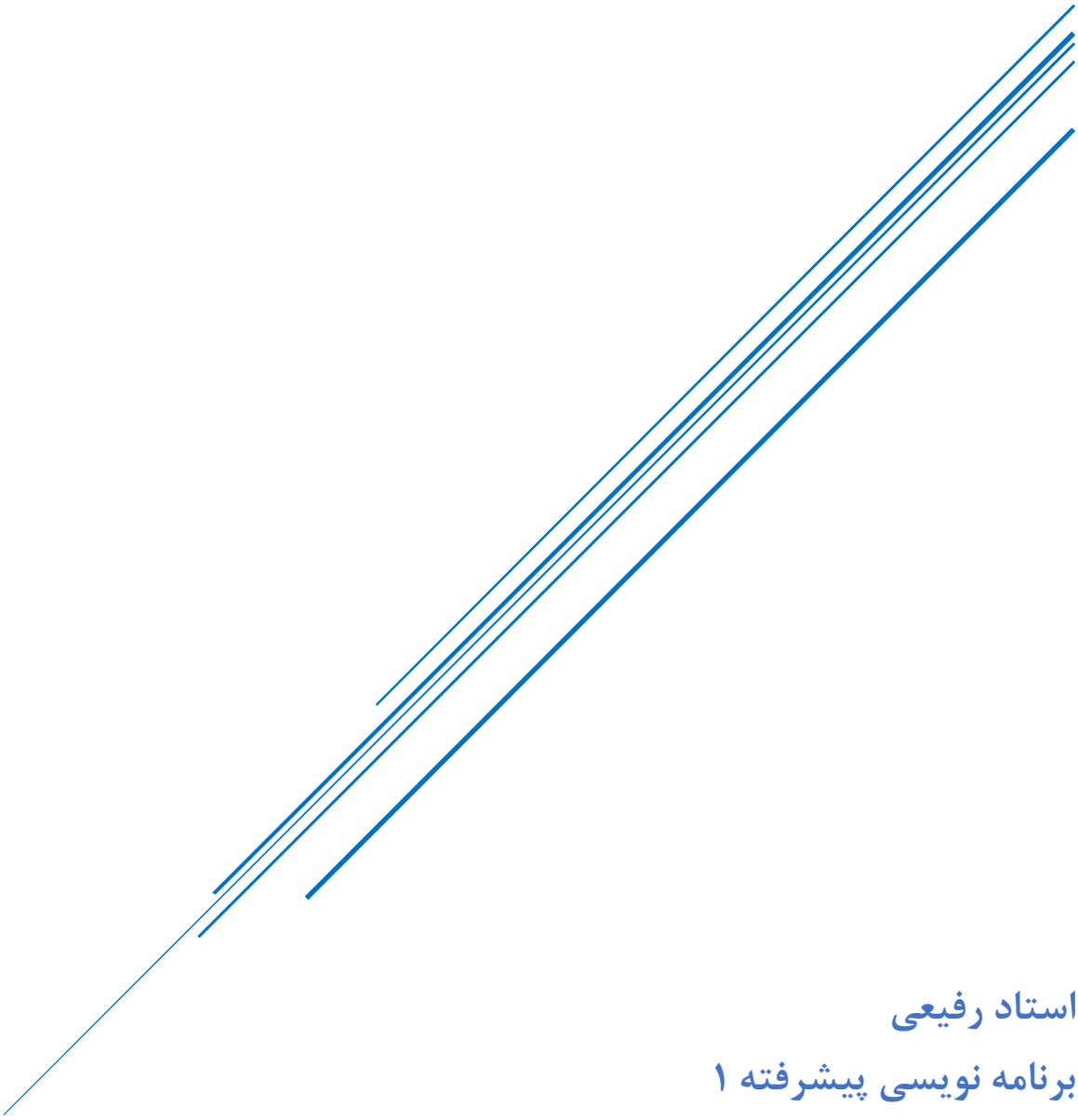
گزارش کار پروژه

خزشگر وب



استاد رفیعی

برنامه نویسی پیشرفته ۱

بخش های برنامه :

۱. پوشه ی FastTextFiles: شامل همه ی فایل های ذخیره و شده و اطلاعات همه ی کامنت ها و ...
۲. پوشه ی UserData: برای نگهداری از کوکی های مرورگر
۳. فایل chromedriver: درایوری که از آن درون برنامه استفاده شد
٤. فایل FastText: شامل کلاسی که بخش امتیازی برنامه در آن قرار داده شد
٥. فایل InstagramCrawler: شامل کلاسی که بخش خزش درون آن وجود دارد
٦. فایل GraphicMenu: شامل کلاسی که همه ی اشیای گرافیکی در آن تعریف شده و با اجرای آن ، برنامه به صورت گرافیکی باز میشود.

