یادگیری ماشین مقیاس‌پذیر و کلان داده‌ها

تألیف و ترجمه:

پویا صبرآموز

۱۳۹۶

مؤسسه آموزشی تألیفی ارشدان

پیشگفتار

کتاب روبه‌رو حاصل ترجمه و تألیف کامل دو تکنولوژی بنیاد نرم‌افزارهای آزاد آپاچ شامل آپاچ ماهوت و آپاچ اسپارک می‌باشد. سعی شده در قسمت‌هایی که مستندات اصلی این تکنولوژی‌ها گویای مطلب اصلی نبوده مثال‌هایی عملی به آن اضافه کرده و آن مطلب را تکمیل کنم.
امیدوارم با تألیف و ترجمه این اثر بتوانم گام‌هایی هرچند کوچک در راه ارتقا دانش تخصصی علوم کامپیوتر برداشته باشم و در انجام رسالتی که بر عهده من است مؤثر واقع شود.
مثال‌ها و تمرین‌های موجود در این کتاب جهت دسترسی ساده‌تر و استفاده آسان همگی در وبگاه آن به نشانی www.tutiran.com موجود می‌باشد.
در خاتمه بر خود لازم می‌دانم از تمامی دوستان و اساتید که با کمک‌هایشان راهنمای من در این مسیر بودند و همچنین سرکار خانم مرضیه یادگار که زحمت طراحی جلد کتاب را بر عهده گرفتند تشکر و قدردانی می‌کنم.
بی‌شک این اثر نیز مانند باقی کتاب‌ها خالی از نقص نبوده است، لذا از تمامی دانش‌پژوهان گرامی و دانشجویان عزیز تقاضا داریم با ارائه نظرات ارزشمند خود ما را در رفع نواقص و کمبودها همراهی کنند.

پویا صبرآموز
info@sabramooz.ir

مقدمه

آینده بسیار از سازمان‌ها با توانایی‌شان به استفاده از فناوری اطلاعات وابسته است با توجه به رشد روزافزون داده‌ها و نیاز به ساخت سیستم‌های خود تصمیم‌گیرنده تصمیم گرفتیم از دو تکنولوژی روز بنیاد نرم‌افزارهای آزاد آپاچ یعنی آپاچ ماهوت و آپاچ اسپارک جهت پیاده‌سازی سیستم‌های یادگیری ماشین و آنالیز کلان داده استفاده کنیم.

در این کتاب در فصل نخست با ارائه مقاله‌ای درباره ضرورت و چگونگی و چرایی الگوریتم‌های ماشین یادگیرنده آشنا می‌شوید.

در ادامه شما می‌توانید با داشتن دانش نسبی روی زبان برنامه‌نویسی جاوا و هدوپ با سیستم‌های یادگیری ماشین و ۳ کاربرد اصلی آن یعنی ۱- سیستم‌های پیشنهاددهنده ۲- سیستم‌های طبقه‌بندی کننده و ۳- سیستم‌های خوشه‌بندی آشنا شوید.

در فصل سوم وارد حوزه تحلیل داده‌ای به وسیله آپاچ اسپارک شده و به تفصیل درباره آن صحبت خواهیم کرد.

در انتها نیز یک پروژه به وسیله هدوپ و با زبآنجاوا آورده شده است. چنان چه با هدوپ آشنایی ندارید بهتر است ابتدا کمی آن را مطالعه کرده و تمرین پیوست ۱ را انجام دهید سپس به ادامه مطالعه کتاب بپردازید.

قبل از هر چیز به دو تعریف کلی می‌پردازیم:

یادگیری ماشین به عنوان یکی از شاخه‌های وسیع و پرکاربرد هوش مصنوعی، یادگیری ماشین (Machine learning) به تنظیم و اکتشاف شیوه‌ها و الگوریتم‌هایی می‌پردازد که بر اساس آن‌ها رایانه‌ها و سامانه‌ها توانایی تعلم و یادگیری پیدا می‌کنند. الگوریتم‌های یادگیری ماشین به طور کلی به سه دسته تقسیم می‌شوند:

یادگیری نظارت شده: زمانی رخ می‌دهد که شما با استفاده از داده‌هایی که به خوبی برچسب‌گذاری شده‌اند به یک ماشین آموزش می‌دهید؛ به بیان دیگر در این نوع یادگیری، داده‌ها از قبل با پاسخ‌های درست برچسب‌گذاری شده‌اند.

یادگیری نظارت نشده: این نوع یادگیری زمانی رخ می‌دهد که ماشین با استفاده از داده‌هایی آموزش می‌بیند که هیچ‌گونه برچسب‌گذاری روی آن‌ها انجام نشده باشد.

یادگیری تقویت شده‌یا نیمه نظارت شده: این نوع یادگیری شباهت زیادی به نوع نظارت نشده دارد و وجه تشابهشان نیز در آن است که داده‌های مورد استفاده برای یادگیری برچسب‌گذاری نمی‌شوند، اما زمانی که پرسشی در مورد داده‌ها مطرح می‌شود، نتیجه درجه‌بندی خواهد شد.

کلان داده

کلان داده‌یا طبق مصوبه فرهنگستان؛ مه داده داده‌ای‌اند بسیار انبوه، پرشتاب و/یا گوناگون که نیاز به روش‌های پردازشی تازه‌ای دارند تا تصمیم‌گیری، بینش تازه و بهینگی پردازش پیشرفته را فراهم آورند و به عبارت کلی بتوانیم در کمترین زمان داده‌ها را به اطلاعات تبدیل کنیم.
وسعت کلان داده از چند ۱۰ ترابایت به چندین پتابایت در یک مجموعه داده می‌رسد. نمونه‌هایی از کلان داده چنین‌اند:
گزارش‌های وبی
سامانه‌های بازشناسی با موج‌های رادیویی
شبکه‌های حسگر
شبکه‌های اجتماعی
متن‌ها و سندهای اینترنتی
نمایه‌های جستجوهای اینترنتی
اخترشناسی
مدرک‌های پزشکی
بایگانی عکس
بایگانی ویدیو
پژوهش‌های زمین‌شناسی و بازرگانی در اندازه‌های بزرگ.

