia solventnosti klientov

Piati za optimalizáciu
Tomáš Antal, 2DAV, 0.23
Erik Božík, 2DAV, 0.23
Róbert Kendereš, 2DAV, 0.08
Teo Pazera, 2DAV, 0.23
Andrej Špitalský, 2DAV, 0.23

Obsah

0 Predstavenie témy **2**
0.1 Zavedenie značenia 2

1 Odvodenie účelovej funkcie a jej gradientu **3**
1.1 Kompaktnejší tvar účelovej funkcie 3
1.2 Gradient účelovej funkcie 4

2 Riešenie optimalizačnej úlohy **5**
2.1 Kvázinewtonovské metódy 5
2.2 Gradientné metódy 5

3 Vizualizácia konvergencie **7**
3.1 Vizualizácia konvergencie kvázinewtonovských metód 7
3.2 Vizualizácia konvergencie gradientných metód 8

4 Binárna klasifikácia solventnosti klientov **9**

5 Nadstavba - všeobecný model pre logistickú regresiu pomocou kvázinewtonovských alebo gradientných metód **10**

6 Zhrnutie **11**

7 Prehľad kódu **12**

0 Predstavenie témy

V našom projekte sa budeme venovať vytváraniu modelu na binárnu klasifikáciu dát. Na pozadí tohto modelu prebieha minimalizácia konkrétnej účelovej funkcie pomocou kvázinewtonovských alebo gradientných metód.

Na začiatku si odvodíme účelovú funkciu, odôvodníme jej tvar a odvodíme jej gradient. Následne ju budeme minimalizovať pomocou kvázinewtonovských a gradientných metód s rôznou dĺžkou kroku a vizualizujeme ich konvergenciu.

Taktiež otestujeme funkčnosť modelu na poskytnutých dátach o solventnosti klientov. O každom klientovi máme uvedený počet mesiacov od otvorenia účtu, pomer úspor a investícií, počet rokov v súčasnom zamestnaní a binárny údaj, či je klient solventný (1) alebo nie (0). My sa budeme snažiť predikovať solventnosť klienta na základe prvých troch údajov.

0.1 Zavedenie značenia

- $m = 699$ značí počet klientov, o ktorých máme dáta
- $v \in \mathbb{R}^m$, i-ta zložka má hodnotu 1, ak je klient i solventný, inak 0
- $u_j \in \mathbb{R}^m$, $j = 1, 2, 3$, vektory údajov o klientoch
  - u_1 – počet mesiacov od otvorenia účtu
  - u_2 – pomer úspor a investícií
  - u_3 – počet rokov v súčasnom zamestnaní
- v^i, u_j^i označujú i-te položky jednotlivých vektorov pre $i = 1, \ldots, m$, $j = 1, 2, 3$

1 Odvodenie účelovej funkcie a jej gradientu

V tejto časti sa budeme venovať odvodzovaniu účelovej funkcie a jej gradientu, ktorú v neskorších častiach budeme minimalizovať, pomocou čoho vytvoríme model na binárnu klasifikáciu.

Do logistickej funkcie $g(z) = \frac{1}{1+e^{-z}}$, ktorá bude odhadovať pravdepodobnosť solventnosti klienta, budeme dosádzať hodnoty $z = x^T u^i$ pre vektor parametrov $x = (x_0, \ldots, x_3)$ a vektor údajov o klientovi $u^i = (1, u_1^i, u_2^i, u_3^i)$, pre $i = 1, \ldots, m$.

Chceme odhadnúť zložky vektora x tak, aby čo najvierohodnejšie predpovedal solventnosť vzhľadom na naše dáta. To vedie k optimalizačnej úlohe:

$$\begin{aligned} &\min J(x) \\ &\quad x \in \mathbb{R}^4 \end{aligned}$$

kde

$$J(x) = -\sum_{i=1}^{m} v^i \ln\left(g\left(x^T u^i\right)\right) + (1 - v^i) \ln\left(1 - g\left(x^T u^i\right)\right) \tag{1}$$