ویژگی های برنامه:

ویژگی های برنامه با نحوه ی ساخت و بخش های مختلف آن ، به صورت قدم ها نوشته شد که در ادامه مشاهده میکنید.

نتایج نمونه:

نتایج یک نمونه از اجرای برنامه ، در پوشه ی FastTextFiles موجود است.

قدم ها و تغییرات جز به جز برنامه:

نکته: قدم ها بر اساس تغییرات در فایل ها و گیت نوشته شده است ؛ در کل ۱۲ قدم ایجاد شد.

قدم ۱: کامیت Initialize

موارد:

۱. اضافه کردن فایل FastText
۲. آپدیت کردن کد های موجود در فایل InstagramCrawler
۳. اضافه کردن فایل step
٤. اضافه کردن chromedriver به برنامه

توضیحات:

در ابتدا ، کلاس و فایل InstagramCrawler ساخته شد ، سپس توابع مورد نیاز برای ست کردن درایور در صورت لاگین یا ورودی که از قبل وجود داشته است. اضافه کردن حالت تمام صفحه و ساختن پوشه برای نگه داشتن کوکی ها در همان پوشه ی برنامه نیز در ادامه اتفاق افتاد.

سپس تابعی برای پیدا کردن تعداد پست های مورد نیاز برای سرچ کردن در یک صفحه نوشته شد که بعدا نیاز به اسکرول اتفاق افتاد که بتوانیم پست هایی بیشتر از ۹ عدد (تعداد پیش فرض پست ها در یک صفحه) را اسکرپ کنیم. تابعی نیز برای آن استفاده شد. ولی با مشکل تعداد ثابت ۱۰ تا پست روبرو شدیم. یعنی هر بار که اسکرول میکرد ، پست های قبلی را پاک میکرد و قابل استفاده در آخر کار نبود.

سپس تابعی برای گرفتن تمامی کامنت ها در نظر گرفته شد ولی چون تمامی کامنت ها در اولین load شدن صفحه حضور نداشت ، باید روی دکمه ای که برای این کار در نظر گرفته شده بود کلیک میکردیم. سپس تابعی برای پیدا کردن تعداد پست های آخر یک اکانت را طراحی کردیم که مثل توابع قبلی کار میکرد.

اینجا با مشکل اسکرول نشدن و سریع اسکرول شدن مواجه شدیم. برای رفع این ها ، یک sleep در قسمت اسکرول کردن قرار دادیم که ابتدا یک ثانیه صبر میکرد و بعد اسکرول میکرد (که بعدا به اول اسکرول کند و بعد یک ثانیه صبر کند تغییر کرد) و برای اسکرول نشدن هم یک عدد در نظر گرفتیم که اگر این عدد ده بار متوالی یکسان بود ، پس دیگر اسکرول ندارد و آخر صفحه رسیده و باید از حلقه بیرون برود.

در کلاس FastText نیز ، ابتدا آن را زیر کلاس InstagramCrawler قرار دادیم که بتوانیم از آن استفاده کنیم. تابعی برای گرفتن هشتک ها و خزش آن ها در نظر گرفتیم که در آخر هر پست را نگاه کرده و کامنت های آن را ذخیره میکرد. سپس یک تابع برای پاک کردن کامنت ها نوشتیم. این تابع برای پاک کردن کاراکتر های اضافی مثل اموجی ها ، هشتک ها ، منشن ها و کاراکتر های غیر اسکی استفاده شد. که هر کدام یک تابع جدا داشت که با هم مخلوط نشده و کد تمیز تری داشته باشد. سپس برای تست چندین اکانت تست شد و کامنت های آن استخراج و در همان فایل به

صورت کامنت ذخیره شد و سپس همین دیکشنری ساخته شده را به توابع clean میدادیم که ساختار بهتری را بدهد و بفهمیم که چه چیز اضافه تری را باید از کامنت ها حذف کنیم.

قدم ۲: کامیت fix function

موارد:

۱. آپدیت کردن و درست کردن باگ های بعضی توابع
۲. دادن annotation و static کردن بعضی توابع که در دیگر فایل ها هم قابل استفاده شدن باشند.