فهرست مطالب

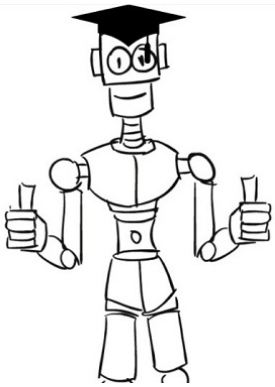
فصل اول

چرا ماشین یادگیرنده

یکی از حوزه‌های تکنولوژی که نقشی قابل توجه در بهبود سرویس‌های ارائه شده در تلفن‌های همراه و فضای مجازی دارد، یادگیری ماشینی است. گاهی اوقات دو عبارت یادگیری ماشینی و هوش مصنوعی به جای یکدیگر مورد استفاده قرار می‌گیرند و این مسئله به‌خصوص زمانی که یک شرکت بزرگ قصد دارد از جدیدترین نوآوری‌هایش سخن بگوید بیشتر به چشم می‌خورد، با این همه هوش مصنوعی و یادگیری ماشینی دو حوزه کاملاً مجزا و البته متصل به یکدیگر در علم کامپیوتر به شمار می‌روند.

از جمله اهداف هوش مصنوعی این است که بتواند رفتار ذهن انسان را تقلید کند که برای این منظور نیز ماشین نیازمند توانمندی‌های یادگیری است. با این همه هدف دانشمندان هوش مصنوعی کاملاً گسترده و جامع است و علاوه بر یادگیری، موارد دیگری شامل نمایش دانش، منطق و حتی اموری نظیر تفکر انتزاعی را نیز در بر می‌گیرد.

از سوی دیگر، یادگیری ماشینی صرفاً روی مقوله نوشتن نرم‌افزار تأکید دارد که می‌تواند از تجربیات گذشته درس بگیرد؛ اما نکته جالب‌تر در این رابطه آنکه یادگیری ماشینی در قیاس با هوش مصنوعی ارتباط نزدیک‌تری با کنکاش داده‌ها و تحلیل‌های آماری دارد. چرا این‌گونه است؟ بهتر است در ابتدا معنای یادگیری ماشینی را برای شما شرح دهیم.



یکی از تعاریف یادگیری ماشینی آن‌طور که از سوی تام میشل پروفسور دانشگاه کارنگی ملون ارائه گردید بدین شرح است: نوعی برنامه کامپیوتری که با توجه به برخی وظایف گروه Tو عملکردP ، تجربه E را شکل می‌دهد، اگر عملکرد آن در گروه وظایف T آن‌طور که توسط P اندازه‌گیری شده با تجربه E بهبود پیدا کند.

برای درک بهتر این تعریف بهتر است آن را به شکل ساده شده روبرو برایتان شرح دهیم: اگر یک برنامه کامپیوتری بتواند عملکرد خود در انجام یک وظیفه را با استفاده از تجربیات قبلی‌اش بهبود ببخشد آنگاه می‌توانید بگویید که آن ماشین یاد گرفته است.

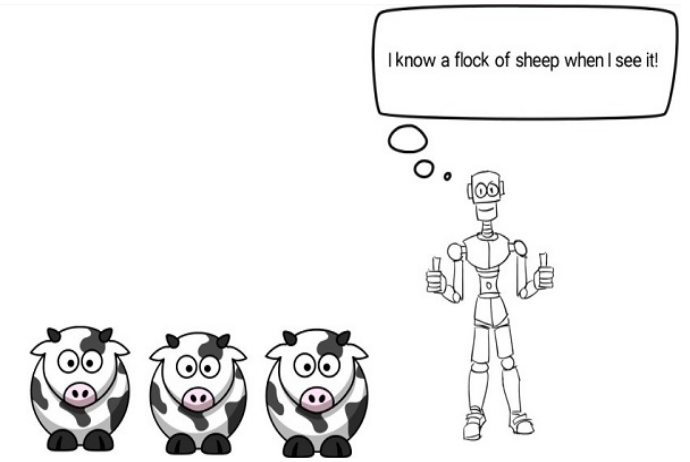
اما در پاره‌ای از موارد برنامه‌نویسان کلیه پارامترهای لازم برای انجام یک وظیفه را برای ماشین برنامه‌نویسی می‌کنند و داده‌های موردنیاز برای انجام آن را نیز در اختیارش قرار می‌دهند که این نوع عملکرد کاملاً با یادگیری ماشینی فرق دارد.

برای نمونه یک برنامه کامپیوتری می‌تواند بازی ایکس او را انجام دهد چون یک برنامه‌نویس کد مناسب و استراتژی برنده شدن را برای آن نوشته است با این همه برنامه‌ای که هیچ استراتژی از پیش تعریف شده‌ای برای این کار ندارد و تنها قوانین بازی و سناریوی پیروزی (اینکه شروط برنده شدن چیست) را می‌داند، باید بازی کردن را با تکرار و تمرین یاد بگیرد.

اما این مسئله صرفاً در مورد بازی‌ها به کار برده نمی‌شود و در مورد برنامه‌هایی که عملیات طبقه‌بندی و پیش‌بینی را انجام می‌دهند نیز صدق می‌کند. طبقه‌بندی فرایندی است که در آن یک ماشین می‌تواند چیزهای مختلف را با استفاده از یک دیتابیس (شامل اطلاعات بصری و داده‌های حاصل از اندازه‌گیری) تشخیص داده و آن‌ها را گروه‌بندی کند.