Z predpisu funkcie si môžeme všimnúť, že suma nadobúda záporné hodnoty, čiže $J(x)$ nadobúda kladné hodnoty. Taktiež si môžeme všimnúť, že ak je klient i solventný, čiže $v^i = 1$ a pre nejaký vektor parametrov x je hodnota $g\left(x^T u^i\right)$ blízka nule, má to za následok „výrazné“ zvyšovanie hodnoty účelovej funkcie. Podobnou logikou vidíme zvyšovanie hodnoty účelovej funkcie pre nesolventných klientov, ak pomocou vektora x mu prisúdime veľkú pravdepodobnosť solventnosti hodnotou $g\left(x^T u^i\right)$. Chceme teda nájsť taký vektor x, že $g\left(x^T u^i\right)$ bude blízke 1 pre solvetného klienta a blízke 0 pre nesolventného.

1.1 Kompaktnejší tvar účelovej funkcie

Pre lepšiu manipuláciu a neskoršiu implementáciu si zjednodušíme tvar účelovej funkcie nasledovne:

$$\begin{aligned} J(x) &= -\sum_{i=1}^{m} v^i \ln\left(g\left(x^T u^i\right)\right) + (1 - v^i) \ln\left(1 - g\left(x^T u^i\right)\right) \\ &= -\sum_{i=1}^{m} v^i \ln\left(\left(1 + e^{-x^T u^i}\right)^{-1}\right) + (1 - v^i) \ln\left(\frac{e^{-x^T u^i}}{1 + e^{-x^T u^i}}\right) \\ &= -\sum_{i=1}^{m} -v^i \ln\left(1 + e^{-x^T u^i}\right) + (1 - v^i)\left(\ln\left(e^{-x^T u^i}\right) - \ln\left(1 + e^{-x^T u^i}\right)\right) \\ &= -\sum_{i=1}^{m} -v^i \ln\left(1 + e^{-x^T u^i}\right) - (1 - v^i) x^T u^i - (1 - v^i) \ln\left(1 + e^{-x^T u^i}\right) \\ &= \sum_{i=1}^{m} (1 - v^i) x^T u^i + \ln\left(1 + e^{-x^T u^i}\right) \end{aligned}$$

S takýmto vyjadrením funkcie $J(x)$ budeme pracovať v nasledujúcich častiach.

1.2 Gradient účelovej funkcie

Vyjadríme si najprv parciálnu deriváciu podľa x_0, potom podľa $x_j, j = 1, 2, 3$, keďže tie sa správajú symetricky.

$$\begin{aligned}
\frac{\partial}{\partial x_0} J(x) &= \frac{\partial}{\partial x_0} \sum_{i=1}^{m} (1 - v^i) x^T u^i + \ln\left(1 + e^{-x^T u^i}\right) \\
&= \sum_{i=1}^{m} \frac{\partial}{\partial x_0} \left((1 - v^i)(x_0 + x_1 u_1^i + x_2 u_2^i + x_3 u_3^i) + \ln\left(1 + e^{-x^T u^i}\right) \right) \\
&= \sum_{i=1}^{m} (1 - v^i) - \frac{e^{-x^T u^i}}{1 + e^{-x^T u^i}} \\
&= \sum_{i=1}^{m} 1 - v^i - \frac{1}{1 + e^{x^T u^i}}
\end{aligned}$$

$$\begin{aligned}
\frac{\partial}{\partial x_j} J(x) &= \frac{\partial}{\partial x_j} \sum_{i=1}^{m} (1 - v^i) x^T u^i + \ln\left(1 + e^{-x^T u^i}\right) \\
&= \sum_{i=1}^{m} \frac{\partial}{\partial x_j} \left((1 - v^i)(x_0 + x_1 u_1^i + x_2 u_2^i + x_3 u_3^i) + \ln\left(1 + e^{-x^T u^i}\right) \right) \\
&= \sum_{i=1}^{m} (1 - v^i) u_j^i - u_j^i \frac{e^{-x^T u^i}}{1 + e^{-x^T u^i}} \\
&= \sum_{i=1}^{m} \left(1 - v^i - \frac{1}{1 + e^{x^T u^i}}\right) u_j^i \qquad\qquad j = 1, 2, 3
\end{aligned}$$