توضیحات:

در این قدم ، در کلاس InstagramCrawler جای بعضی از متغییر ها و اسم آن ها تغییر کرد ، بعضی از توابع نیز تغییر نام دادند. همچین تابعی که برای پیدا کردن پست ها بود نیز تغییر کرد که هم اسم آن اکانت و هم آدرس آن را ذخیره کند.

بسیاری از متغییر ها هم به local variable تغییر داده شدند.

سپس تابعی که برای پیدا کردن تعداد پست های مورد نیاز برای سرچ کردن در یک صفحه نوشته شده بود آپدیت شد که بعدا نیاز به اسکرول اتفاق افتاد که بتوانیم پست هایی بیشتر از ۹ عدد (تعداد پیش فرض پست ها در یک صفحه) را اسکرپ کنیم. برای همین متغییری اضافه شد که با هر بار اسکرول کردن پست ها را پیدا کند و ذخیره کند. ولی این یک مشکل دیگر داشت که تعداد لینک های تکراری زیاد میشود. برای همین یک تابع هم برای پیدا کردن تکراری ها و حذف آن ها بر اساس order نوشته شد ، البته میتوانستیم از set هم استفاده کنیم ، ولی مشکل آن این بود که order را رعایت نمیکرد و اگر پست اول را میخواستیم ، نمیتوانستیم بفهمیم کدام است.

سپس یک سری تست در بدنه ی این کلاس نوشته شد که با اجرای مستقیم این کلاس اجرا میشدند و تعدادی پست و اکانت را پیدا کرده و نمایش میدادند.

قدم ۳: کامیت change method

موارد:

۳. تغییرات در توابع
٤. دادن annotation و static کردن بعضی توابع که در دیگر فایل ها هم قابل استفاده شدن باشند.
٥. عوض کردن sleep با WebDriverWait
٦. درست کردن xpath برای کلید های not now
۷. اضافه کردن کلاس های exception
۸. درست کردن لیست رشته ای که کامنت ها را نگه داری میکرد

توضیحات:

در این قدم ، در کلاس FastText ، زیر کلاس بودن از کلاس InstagramCrawler برداشته شد که که رابطه ی Has-a برقرار باشد. سپس یک متغییر اضافه شد به کلاس که یک شی از InstagramCrawler را نگه میداشت. به همین دلیل بقیه ی توابع هم آپدیت شدند که از متغییر های محلی یا پارامتر هایی که به آن ها میدادیم استفاده کنند. سپس نتایج بررسی هر قسمت پرینت میشد. بعد از آن یک تابع برای لیبل زدن دستی کامنت ها نوشته شد که بعد از خزش شدن همه ی کامنت ها ، از کاربر یکی یکی درخواست میکرد که برای هر کامنت لیبل بزند و در آخر هم لیبل های حذف شده و لیبل های زده شده را نشان میداد. سپس یک تابع دیگر اضافه شد که عبارت __label__ را در جلوی هر کامنت نوشته سپس نوع آن را بنویسد ، این کار برای ذخیره کردن و load کردن آن توسط کتابخانه ی fasttext مورد نیاز بود. سپس تابع cleaning نیز استاتیک شد.

توابعی هم برای ذخیره و load کردن اطلاعات در فرمت های مختلف نوشته شد که اگر به هر دلیلی در بین اجرا ، برنامه کرش کرد یا بسته شد یا ارور خورد ، مقادیر قبلی از بین نرود و خب برای خزش کردن ۱۵۰۰ کامنت ، این کاملا امری ضروری و درست بود و بسیار هم استفاده شد. همچنین هر قسمت پرینت نیز داشت که مطمئن باشیم اگر قسمت ذخیره کردن اجرا نشد ، حداقل پرینت شده ی مقادیر را داشته باشیم.

بعد از تست کردن های متخلف و فهمیدن باگ های مختلف ، کامنت های اضافی نیز حذف شد.

و سپس متادیتا برای همه ی توابع و متغییرها نوشته شد که نوع پارامتر و خروجی خود را نشان دهند.

در کلاس InstagramCrawler کتابخانه هایی که مورد نیاز بود ولی در FastText بود را به این کلاس انتقال داده و مواردی به نحوه ی استفاده از کوکی ها افزوده شد و م واردی نیز حذف شد. سپس یوزر نیم و پسورد نیز از بدنه ی تابع آن حذف و به صورت پارامتر در آمد. همچنین استفاده از sleep برای زمان دادن برای load شدن صفحات وب کار درستی نبود پس برای همین از خود درایور استفاده شد و این زمان را در اختیار آن گذاشت. به این صورت که اگر در زمان کمتری صفحه کامل load شد ، دیگر زمان بیشتری صبر نمیکند و ادامه ی کد را اجرا میکند. پس یک زمان بسیاری زیادی صرفه جویی میشود. همچنین exception های مختلف و نحوه ی هندل کردن آن ها افزوده شد.