پیش‌بینی (که از آن تحت عنوان سیر بازگشت داده‌های آماری یاد می‌شود) زمانی رخ می‌دهد که یک ماشین بتواند ارزش یک چیز را براساس ارزش‌های قبلی حدس بزند (پیش‌بینی کند). برای

نمونه با در نظر گرفتن مجموعه‌ای از خصوصیات برای یک خانه، تصمیم می‌گیرد که ارزش آن براساس خانه‌های فروخته شده قبلی چقدر است.



با این توضیحات، به تعریف دیگری از یادگیری ماشینی می‌رسیم که در واقع همان استخراج دانش از داده‌هاست. در این تعریف شما با یک پرسش روبرو هستید و قصد دارید به آن پاسخ دهید و از طرفی، تصورتان این است که جواب در داخل داده‌ها قرار دارد و شاید به همین خاطر است که یادگیری ماشینی به داده‌های آماری و کنکاش آن‌ها مربوط می‌شود.

انواع یادگیری ماشینی

یادگیری ماشینی را می‌توان به سه گروه اصلی تقسیم کرد: نظارتی، غیر نظارتی و تقویت شده که تعاریف هر یک نیز به شرح زیر است.

یادگیری نظارت شده :زمانی رخ می‌دهد که شما با استفاده از داده‌هایی که به خوبی برچسب‌گذاری شده‌اند به یک ماشین آموزش می‌دهید؛ به بیان دیگر در این نوع یادگیری، داده‌ها از قبل با پاسخ‌های درست (نتیجه) برچسب‌گذاری شده‌اند. برای نمونه به ماشین عکسی از حرف Aرا نشان می‌دهید. سپس پرچم ایران که سه رنگ دارد را به آن نشان می‌دهید. یاد می‌دهید که یکی از رنگ‌ها قرمز است و یکی سبز و دیگری سفید. هرچه این مجموعه اطلاعاتی بزرگ‌تر باشد ماشین هم بیشتر می‌تواند در مورد موضوع یاد بگیرد.

پس از آنکه آموزش دادن به ماشین به اتمام رسید، داده‌هایی در اختیارش قرار داده می‌شوند که کاملاً تازگی دارند و قبلاً آن‌ها را دریافت نکرده. سپس الگوریتم یادگیری با استفاده از تجربیات

قبلی خود آن اطلاعات را تحلیل می‌کند. مثلاً حرف A را تشخیص می‌دهد و یا رنگ قرمز را مشخص می‌کند.

یادگیری نظارت نشده :این نوع یادگیری زمانی رخ می‌دهد که ماشین با استفاده از داده‌هایی آموزش می‌بینید که هیچ‌گونه برچسب‌گذاری روی آن‌ها انجام نشده. در این روش، هرگز به الگوریتم یادگیری گفته نمی‌شود که داده‌ها نمایانگر چه هستند. برای نمونه گفته می‌شود که اینجا یک حرف داریم اما هیچ‌گونه اطلاعاتی در مورد اینکه صحبت از کدام حرف است، به الگوریتم داده نمی‌شود یا در اینجا مشخصات پرچم را داریم اما نامی از پرچم به میان نمی‌آید.

یادگیری نظارت نشده همچون گوش دادن به یک فایل صوتی به زبانی است که نمی‌دانید؛ نه دیکشنری در اختیار دارید و نه حتی یک ناظر (معلم) که به شما بگوید در آن فایل صوتی چه حرف‌هایی گفته می‌شود. اگر تنها به یکی از فایل‌های صوتی ضبط شده به آن زبان گوش دهید چیز زیادی دستگیرتان نمی‌شود اما چنانچه صدها ساعت پای آن‌ها بنشینید مغزتان شروع به ایجاد نوعی الگو در مورد آن زبان می‌کند.

از این زمان به بعد شروع به تشخیص الگوها می‌کنید و به تدریج در حین گوش دادن به آن پادکست‌ها انتظار شنیدن اصوات خاصی را خواهید داشت. زمانی که یک دیکشنری در اختیارتان قرار داده شود یا اینکه از راهنمایی‌های یک مربی بهره‌مند شوید آنگاه با سرعت بیشتری شروع به یادگیری آن زبان خواهید کرد.

نکته کلیدی در مورد یادگیری نظارت نشده آن است که پس از پردازش اطلاعات بدون برچسب، تنها کافی است که یک نمونه از داده‌های برچسب‌گذاری شده در اختیار الگوریتم یادگیری قرار داده شود تا کارایی کامل پیدا کند.

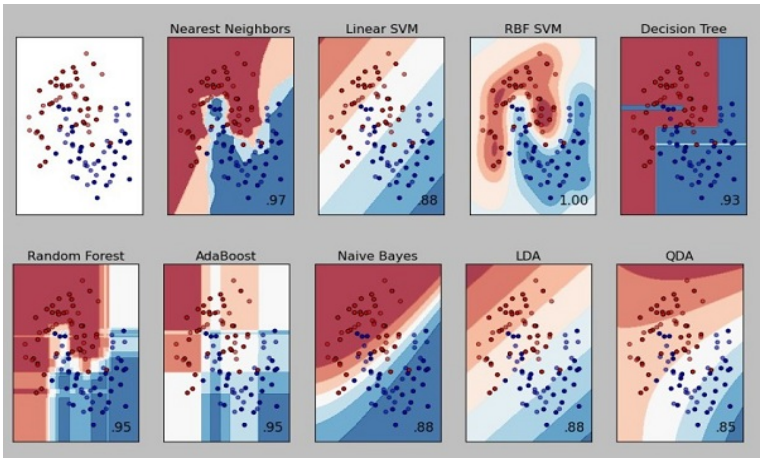
به عنوان مثال پس از پردازش هزاران عکس مربوط به حروف انگلیسی، تنها با پردازش حرفA ، بلافاصله یک بخش کامل از داده‌های پردازش شده برچسب‌گذاری می‌شوند. مزیت این روش آن است که به مجموعه کوچکی از داده‌های برچسب‌گذاری شده برای این کار نیاز است. ایجاد داده‌های برچسب‌گذاری شده نیز به مراتب سخت‌تر از داده‌های بدون برچسب است. به طور کلی همه ما به حجم انبوهی از داده‌های بدون برچسب دسترسی داریم و تنها بخش کوچکی از آن‌ها برچسب‌گذاری شده‌اند.