Toto vieme kompaktne zapísať nasledovne:

$$\nabla J(x) = \sum_{i=1}^{m} \begin{pmatrix} 1 \\ u_1^i \\ u_2^i \\ u_3^i \end{pmatrix} \left(1 - v^i - \frac{1}{1 + e^{x^T u^i}}\right)$$

2 Riešenie optimalizačnej úlohy

V tejto časti sa venujeme riešeniu optimalizačnej úlohy 1 rôznymi metódami. Tie boli implementované v Pythone. Konkrétne sme implementovali gradientné metódy (s optimálnou a konštnantou dĺžkou kroku) a kvázinewtonovské metódy BFGS a DFP (s približne optimálnou dĺžkou kroku nájdenou backtracking-om alebo s optimálnou dĺžkou kroku, nájdenou bisekciou).

Ako štartovací bod sme pri každej metóde volili $x_0 = (0, 0, 0, 0)^T$ a ako kritérium optimality bolo použité $||\nabla J(x^k)|| \leq 10^{-3}$. Optimálnym bodom bude teda vektor parametrov x, ktorý budeme používať v logistickej funkcii na odhadovanie solventnosti klienta podľa jeho dát.

2.1 Kvázinewtonovské metódy

Minimalizujeme 1 pomocou metód BFGS a DFP s optimálnym krokom (nájdeným bisekciou) a približne optimálnym krokom (nájdeným backtrackingom).

	x_0	x_1	x_2	x_3
BFGS + backtracking	0.20751015	-0.04712048	0.31535175	0.30654686
BFGS + bisekcia	0.20751337	-0.04712051	0.31535088	0.30654664
DFP + backtracking	0.20750999	-0.04712047	0.31535176	0.30654688
DFP + bisekcia	0.20751338	-0.04712052	0.31535087	0.30654663

Všetky štyri minimalizácie skonvergovali k minimu (vzhľadom na kritérium optimality) za menej ako 13 iterácií. Vidíme, že optimálne hodnoty všetkých štyroch minimalizácií sa odlišujú najskôr v ráde 10^{-5}, čiže môžeme predpokladať, že konvergujú k rovnakému bodu minima.

Môžeme si všimnúť, že pozitívny vplyv na pravdepodobnosť solventnosti klienta má druhý sledovaný parameter, čiže pomer úspor a investícií, a tretí sledovaný parameter, čiže počet rokov v súčasnom zamestnaní. Takisto si môžeme všimnúť, že počet mesiacov od otvorenia účtu (prvý sledovaný parameter), má na odhad pravdepodobnosti solventnosti klienta negatívny vplyv, čo je prekvapivý výsledok.

Nájdený koeficient x_0 má za následok to, že pre klienta, ktorého parametre sú $(0, 0, 0)^T$, po dosadení do logistickej funkcie dostaneme pravdepodobnosť solventnosti približne 0.5517.

Môžeme si takisto porovnať čas (v sekundách) potrebných na nájdenie minima pre jednotlivé metódy.

	čas[s]
BFGS + backtracking	0.0067
BFGS + bisekcia	0.0074
DFP + backtracking	0.0031
DFP + bisekcia	0.0069

Vidíme, že pre obidve implementované kvázinewtonovské metódy je implementácia s približne optimálnou dĺžkou rýchlejšia.

2.2 Gradientné metódy

Podobne ako vyššie, minimalizujeme 1 pomocou gradientnej metódy s optimálnym a s konštantným krokom. Na nájdenie optimálneho kroku používame bisekciu, ako konštantný krok používame $2 \cdot 10^{-5}$.

	x_0	x_1	x_2	x_3
optimálny krok	0.20742273	-0.04711977	0.31535679	0.30656397
konštantný krok	0.19322267	-0.0470058	0.31617533	0.30934507

Gradientná metóda s optimálnym krokom skonvergovala (vzhľadom na kritérium optimality) po rádovo 5000 iteráciách. Gradientná metóda s konštantným krokom nedokonvergovala (vzhľadom na kritérium optimality) ani po 10000 iteráciách (neboli sme schopní nájsť experimentovaním vhodnú dĺžku kroku, pre ktorú by skonvergovala).

