Ministerul Educaţiei, Culturii și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Departamentul Ingineria Software și Automatică

**RAPORT**

Lucru individual

Disciplina:Sisteme de inteligență artificială

Tema: Machine learning industrial: 10 principii de proiectare

A efectuat: st.gr. IA-182 ,

Ulmanu Cristian

A verificat : asist.univ. Stratulat Ștefan

Chișinău 2020

În zilele noastre, în fiecare zi sunt create noi servicii, aplicații și alte programe importante care vă permit să creați lucruri incredibile: de la software pentru controlul unei rachete SpaceX până la interacțiunea cu un ceainic în camera alăturată prin intermediul unui smartphone.

Și, uneori, fiecare programator începător, indiferent dacă este un start-up pasionat sau un Full Stack obișnuit sau Data Scientist, devine mai devreme sau mai târziu la realizarea că există anumite reguli pentru programare și dezvoltare de software care simplifică mult viața.

În acest articol, voi descrie pe scurt 10 principii ale modului de programare a învățării automate industriale, astfel încât să poată fi ușor integrată într-o aplicație / serviciu, pe baza metodologiei de aplicație cu 12 factori propusă de echipa Heroku. Inițiativa mea este de a crește gradul de conștientizare a acestei tehnici, care poate ajuta mulți dezvoltatori și oameni din Data Science.

Acest articol este prologul unei serii de articole despre machine learning. În ele, voi continua să vorbesc despre cum, de fapt, să fac un model și să îl rulez în producție, să creeze un API pentru acesta, precum și exemple din diverse domenii și companii care au ML încorporat în sistemele lor.

**Principiul 1: O bază de cod unică**

Unii programatori din etapele incipiente uită de Git din lene să realizeze (sau din anumite motive). Fie uită cuvântul, adică încarcă fișiere în unitate / doar trimit text / trimit porumbei, fie nu se gândesc la fluxul lor de lucru și fiecare se angajează la propria ramură și apoi la comandant.

Această regulă spune: aveți o bază de cod și multe implementări.

Git poate fi utilizat atât în ​​producție, cât și în cercetare și dezvoltare , unde este utilizat mai rar.

De exemplu, în faza de cercetare și dezvoltare, puteți lăsa angajamente cu diferite metode și modele de procesare a datelor și apoi alegeți-l pe cel mai bun și continuați să lucrați cu acesta.

În al doilea rând, în producție este de neînlocuit - va trebui să urmăriți în mod constant schimbarea codului dvs. și să știți ce model a dat cele mai bune rezultate, ce cod a funcționat și ce s-a întâmplat din cauza a ceea ce a încetat să funcționeze sau a început să emită rezultate incorecte. Pentru asta sunt commiturile!

Sau puteți crea un pachet pentru proiectul dvs., punându-l în Gemfury, de exemplu, și apoi pur și simplu importând funcții din acesta în alte proiecte, astfel încât să nu trebuie să le rescrieți de 1000 de ori, dar mai multe despre asta mai târziu.

**Principiul 2. Declarați și izolați în mod clar dependențele**

Fiecare proiect are biblioteci diferite pe care le importați din exterior pentru a le aplica undeva. Indiferent dacă este vorba de biblioteci Python sau biblioteci de alte limbi în scopuri diferite sau instrumente de sistem - sarcina dvs. este:

* Declarați în mod clar dependențele, adică un fișier care va conține toate bibliotecile, instrumentele și versiunile lor care sunt utilizate în proiectul dvs. și care trebuie instalate (de exemplu, în Python acest lucru se poate face folosind Pipfile sau requirements.txt. Link care permite bine înțeles: [realpython.com/pipenv-guide](https://realpython.com/pipenv-guide/) )
* Izolați dependențele special pentru programul dvs. în timpul dezvoltării. Nu doriți să modificați în mod constant versiunile și să reinstalați, de exemplu, Tensorflow?