یادگیری تقویت شده :این نوع یادگیری شباهت زیادی به نوع نظارت نشده دارد و وجه تشابهشان نیز در آن است که داده‌های مورد استفاده برای یادگیری برچسب‌گذاری نمی‌شوند، با این همه زمانی که پرسشی در مورد داده‌ها مطرح می‌شود، نتیجه درجه‌بندی خواهد شد. یک مثال خوب

برای این نوع یادگیری انجام بازی است. اگر ماشین برنده بازی شود، سپس از نتیجه کار برای تقویت حرکات آتی خود در حین بازی بهره می‌گیرد.
مجدداً باید تأکید کنیم که اگر کامپیوتر تنها یک یا دو بار بازی را انجام دهد این روش تأثیری در عملکرد آن نخواهد داشت اما اگر هزاران بار و حتی میلیون‌ها بار این کار را انجام دهد آنگاه اثر کلی این کار باعث شکل‌گیری نوعی استراتژی پیروزی در آن می‌شود.

عملکرد یادگیری ماشینی چگونه است؟

مهندسانی که در زمینه ساخت سیستم‌های یادگیری ماشینی فعالیت دارند تکنیک‌های مختلفی را برای این منظور مورد استفاده قرار می‌دهند. همان‌طور که پیشتر گفته شد تعداد زیادی از این تکنیک‌ها به کنکاش داده‌ها و آمارها مربوط می‌شوند. برای نمونه، اگر مجموعه‌ای از اطلاعات را در اختیار داشته باشید که خصوصیات انواع گوناگونی از سکه (شامل وزن و شعاع) را تعریف کنند آنگاه می‌توانید از تکنیک‌های آماری نظیر الگوریتم «نزدیک‌ترین همسایه «برای طبقه‌بندی سکه‌ای که قبلاً مشاهده نشده استفاده نمایید.
کاری که الگوریتم «نزدیک‌ترین همسایه» انجام می‌دهد آن است که به دنبال طبقه‌بندی نزدیک‌ترین همسایه آن سکه می‌گردد و سپس همان طبقه‌بندی را برای آن سکه جدید نیز قائل می‌شود.
تعداد همسایه‌هایی که برای اتخاذ این تصمیم مورد استناد قرار گرفته‌اند با عنوان «K» شناخته می‌شود و بر همین اساس عنوان کامل برای الگوریتم به این شرح خواهد بود K» :همسایه نزدیک.«
با این همه الگوریتم‌های بیشمار دیگری نیز وجود دارند که سعی می‌کنند همین کار را با استفاده از متدهای متفاوتی انجام دهند. به نمودارهای زیر نگاه کنید:



تصویری که در بالا سمت چپ قرار دارد، مجموعه داده‌های موجود ما را نشان می‌دهد. این داده‌ها به دو گروه طبقه‌بندی شده‌اند: آبی و قرمز و باید بگوییم که کاملاً فرضی هستند با این حال می‌توانند نمایانگر هر چیزی باشند و از وزن و شعاع سکه‌ها گرفته تا تعداد گلبرگ‌های روی یک گیاه و اندازه آن‌ها را در بر بگیرند.

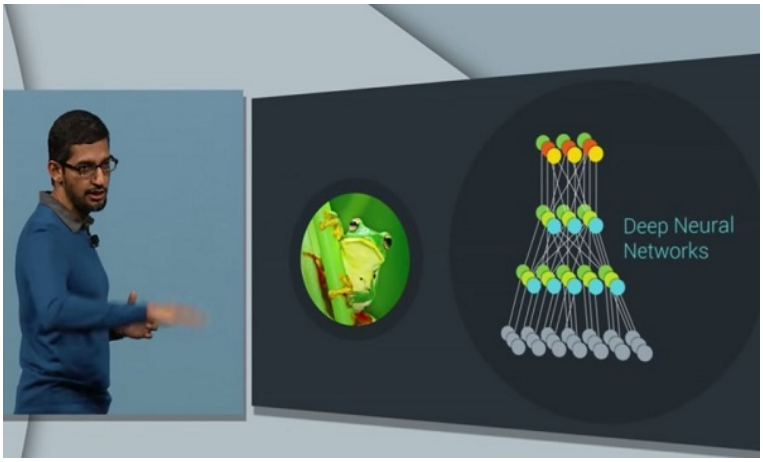
همان‌طور که مشاهده می‌کنید در این تصویر با چند گروه‌بندی قطعی نیز روبرو هستیم. هر آنچه که در گوشه سمت چپ در بالای تصویر قرار دارد در دسته قرمز جای می‌گیرد و هر آنچه که در پایین صفحه سمت راست قرار گرفته به گروه آبی تعلق دارد.

با این همه در میانه‌های تصویر شاهد نوعی تقاطع و یا به نوعی تداخل هستیم. اگر داده‌ای جدید را دریافت کنید که در میانه‌های این تصویر جای بگیرد آنگاه این سؤال پیش می‌آید که آن نمونه به گروه قرمز تعلق دارد یا آبی؟

تصاویر دیگر نیز الگوریتم‌های مختلف و نحوه گروه‌بندی نمونه‌های جدید از طریق آن‌ها را نشان می‌دهند. اگر داده جدید در یک منطقه سفید قرار بگیرد نمی‌توان آن را با استفاده از این متد طبقه‌بندی کرد. اعداد و ارقامی که در گوشه سمت راست در قسمت پایین عکس‌ها می‌بینید نیز دقت طبقه‌بندی را نشان می‌دهند.