قدم ٤: کامیت change method

موارد:

۱. درست کردن تابع در FastText
۲. ساخت یک دیتافریم
۳. ذخیره کردن دیتافریم
٤. اضافه شدن multithreading

توضیحات:

در این قدم ، در کلاس FastText یک تابع حلقه ی اشتباهی داشت که درست شد.

در کلاس InstagramCrawler نیز پارامتر متد ها کمی تغییر کرد و محدوده ی exception ها هم از ۱۰ به ۵ رسید. (تعداد خطای مجاز برای انجام یک عمل)

توابع نیز ملزم به استفاده ی متغییر های محلی شدند و نام متغییر ها هم تغییر کرد که درک بهتری از کد بدهد.

همچنین اینجا اولین قسمت multithreading اضافه شد که بعد از خزش کردن همه ی اکانت ها و ذخیره کردن html آن ها ، هر کدام را به صورت یک نخ جداگانه با bs٤ پیمایش کند.

این قسمت برای خزش کردن فقط نوشته شد چون درایور ، threading secure نبود و اگر دستوری روی آن اعمال میشد در همه ی تب ها انجام میشد و برنامه را عملا خراب میکرد.

سپس قسمت خزش یک صفحه اضافه شد که هر کامنت را با متن و اکانت و تعداد لایک آن به دست می آورد.

همچنین در اینجا تابعی برای ذخیره کردن مقادیر ساخته شد که دیتا فریمی از داده های موجود مثل یوزرنیم و کامنت و لایک های آن ساخته و سپس در فایل ذخیره میکرد.

قدم ۵: کامیت graphic part

موارد:

۱. ساختن بخش گرافیکی برنامه

توضیحات:

یک کلاس به نام GraphicMenu ساخته شد و مقادیر و اشیای داخل پنجره نوشته شد. همچنین برای ساختن برخی اشیا ، یک تابع در نظر گرفته شد که با گرفتن پارامتر هایی ، آن شی را میساخت. سپس تست شده و کارایی آن افزوده شد و قسمت های متخلف به آن اضافه شد تا تکمیل شود.

قدم ٦: کامیت graphic update

موارد:

١. آپدیت کردن بخش گرافیکی و ابجکت های آن
٢. درست کردن بخشی از بقیه ی کلاس ها
٣. حذف کردن static از برخی توابع برای راحتی کار
٤. آپدیت کردن قسمت __main__
٥. تابع برای ذخیره کردن train و valid
٦. درست کردن پرینت ها و اضافه کردن بخش های بیشتر
٧. اضافه کردن تابع fasttext

توضیحات:

در GraphicMenu ، توابعی برای ساخت یک دسته شی ساخته شد ، به عنوان مثال برای بیشتر کردن خوانایی کد ، همه ی لیبل ها در یک تابع جمع شد و با صدا زدن آن همه ی آن ها به صورت یکجا ساخته میشدند. همچنین متغییر های اضافی و اشیای جدید در آن قرار گرفت و رنگ و حال هوای آن نیز تغییر کرد. بخش های بیشتری به توابعِ ساخت اشیا اضافه شد ، همچنین آیکون نمایش داده شدن موس روی اشیا تغییر کرد. همچنین توابعی برای دکمه های + و – و open در نظر گرفته شد و نوشته شد.

در کلاس FastText ، پرینت ها بهتر شد و خوانایی اجرای برنامه را بالاتر برد ، همچنین تابع fasttext اضافه شد که یک مدل از فایل train ساخته و با valid مقایسه کند و نتیجه را چاپ کند. همچنین اسم بعضی متغییر ها و توابع تغییر کرد.

همچنین توابعی برای ذخیره کردن کامنت ها به صورت pkl ساخته شد و همچنین برای ذخیره کردن کامنت های خزش شده در فایل valid و train نیز تابعی نوشته شد که در خود برنامه استفاده شود.

همچنین در آخر فایل تست هایی اضافه شد و بعد از موفقیت آن ها ، کامنت شدند که ادامه ی برنامه قابل تست کردن باشد.

قدم ۷: کامیت migrate signup

موارد:

۱. Migrate کردن تابع get_driver و sign_up

توضیحات:

در کلاس InstagramCrawler ، برای کمتر کردن تعداد خطوط تکراری ، این دو تابع با هم ادغام و پارامتری به عنوان ورودی داده شد ، همچنین باگ برخی از توابع رفع شد که خطا ندهند.