Astfel, dezvoltatorii care se vor alătura echipei dvs. în viitor vor putea să se familiarizeze rapid cu bibliotecile și versiunile lor care sunt utilizate în proiectul dvs., precum și veți putea controla versiunile și bibliotecile în sine instalate pentru un anumit proiect, ceea ce vă va ajuta să evitați incompatibilitatea bibliotecilor sau a versiunilor acestora.  
  
De asemenea, aplicația dvs. nu trebuie să se bazeze pe instrumentele de sistem care ar putea fi instalate pe un anumit sistem de operare. Aceste instrumente trebuie, de asemenea, declarate în manifestul de dependențe. Acest lucru este necesar pentru a evita situațiile în care versiunea instrumentelor (precum și disponibilitatea acestora) nu se potrivește cu instrumentele de sistem ale unui anumit sistem de operare.  
  
Astfel, chiar dacă curl-ul poate fi utilizat pe aproape toate computerele, ar trebui să îl declarați în dependențe, deoarece atunci când migrați către o altă platformă este posibil să nu existe sau versiunea nu va fi cea de care aveți nevoie inițial.  
  
De exemplu, requirements.txt ar putea arăta astfel:

*# Model Building Requirements*

numpy>=1.18.1,<1.19.0

pandas>=0.25.3,<0.26.0

scikit-learn>=0.22.1,<0.23.0

joblib>=0.14.1,<0.15.0

*# testing requirements*

pytest>=5.3.2,<6.0.0

*# packaging* setuptools>=41.4.0,<42.0.0

wheel>=0.33.6,<0.34.0

*# fetching datasets*

kaggle>=1.5.6,<1.6.0

**Principiul 3. Configurații**

Mulți au auzit de povești în care diferiți dezvoltatori au încărcat accidental cod în GitHub în depozite deschise cu parole și alte chei de la AWS, trezindu-se a doua zi cu o datorie de 6.000 USD sau chiar cu toți 50.000 USD.

Desigur, aceste cazuri sunt extreme, dar foarte revelatoare. Dacă stocați acreditările sau alte date necesare pentru configurare în interiorul codului, faceți o greșeală și cred că nu merită să explicați de ce.  
  
Exemple de date care sunt de obicei stocate în variabile de mediu:

* Numele de domeniu
* URL-ul API / URI-urile
* Chei publice și private
* Contacte (poștă, telefoane etc.)

În acest fel, nu trebuie să schimbați codul tot timpul dacă variabilele de configurare se schimbă. Acest lucru vă va economisi timp, efort și bani.  
  
De exemplu, dacă utilizați API-ul Kaggle pentru a efectua teste (de exemplu, descărcați și rulați modelul prin el pentru a testa dacă modelul funcționează bine la pornire), atunci cheile private de la Kaggle, cum ar fi KAGGLE\_USERNAME și KAGGLE\_KEY, ar trebui să fie stocate în variabilele de mediu.

**Principiul 4: Servicii pentru terți**

Ideea aici este de a proiecta programul în așa fel încât să nu existe nicio distincție între resursele locale și terțe în ceea ce privește codul. De exemplu, puteți conecta atât MySQL local, cât și terță parte. Același lucru este valabil pentru diferite API-uri, cum ar fi Google Maps sau Twitter API.  
  
Pentru a dezactiva un serviciu terț sau a conecta altul, trebuie doar să schimbați tastele din configurația din variabilele de mediu, despre care am vorbit în paragraful de mai sus.  
  
Deci, de exemplu, în loc să specificați de fiecare dată calea către fișierele cu seturi de date din interiorul codului, este mai bine să utilizați biblioteca pathlib și să declarați calea către seturile de date în config.py, astfel încât, indiferent de ce serviciu utilizați (de exemplu, CircleCI), programul a reușit să afle calea către seturile de date, ținând cont de structura noului sistem de fișiere din noul serviciu.