شبکه‌های عصبی

یکی از اصطلاحاتی که به تناوب از سوی شرکت‌هایی نظیر گوگل و فیسبوک مورد استفاده قرار می‌گیرد «شبکه عصبی» است. یک شبکه عصبی در اصل نوعی تکنیک یادگیری ماشینی است که براساس نحوه عملکرد نورون‌های مغز انسان طراحی شده و از این ایده پیروی می‌کند که نورون‌ها پس از دریافت تعدادی داده ورودی، سیگنالی را برحسب تفسیر خود از آن اطلاعات پخش می‌کنند. در اصطلاحات رایج یادگیری ماشینی این کار از طریق دست‌کاری ماتریکس و همچنین نوعی تابع فعال‌سازی انجام می‌گیرد.



کاربرد شبکه‌های عصبی در سال‌های اخیر افزایشی چشمگیر داشته و هم‌اکنون نیز از این شبکه‌ها همراه با لایه‌های متعددی از نورون‌های متصل به هم استفاده می‌شود. در جریان کنفرانس Google I/O سال ۲۰۱۵، ساندار پیچای نایب رئیس بخش محصولات گوگل توضیح داد که چطور یادگیری ماشینی و شبکه‌های عصبی به این شرکت کمک کرده تا مأموریت اصلی خود یعنی سازمان‌دهی اطلاعات جهانی را به انجام رسانده و دسترسی به این اطلاعات را برای همه کاربران سطح دنیا فراهم نماید.

از همین روست که می‌توانید از Google Now سؤالاتی مانند این را بپرسید :در زبان اسپانیایی چطور می‌گویید قورباغه؟ و به خاطر همین شبکه‌های عصبی است که گوگل می‌تواند اموری نظیر تشخیص صدا، پردازش زبان‌های طبیعی و ترجمه را انجام دهد.

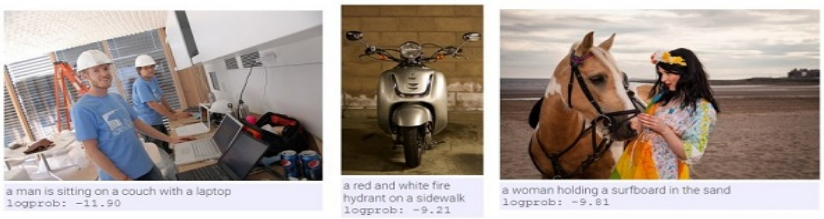
در حال حاضر گوگل از شبکه‌های عصبی ۳۰ لایه استفاده می‌کند که رقمی فوق‌العاده محسوب می‌شود و به خاطر استفاده از آن‌هاست که نرخ خطای تشخیص کلام گوگل از ۲۳ درصد در سال ۲۰۱۳ میلادی به ۸ درصد در سال ۲۰۱۵ کاهش پیدا کرد.

برخی از نمونه‌های یادگیری ماشینی

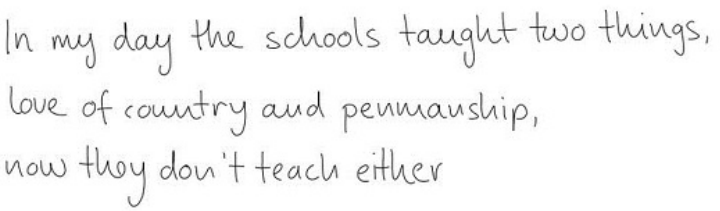
پس مشخص شد که شرکت‌هایی نظیر گوگل و فیسبوک از یادگیری ماشینی برای بهبود سرویس‌های خود بهره می‌گیرند. حال این سؤال مطرح می‌شود که این نوع یادگیری چه دستاوردهایی می‌تواند برای انسان داشته باشد؟ یکی از حوزه‌های جالب، حاشیه‌نویسی

عکس‌هاست. در این بخش تعدادی عکس در اختیار ماشین قرار داده شده و از آن خواسته می‌شود که آن‌ها را توصیف کند که در زیر می‌توانید نمونه‌هایی از آن‌ها را مشاهده نمایید.

توضیحات ارائه شده بری دو عکس نخست کاملاً درست هستند (هرچند که سینکی در داخل تصویر اول دیده نمی‌شود) و توضیحات ارائه شده برای عکس سوم نیز از آن جهت جالب است که کامپیوتر توانسته جعبه دونات ها را تشخیص دهد اما دیگر اجزای به نمایش در آمده در آن را به اشتباه فنجان قهوه تصور کرده.



نمونه دیگر این است که به کامپیوتر فرایند نوشتن یاد داده شودCleveland Amory ،. مؤلف، گزارشگر و صاحب‌نظر آمریکایی است که زمانی نوشت: «در دوران کودکی‌ام، مدرسه‌ها دو چیز را به ما یاد می‌دادند، نخست عشق به وطن و دیگری خوش‌نویسی اما این روزها خبری از این چیزها نیست» کاش می‌شد نظر آموری را در مورد دست خط زیر پرسید.



نمونه دست خط بالا توسط شبکه عصبی بازگشتی (نوعی شبکه عصبی مصنوعی که در آن، ارتباط میان واحدها نوعی چرخه هدفمند را تشکیل می‌دهد) ایجاد شده .خالقان این سیستم برای آموزش دادن به آن، از ۲۲۱ نویسنده درخواست کردند تا از یک تخته سفید هوشمند استفاده کرده و تعدادی متن را روی آن‌ها بنویسند.

در جریان فرایند نوشتن، نحوه قرارگیری قلم این افراد با استفاده از پرتو مادون‌قرمز دنبال می‌شد که این کار به شکل‌گیری مختصات X و Y انجامید و از آن برای آموزش نظارت شده بهره گرفته شد. همان‌طور که در تصویر می‌بینید نتایج فوق‌العاده بودند و حالا این ماشین قادر است به سبک‌های مختلف و با سطوح مختلفی از نامرتبی بنویسد.