Signifikancia jednotlivých parametrov klientov je zhodná s tou, ktorá je popísaná vyššie. Takisto, pre klienta $(0, 0, 0)^T$ je odhad pravdepodobnosti solventnosti približne 0.5517 (pre optimálny krok), resp. 0.5482 (pre konštantný krok).

Rovnako ako vyššie, môžeme porovnať časy potrebné na minimalizáciu.

	čas[s]
optimálny krok	6.1772
konštantný krok	0.8142

Vidíme, že hľadanie optimálneho kroku v každej iterácií pridá približne 5.3 sekundy k času výpočtu, aj keď iterácií bolo rádovo polovica oproti konštantnému kroku. Skúsili sme preto nastaviť maximálny počet iterácií pre gradientnú metódu s konštatným krokom na 10^5. Už po rádovo 30000 iteráciách bolo dosiahnuté kritérium optimality a jeho nájdenie trvalo približne 0.9181 sekundy. Vidíme teda, že pri vysokom počte iterácií môže mať zmysel použiť skôr konštatný krok, keďže vieme výrazne ušetriť čas potrebný na hľadanie optimálneho kroku.

$$J^*_{GM\ const} = (0.20742016,\ -0.04711976,\ 0.31535694,\ 0.30656448)^T$$

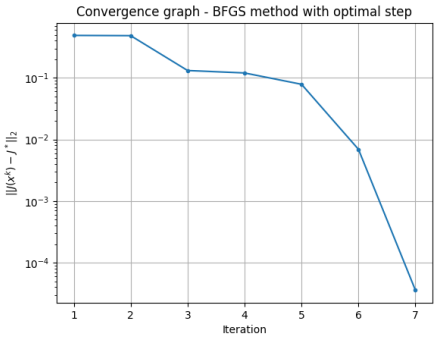
3 Vizualizácia konvergencie

Počas minimalizácie popísanej v predošlej sekcii sme taktiež ukladali body, cez ktoré metóda prechádzala. Ak označíme J^* nájdené minimum danou metódou, môžeme vizualizovať euklidovskú normu rozdielu $J(x^k) - J^*$, kde x^k je aproximácia minima v k-tej iterácii.

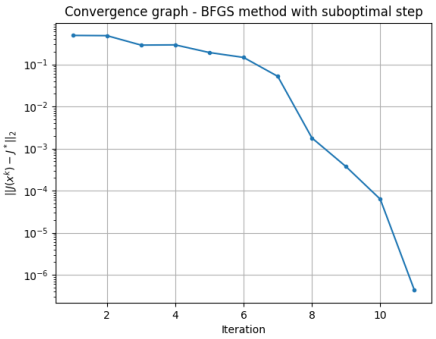
V nasledujúcich grafoch používame logaritmickú škálu na y-ovej osi, kde zobrazujeme $\|J(x^k) - J^*\|_2$. Na x-ovej je zobrazené poradové číslo iterácie. Bod x^k, v ktorom sa nadobúda hodnota J^* nie je zahrnutý v grafe, keďže by sme ho nevedeli vyobraziť na logaritmickej osi.

3.1 Vizualizácia konvergencie kvázinewtonovských metód

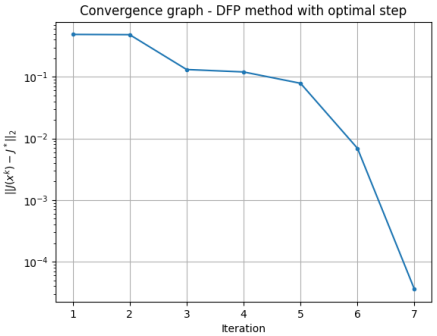
Môžeme si všimnúť, že kvázinewtonovské metódy našli aproximáciu minima za menej ako 13 iterácií. Taktiež môžeme podľa tvaru lomenej čiary odhadovať kvadratickú konvergenciu metód, no graf s takýmto malým počtom vykreslených iterácií nepodáva dostatočnú informáciu na istejší odhad.