همچنین توابع جایگزین شده با مقادیر جدید افزوده شد که پارامتر های خود را بگیرند.

قدم ۸: کامیت migrate signup

موارد:

۱. اضافه کردن progressbar
۲. آپدیت کردن کلاس FastText
۳. افزودن توابع به کلید های گرافیکی
٤. اضافه کردن لیبل نوتیفیکیشن
٥. اضافه کردن آبجکت های اضافی تر در گرافیکی
٦. وصل کردن checkbox ها
۷. مولتی تردینگ کردن کلید ها

توضیحات:

در کلاس InstagramCrawler ، progressbar افزوده شد که اگر ورودی آن را بگیرد ، روی آن اعمال کند ، به عنوان مثال آن را از ابتدا تعریف کند ، مقدار ان را خالی یا زیاد کند و ...

همچنین نام برخی کتابخانه ها خلاصه تر شد.

در کلاس گرافیکی ، کتابخانه های اضافه تری افزوده شد که متادیتا را کامل کند و خطا ها را هندل کند و همچنین قابلیت های کلید ها را به آن ها بدهد. همچنین نمودار اضافه شد.

قسمت State اضافه شد که کاربر مجبور باشد ترتیب انجام کار ها را رعایت کند. همچنین متغییر هایی برای نگه داشتن مقدار checkbox ها افزوده شد. فونت ها ، نوشته ها و ... بررسی و درست شد. لوکیشن آبجکت ها روی گرافیکی درست شد.

همچنین کامند هایی برای دکمه های اضافه شد. یک صفحه نیز برای هشتک ها و کشیدن نمودار از عملکرد کتابخانه fasttext ساخته شد.

در کلاس FastText تابع make model ساخته شد که مدل ساخته شود و ... همچنین اگر پیش بینی ای به آن داده شد بتواند انجام دهد. بقیه ی توابع نیز همگی آپدیت شدند و هندل کردن خطا ها به آن ها افزوده شد. همچنین تست های بیشتری افزوده شد.

قدم ۹: کامیت progressbar

موارد:

۱. اضافه کردن progressbar و تست عملکرد
۲. درست کردن مشکلات خزش ها با استفاده از گرافیکی

توضیحات:

در کلاس گرافیکی ، یک سری مقادیر تغییر کرد و جای آبجکت ها عوض شد. همچنین متن های ارور ها عوض شد. برای هر قسمت نیز یک State در نظر گرفته شد.

در کلاس InstagramCrawler ، لیبل های passed و left اضافه شد که مقدار زمان صرف شده و مقدار زمان مانده را حساب کند. همچنین تابعی نوشته شد که به جای تکرار کد در هر قسمت ، اعمال رو progressbar را با پارامتر های خود انجام دهد.

قدم ۱۰: کامیت fix progressbar

موارد:

۱. درست کردن مشکلات progressbar
۲. عوض کردن نسخه ی ۹۶ درایور به همه ی درایور ها
۳. برطرف کردن باگ های کوچک

توضیحات:

در این قدم ، کلاس FastText کد های تکراری حذف شد ، همچنین برای تست ، بخشی از موارد آن کامنت شد.

در کلاس GraphicalMenu ، بعضی آبجکت ها لوکیشن آن ها عوض شد ، باگ ها رفع شد.

در کلاس InstagramCrawler ، از اسم درایور ۹۶ حذف شد ، همچنین شرط برای وجود progressbar اضافه شد که در صورتی که برنامه به صورت کنسولی باز شد ، باگ نداشته باشد.

قدم ۱۱: کامیت FastText cleaned

موارد:

۱. Cleancode برای کلاس FastText
۲. کامنت گذاری و داکیومنت نویسی برای FastText
۳. رفع باگ ها

توضیحات:

در این قدم ، کلاس FastText ، کد ها توضیح داده شد ، داک نوشته شد ، توضیحیات و کامنت ها اضافه شد ، و آخرین متادیتا ها افزوده شد.

قدم ۱۲: کامیت all files

موارد:

۱. تمیز کردن همه ی فایل ها
۲. کامنت گذاری و داک نویسی برای همه ی فایل ها

توضیحات:

در این قدم ، تمامی کلاس ها ، ابتدا کتابخانه های آن ها مرتب شد ، سپس داک استرینگ نوشته شد و متادیتا ها آپدیت شد (در صورت لزوم) ، سپس کامنت گذاری شد.

اشیای گرافیکی نیز همه در توابع ساخت خود قرار گرفتند.