گوگل به تازگی مقاله‌ای را در رابطه با شبکه‌های عصبی منتشر کرده و یادآور شده که از این شبکه به عنوان روشی برای الگوسازی مکالمات بهره می برد و پژوهشگران این شرکت در جریان آزمایشات خود با استفاده از ۶۲ میلیون جمله دریافت شده از زیرنویس تعدادی فیلم، به ماشین آموزش دادند.

همان‌طور که تصورش را خواهید کرد، نتایج فوق‌العاده‌اند. در بخشی از این آزمایش، ماشین مدعی می‌شود «از اینکه یک فیلسوفم احساس شرم نمی‌کنم!» و در ادامه زمانی که از آن در مورد اخلاقیات و اصول اخلاقی سؤال شد پاسخ داد» :حس و حال شرکت در یک بحث فلسفی را ندارم«.

بنابراین این‌گونه به نظر می‌رسد که اگر به صورت مستمر زیرنویس فیلم‌های هالیوودی را به یک ماشین بدهیم، می‌توانیم یک فیلسوف احساساتی را داشته باشیم.

نتیجه‌گیری

برخلاف بسیاری از حوزه‌های پژوهش در زمینه هوش مصنوعی، یادگیری ماشینی را نمی‌توان به عنوان یک هدف نامشهود در نظر گرفت؛ در واقع یادگیری ماشینی نوعی واقعیت است که هم‌اکنون برای بهبود سرویس‌های مورد استفاده انسان به کار گرفته می‌شود.

از بسیاری جهات می‌توان یادگیری ماشینی را نوعی ستاره فراموش شده در نظر گرفت که در پشت‌صحنه مشغول فعالیت است و همه تلاشش را به کار می‌گیرد تا پاسخ‌هایی که به دنبالشان هستیم را بیابد. (منبع -http://www.androidauthority.com/what-is-machine-learning /621659)

در ادامه دو تکنولوژی محبوب بنیاد آپاچ یکی در زمینه یادگیری ماشین مقیاس‌پذیر و دیگری در زمینه آنالیز اطلاعات بزرگ را به صورت تخصصی بررسی می‌کنیم:

ماهوت
اسپارک
لازم به ذکر است پیش‌نیاز این دو زبان برنامه‌نویسی جاوا و هدوپ می‌باشد.
در پیوست نیز یک نمونه پروژه هدوپ جهت آشنایی آورده شده است، در صورتی که با تکنولوژی‌های فوق آشنایی ندارید ابتدا آن‌ها را تا سطح متوسط بیاموزید سپس ادامه این کتاب را بخوانید.

✍ درباره ماهوت (apache mahout)

ماهوت یک پروژه متن باز است که به طور مشخص برای معرفی الگوریتم‌های ماشین یادگیرنده‌ی با قابلیت مقیاس‌پذیری بالا به کار می‌رود.
در این آموزش شما یاد می‌گیرید چگونه یک سیستم Recommendation یا پیشنهاددهنده بسازید و یا مستندات خود را به صورت کاربردی‌تر در خوشه‌ها (Clusters) مرتب کنید.

✍ پیش‌نیازها

قبل از شروع این آموزش لازم است شما تجربه کار با زبان JAVA، Hadoop و یکی از توزیع‌های سیستم عامل لینوکس را داشته باشید.

فصل دوم

معرفی

در روزگاری زندگی می‌کنیم که اطلاعات بـه وفـور در دسـترس مـا وجـود دارد. سـر بـار اضـافی اطلاعات گاهی به اندازه‌ای می‌رسد که مدیریت یک صـندوق ایمیـل سـاده هـم مشـکل بـه نظـر می‌رسد! حال شما اندازه داده‌ها و اطلاعات سایت‌هایی مانند facebook، twitter و youtube را در یک روز که باید جمع‌آوری و مدیریت شوند، تصور کنید.

حتی برای سایت‌های کوچک‌تر و کمتر شناخته شده هم این انبوه اطلاعات وجود دارد.

به طور معمول ما در این موقع برای آنالیز این اطلاعـات و تشـخیص Trend هـای آن‌هـا و رسـم نتایج، به سراغ الگوریتم‌های داده‌کاوی می‌رویم.

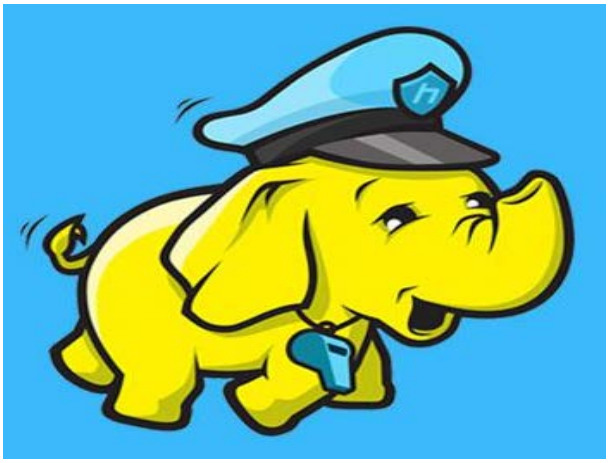
به هر حال هیچ الگوریتم داده‌کاوی نمی‌تواند برای پردازش مجموعـه عظیمـی از داده‌هـا و فـراهم کـردن نتایج در زمان مناسب کافی باشد، مگر این‌که وظیفه محاسبات در چند ماشین توزیع شده در فضای ابری محول شود.

ما در حال حاضر چارچوب‌های جدید داریم که به ما اجازه شکستن وظایف محاسباتی به چندین بخش و اجرا شدن هر بخش روی یک ماشین را می‌دهد.

ماهوت mahout یک چارچوب داده‌کاوی است که معمولاً به همراه زیر ساخت هـدوپ (Hadoop) اجرا می‌شود که وظیفه هدوپ مدیریت کردن مقادیر عظیم داده می‌باشد.

ماهوت آپاچی یا Apache Mahout چیست؟

ماهوت کسی است که استادانه سوار فیل شده و با آن حرکت می‌کند. دلیل انتخاب این نام این است که نماد هدوپ یک فیل است و ماهوت تقریباً همیشه روی هدوپ اجرا می‌شود.