**Principiul 5. Asamblați, lansați, runtime**

Mulți oameni din știința datelor consideră că este util să învețe abilități de scriere software. Dacă dorim ca programul nostru să se blocheze cât mai rar posibil și să ruleze fără probleme cât mai mult posibil, trebuie să împărțim procesul de lansare a noii versiuni în 3 etape:

1. *Etapa de asamblare*. Vă transformați codul gol cu ​​resurse separate într-un așa-numit pachet care conține toate codurile și datele necesare. Acest pachet se numește ansamblu.
2. Etapa de *lansare* - aici ne conectăm configurarea la asamblare, fără de care nu am putea lansa programul nostru. Acum este o versiune complet gata de lansare.
3. Urmează etapa de *execuție* . Aici lansăm aplicația pornind procesele necesare de la versiunea noastră.

Un astfel de sistem pentru lansarea de noi versiuni ale unui model sau a întregii conducte permite împărțirea rolurilor între administratori și dezvoltatori, permite urmărirea versiunilor și previne opririle nedorite ale programului.  
  
Pentru sarcina de lansare, au fost create multe servicii diferite în care puteți scrie procese pentru a vă rula într-un fișier .yml (de exemplu, în CircleCI acesta este config.yml pentru a susține procesul în sine). Wheely este minunat la crearea de pachete pentru proiecte.  
  
Veți putea crea pachete cu diferite versiuni ale modelului dvs. de învățare automată, apoi le pachetați și consultați pachetele necesare și versiunile acestora pentru a utiliza funcțiile pe care le-ați scris de acolo. Acest lucru vă va ajuta să creați un API pentru modelul dvs., iar pachetul dvs. poate fi plasat pe Gemfury, de exemplu.

**Principiul 6. Executăm modelul dvs. ca unul sau mai multe procese.**

**În** plus, procesele nu ar trebui să aibă date partajate. Adică, procesele trebuie să existe separat și toate tipurile de date trebuie să existe separat, de exemplu, pe servicii terțe, cum ar fi MySQL sau altele, în funcție de ce aveți nevoie.  
  
Adică, cu siguranță nu merită stocarea datelor în sistemul de fișiere al procesului, altfel poate duce la ștergerea acestor date în următoarea versiune / modificare a configurațiilor sau transferul sistemului pe care rulează programul.  
  
Există însă o excepție: pentru proiectele de învățare automată, puteți stoca memoria cache a bibliotecii, astfel încât să nu le reinstalați de fiecare dată când lansați o nouă versiune, dacă nu au fost făcute biblioteci suplimentare sau modificări ale versiunilor acestora. Astfel, veți scurta timpul de lansare al modelului dvs. în industrie.  
  
Pentru a rula modelul ca mai multe procese, puteți crea un fișier .yml în care să indicați doar procesele necesare și succesiunea acestora.

**Principiul 7. Reciclabilitate**

Procesele care rulează pe modelul din aplicația dvs. ar trebui să fie ușor de pornit și oprit. Astfel, vă va permite să implementați rapid modificări de cod, modificări de configurație, să scalați rapid și flexibil și să preveniți eventualele defecțiuni ale versiunii de lucru.  
  
Adică, procesul dvs. cu un model ar trebui:

* Minimizează timpul de pornire. În mod ideal, ora de pornire (de la momentul în care a fost emisă comanda de pornire până la momentul în care procesul intră în funcțiune) nu trebuie să depășească câteva secunde. Memorarea în cache a bibliotecii descrisă mai sus este o tehnică de reducere a timpului de pornire.
* Terminați corect. Adică, ascultarea portului de serviciu este de fapt suspendată, iar noile cereri trimise către acest port nu vor fi procesate. Aici trebuie să configurați o conexiune bună cu inginerii DevOps sau trebuie să înțelegeți cum funcționează (acesta din urmă, desigur, este de dorit, dar comunicarea trebuie menținută întotdeauna, în orice proiect!)