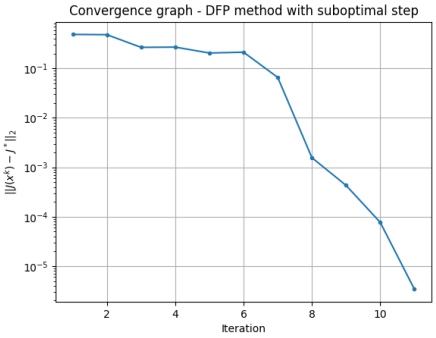
(a) BFGS s optimálnym krokom



(b) BFGS s približne optimálnym krokom



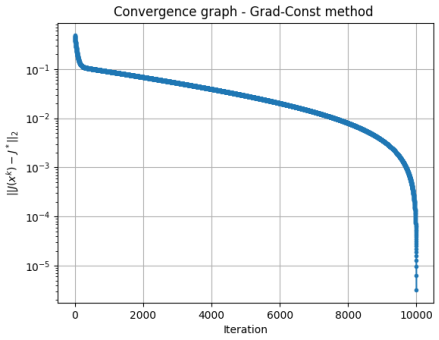
(c) DFP s optimálnym krokom



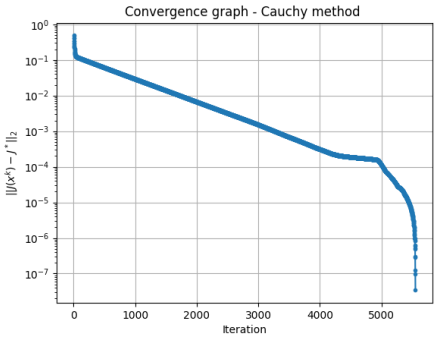
(d) DFP s približne optimálnym krokom

3.2 Vizualizácia konvergencie gradientných metód

Vidíme, že pri gradientných metódach je počet iterácií na nájdenie aproximácie minima rádovo vyšší (ako bolo spomenuté, gradientná metóda s konštantným krokom po 10000 iteráciách nenašla aproximáciu minima takú, ktorá by spĺňala kritérium optimality). Taktiež z grafov môžeme odhadnúť, že väčšiu časť minimalizácie konvergovali k optimu lineárne.



(a) konštantný krok



(b) optimálny krok

4 Binárna klasifikácia solventnosti klientov

Minimalizáciou funkcie 1 sme našli taký vektor koeficientov x, aby čo najkonzistentnejšie platilo, že po dosadení do logistickej funkcie bola hodnota $g(x^T u^i) \geq 0.5$ pre $v^i = 1$, inak chceme, aby platilo $g(x^T u^i) < 0.5$. Vektory u^i a hodnoty v^i, podľa ktorých bola funkcia 1 vytvorená a podľa ktorých sme našli vektor x, sú uložené v súbore `credit_risk_train.csv`.

Chceli by sme zistiť, či nájdený vektor x bude spĺňať vlastnosť popísanú vyššie aj pre také vektory $u^{i'}$ (i' značí, že sa už nejedná o vektory z `credit_risk_train.csv`), ktoré neboli zahrnuté v účelovej funkcii, teda neboli zohľadňované pri minimalizácii. Na to nám poslúžia dáta `credit_risk_test.csv`. Budeme postupne počítať hodnoty $g(x^T u^{i'}) =: p$, pričom ak $p \geq 0.5$, tak povieme, že náš odhad $v^{i'}$ je 1, inak 0.

Pre nájdené aproximácie miním všetkými 6 metódami (pri gradientnej metóde s konštantným krokom použijeme aproximáciu minima po 10000 iteráciách) vypíšeme podiel správnych predikcií hodnôt $v^{i'}$.

	podiel správnych predikcií $v^{i'}$
BFGS s optimálnym krokom	0.7209
BFGS s približne optimálnym krokom	0.7209
DFP s optimálnym krokom	0.7209
DFP s približne optimálnym krokom	0.7209
GM s optimálnym krokom	0.7209
GM s konštantným krokom	0.7176

Vidíme, že všetky metódy až na gradientnú s konštantným krokom majú zhodný podiel správnych predikcií $v^{i'}$. Je to pravdepodobne spôsobené tým, že ich aproximácie minima sú si navzájom veľmi blízke, čiže tento rozdiel sa nemusí prejaviť na pomerne malom množstve dát v `credit_risk_test.csv`. Môžeme teda zhodnotiť, že náš model na binárnu klasifikáciu mal pre tieto dáta 72% úspešnosť a vzhľadom na časovú efektivitu (spomenutú vyššie) je najvýhodnejšia implementácia pomocou jednej z kvázinewtonovských metód s približne optimálnym krokom.