هدوپ یک چارچوب متن باز برای آپاچی است که با استفاده از مدل‌های برنامه‌نویسی ساده اجازه ذخیره و پردازش Big data یا ابر داده را در محیط‌های توزیع شده و بین خوشه‌هایی از کامپیوترها فراهم می‌کند.

ماهوت آپاچی یا ماهوت یک پروژه متن باز است که عموماً بـرای سـاخت الگـوریتم‌هـای ماشـین یادگیرنده به صورت مقیاس‌پذیر یا scalable به کار می‌رود.
به وسیله ماهوت می‌توانیم تکنیک‌هـای یـادگیری ماشـین یـا macine learning محبـوب زیـر را پیاده‌سازی کنیم:
سیستم‌های پیشنهاددهنده یا Recommendation
طبقه‌بندی کردن یا Classification
خوشه‌بندی کردن یا Clustering
ماهوت آپاچی در سال ۲۰۰۸ و به عنوان یک زیر پـروژه از پـروژه Lucene بـود. در سـال ۲۰۱۰ ماهوت به یکی از پروژه‌های سطح بالای آپاچی تبدیل شد.

ویژگی‌های ماهوت

ویژگی‌ها و مشخصات اصلی ماهوت در زیر آمده‌اند:
الگوریتم‌های ماهوت روی (بالای) هدوپ نوشته می‌شوند، بنابراین به خوبی در محیط‌های توزیع شده کـار می‌کند. ماهوت از کتابخانه هدوپ برای بالا بردن کارایی در محیط ابری استفاده می‌کند.
ماهوتcoder را که یک فریم ورک یـا چـارچوب (و آمـاده بـرای اسـتفاده) بـرای انجـام وظـایف داده‌کاوی روی قسمت‌های بزرگ داده است را پیشنهاد می‌دهد.
ماهوت به برنامه‌ها امکان آنالیز کارایی مجموعه‌های بزرگ داده در زمان سریع را می‌دهد.
شامل چندین پیـاده‌سـازی MapReduce فعـال روی خوشـه ماننـد k-means،fuzzy k-means ، Canopy، Dirichlet و Mean-Shift می‌باشد.
ماهوت از پیاده‌سازی‌های طبقه‌بندی Naive Bayesتوزیع شده و Naive Bayes مکمل پشتیبانی می‌کند.
ماهوت به منظور تحقق برنامه‌نویسی تکاملی، با قابلیت توابع توزیع شده شایسته و سازگار آمده است.
ماهوت شامل کتابخانه‌های ماتریکس (Matrix) و بردار (Vector) می‌باشد.

برنامه‌های استفاده‌کننده از ماهوت

شـرکت‌هـای بزرگـی ماننـد Adobe، Facebook، LinkedIn، Foursquare، Twitter و Yahoo از ماهوت استفاده می‌کنند.

Foursquare به شما برای پیدا کردن مکان‌ها، رستوران‌ها و محیط‌های تفریحی در یک منطقه مشخص کمک می‌کند. فور اسکوئر برای این کار از موتور پیشنهاددهنده یا recommender engine ماهوت استفاده می‌کند.

Twitter برای مدل‌سازی علایق کاربران از ماهوت استفاده می‌کند.

Yahoo برای الگو کاوی یا pattern mining از ماهوت استفاده می‌کند.

فصل سوم

یادگیری ماشین (Machine Learning)

ماهوت یک کتابخانه یادگیری ماشین کاملاً مقیاس‌پذیر و قابل گسترش است که به برنامه‌نویسان برای استفاده از الگوریتم‌های بهینه کمک می‌کند. ماهوت تکنیک‌های یادگیری ماشین محبوبی مانند سیستم‌های پیشنهاددهنده، طبقه‌بندی و خوشه‌بندی را پیاده‌سازی می‌کند؛ بنابراین لازم است که قبل از این‌که جلوتر برویم کمی از یادگیری ماشین صحبت کنیم.

ماشین یادگیرنده یا یادگیری ماشین (Machine Learning) چیست؟

یادگیری ماشین شاخه‌ای از علم است که با برنامه‌نویسی سیستم‌ها سروکار دارد به نحوی که سیستم‌ها بتوانند به صورت خودکار یاد بگیرند و تجربه خود را بهبود ببخشند. در اینجا یادگیری به معنای تشخیص و فهم داده‌های ورودی و اتخاذ تصمیمات عاقلانه بر پایه داده‌های قبلی می‌باشد.

خیلی سخت است که بتوانید تمام تصمیم‌ها را روی تمام ورودی‌های ممکن فراهم کنید. برای حل این مشکل، الگوریتم‌ها گسترش پیدا کردند. این الگوریتم‌ها دانش را با استفاده از داده‌های خاص و تجربه گذشته و استفاده از اصول آماری، نظریه احتمال، منطق، بهینه‌سازی ترکیبی، جستجو، یادگیری تقویتی (reinforcement learning) و نظریه کنترل می‌سازند.

این الگوریتم‌های گسترش یافته حالت پایه برنامه‌های مختلفی هستند مانند:

پردازش بصری (Vision Processing)

پردازش زبان (Language processing)

پیش‌بینی (مثلاً ترند بازار سرمایه) (Forecasting)

تشخیص الگو (Pattern recognition)

بازی‌ها (Games)

داده‌کاوی (Data mining)
سیستم‌های خبره (Expert systems)
رباتیک (Robotics)
یادگیری ماشین یک فضای وسیع است که صحبت از آن در این آموزش نمی‌گنجد. برای پیاده‌سازی تکنیک یادگیری ماشین راه‌های مختلفی وجود دارد، اما دو راه خیلی متداول یادگیری نظارتی یا supervised learning و یادگیری غیر نظارتی یا unsupervised learning می‌باشد.