**Principiul 8: Implementare / integrare continuă**

Multe companii folosesc separarea între echipele de dezvoltare și implementare a aplicațiilor (punând aplicația la dispoziția utilizatorilor finali). Acest lucru poate încetini foarte mult dezvoltarea de software și poate progresa spre îmbunătățirea acestuia. De asemenea, strică cultura DevOps, unde dezvoltarea și integrarea sunt aproximativ combinate.  
  
Prin urmare, acest principiu afirmă că mediul dvs. de dezvoltare ar trebui să fie cât mai aproape de mediul dvs. de producție.  
  
Acest lucru va permite:

1. Reduceți de zece ori timpul de eliberare
2. Reduceți numărul de erori datorate incompatibilității codului.
3. De asemenea, ajută la reducerea volumului de muncă, deoarece dezvoltatorii și oamenii care implementează aplicația sunt acum o singură echipă.

Instrumentele care vă permit să lucrați cu acestea sunt CircleCI, Travis CI, GitLab CI și altele.  
  
Puteți face rapid adăugiri la model, îl puteți actualiza și lansa imediat, în timp ce va fi ușor, în caz de eșec, să reveniți foarte repede la versiunea de lucru, fără ca utilizatorul final să-l observe chiar. Acest lucru se poate face mai ales rapid și ușor dacă aveți teste bune.

**Principiul 9. Logs dvs.**

Logs (sau „Logs”) sunt evenimente înregistrate, de obicei în format text, care apar în cadrul aplicației (flux de evenimente). Exemplu simplu: „2020-02-02 - nivel sistem - nume proces”. Acestea sunt concepute astfel încât dezvoltatorul să poată vedea literalmente ce se întâmplă când programul rulează. El vede progresul proceselor și înțelege dacă este așa cum intenționează însuși dezvoltatorul.

Acest principiu spune că nu ar trebui să stocați Logs în sistemul de fișiere - trebuie doar să le „afișați”, de exemplu, să o faceți în ieșirea standard a stdout-ului sistemului. Și în acest fel, fluxul poate fi monitorizat în terminal în timpul dezvoltării.

Înseamnă asta că nu trebuie să salvați deloc Logs? Desigur că nu. Doar că aplicația dvs. nu ar trebui să facă acest lucru - lăsați-o în sarcina serviciilor terțe. Aplicația dvs. poate redirecționa Logs doar către un anumit fișier sau terminal pentru vizualizare live sau le poate redirecționa către un sistem de stocare cu scop general (cum ar fi Hadoop). Aplicația dvs. în sine nu ar trebui să stocheze sau să interacționeze cu Logs.

**Principiul 10. Test!**

Pentru ML industrială, această fază este extrem de importantă, deoarece trebuie să înțelegeți că modelul funcționează corect și oferă ceea ce doriți.Testele pot fi create cu pytest și testate cu un mic set de date dacă aveți o sarcină de regresie / clasificare.  
  
Nu uitați să setați aceeași sămânță pentru modelele de învățare profundă, astfel încât acestea să nu producă în mod constant rezultate diferite.Aceasta a fost o scurtă descriere a 10 principii și, desigur, este dificil să le folosești fără să încerci și să vezi cum funcționează, așa că acest articol este doar un prolog la o serie de articole interesante în care voi dezvălui cum să creezi modele industriale de învățare automată cum să le integrăm în sisteme și cum aceste principii pot face viața mai ușoară pentru noi toți.  
  
De asemenea, voi încerca să folosesc câteva linii directoare interesante pe care cineva le poate lăsa în comentarii dacă dorește.

**Bibliografie:**

1. [Resursă electronică]

<https://12factor.net/>

1. [Resursă electronică]

<https://medium.com/chingu/an-introduction-to-environment-variables-and-how-to-use-them-f602f66d15fa>

1. [Resursă electronică]

https://realpython.com/pipenv-guide/