5 Nadstavba - všeobecný model pre logistickú regresiu pomocou kvázinewtonovských alebo gradientných metód

Ako nadstavbu sme si zvolili implementovať všeobecný model na binárnu klasifikáciu, teda nie viazaný len na dáta o solventnosti. V neskorších projektoch by teda malo byť možné používať náš kód ako balík, z ktorého sa importuje trieda na binárnu klasifikáciu pomocou:

```
from logistic_regression import LogisticRegression
```

Následne, pomocou metódy `LogisticRegression.fit()` sa zistí vektor koeficientov x. Táto metóda očakáva trénovaciu maticu nezávislých vektorov `u`, vektor binárnych dát `v`, označenie metódy, ktorá má byť použitá na minimalizáciu účelovej funkcie (buď `"BFGS"`, `"DFP"`, `"Cauchy"` alebo `"Grad-Const"`) a v prípade použitia BFGS alebo DFP metódy očakáva aj požadovanú dĺžku kroku (buď `"optimal"` alebo `"suboptimal"`). Príklad použitia môže vyzerať takto:

```
log_reg = LogisticRegression()
log_reg.fit(u=u_train, v=v_train, method="DFP", step="optimal")
```

Vektor predikovaných $v^{i'}$ sa získa pomocou `LogisticRegression.predict()`, ktorá očakáva testovaciu maticu `u_test`, s rovnakým počtom stĺpcov, ako matica `u` (musí tomuto volaniu však predchádzať volanie metódy `fit()`):

```
log_reg.predict(u_test)
```

Konvergenčný graf sa generuje pomocou `LogisticRegression.visualize()`, ktorý očakáva ako argument inštanciu triedy `matplotlib.pyplot.Axes`:

```
import matplotlib.pyplot as plt
ax = plt.gca()
log_reg.visualize(ax)
```

Práca s touto triedou aj s minimalizačnými metódami je podrobne popísaná v dokumentácii v samotnom kóde.

Príklad použitia tohto modulu sa dá spustiť v terminálovom okne, ktoré je otvorené v priečinku `source` jedným z príkazov (podľa operačného systému, resp. nainštalovaného nástroja `make`):

```
python -m logistic_regression
python3 -m logistic_regression
make log_reg
```

6 Zhrnutie

V našom projekte sme sa venovali vytváraniu všeobecného modelu na binárnu klasifikáciu dát. Funkčnosť modelu sme ukázali na dátach o solventnosti klientov, kde sme dosiahli 72% úspešnosť predikcie. Náš kód je taktiež pripravený ako balík, z ktorého sa dá trieda `LogisticRegression` využiť na binárnu klasifikáciu. V budúcnosti by sme sa mohli venovať rôznym optimalizáciám najmä minimalizačnej časti modelu tak, aby efektívne zvládal aj veľké trénovacie dáta.

7 Prehľad kódu

V nasledujúcich riadkoch je popísaný obsah priečinku `source`, kde je lokalizovaný kód a dáta:

```
source
├── data
│   ├── credit_risk_train.csv - trénovacie dáta o solventnosti klientov
│   └── credit_risk_test.csv - testovacie dáta o solventnosti klientov
├── minimization_methods - metódy na minimalizáciu funkcií viac premenných
│   ├── gradient_descent.py - GM s konštantným a optimálnym krokom
│   ├── quasi_newton.py - KNM s optimálnym/približne optimálnym krokom
│   └── minimization_in _direction.py - bisekcia a backtracking
├── logistic_regression.py - trieda LogisticRegression
├── solvency_log_reg_results - priečinok s výsledkami minimalizácií
├── solvency_log_reg_results.py - skript spúšťajúci všetky minimalizácie
├── unit_tests.py - skript na otestovanie funkčnosti metód
├── visualizer.py - implementované vykreslenie konvergenčných grafov
└── Makefile - terminálové pravidlá na spúšťanie častí projektu
```