Supervised Learning یا یادگیری نظارتی

یادگیری نظارتی یا supervised learning بر پایه‌یادگیری یک تابع از طریق داده‌های آموزشی موجود می‌باشد.
الگوریتم یادگیری نظارتی داده‌های آموزشی را آنالیز کرده و تابع استنتاج را معرفی می‌کند که می‌تواند برای map کردن مثال‌های جدید از آن استفاده شود.
متداول‌ترین مثال‌های یادگیری نظارتی:
طبقه‌بندی ایمیل‌ها به عنوان اسپم
نام‌گذاری صفحات وب براساس محتوای آن‌ها
تشخیص صدا
الگوریتم‌های یادگیری نظارتی بسیاری وجود دارند مانند شبکه‌های عصبی، ماشین‌های پشتیبان بردار (Support Vector Machines SVMs) و طبقه‌بندی کننده Naive Bayes. ماهوت طبقه‌بندی کننده Naive Bayes را پیاده‌سازی می‌کند.

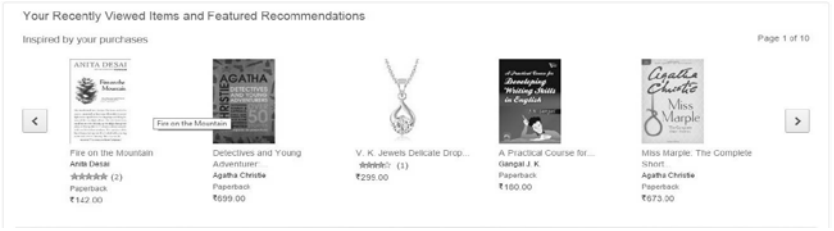
Unsupervised Learning یا یادگیری غیر نظارتی

یادگیری غیر نظارتی یا unsupervised learning داده‌های بدون لیبل یا unlabeled data را بدون داشتن هیچ مجموعه داده‌ای جهت آموزش، حس می‌کند.
یادگیری غیر نظارتی ابزاری به شدت قدرتمند برای آنالیز داده‌های موجود و دیدن الگوها و ترند ها است. از این نوع برای خوشه‌بندی ورودی‌های شبیه هم درون گروه‌های منطقی استفاده می‌شود. راه‌های متداول‌ترین راه‌های یادگیری غیر نظارتی:
k-means
نقشه‌های خود مرتب کننده (self organizing maps)

خوشه‌بندی سلسله مراتبی (hierarchical clustering)
در ادامه با ۳ تکنیک محبوب و توضیح آن‌ها آموزش ماهوت را ادامه خواهیم داد.

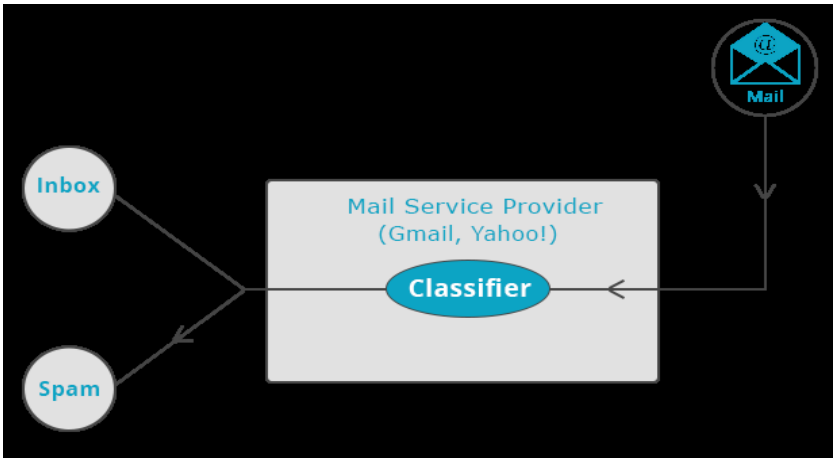
✍ پیشنهاد یا Recommendation

پیشنهاد یا Recommendation از محبوب‌ترین تکنیک‌های یادگیری ماشین است که پیشنهادات نزدیک را بر اساس اطلاعات کاربر نظیر خریدهای قبلی، کلیک‌ها و رتبه‌بندی‌ها ارائه می‌کند.
سایت آمازون از این تکنیک برای "نمایش لیست آیتم‌های پیشنهادی که ممکن است به آن‌ها علاقه داشته باشید"، رسم اطلاعات از فعالیت‌های گذشته شما استفاده می‌کند. این موتورهای پیشنهاددهنده در پس‌زمینه آمازون و به منظور ضبط رفتار کاربر و پیشنهاد آیتم‌های انتخابی براساس رفتار قبلی کار می‌کنند.
فیس بوک از تکنیک پیشنهاددهنده (Recommeder) برای تشخیص و پیشنهاد در قسمت "people you may know list" یا افرادی که احتمالاً می‌شناسید، استفاده می‌کند.



✍ طبقه‌بندی یا Classification

طبقه‌بندی یا Classification با عنوان رده‌بندی یا categorization هم شناخته می‌شود که یک تکنیک یادگیری ماشین می‌باشد که از داده‌های شناخته شده برای تشخیص این‌که داده‌های جدید در چه گروهی قرار بگیرند، استفاده می‌شود. طبقه‌بندی یکی از حالت‌های یادگیری نظارتی یا Supervised Learning محسوب می‌شود.
سرویس‌های ایمیل نظیر یاهو و Gmail از این تکنیک برای تشخیص اسپم بودن یک ایمیل جدید استفاده می‌کنند. این الگوریتم طبقه‌بندی خودش را با آنالیز عادت‌های کاربر که کدام ایمیل‌ها را اسپم تشخیص می‌دهد، آموزش می‌دهد. براساس این آموزش طبقه‌بندی کننده تصمیم می‌گیرد که آیا ایمیل بعدی باید در Inbox قرار گیرد یا در Spam.
برنامه Itunes شرکت اپل از طبقه‌بندی برای ایجاد playlist ها استفاده می‌کند.



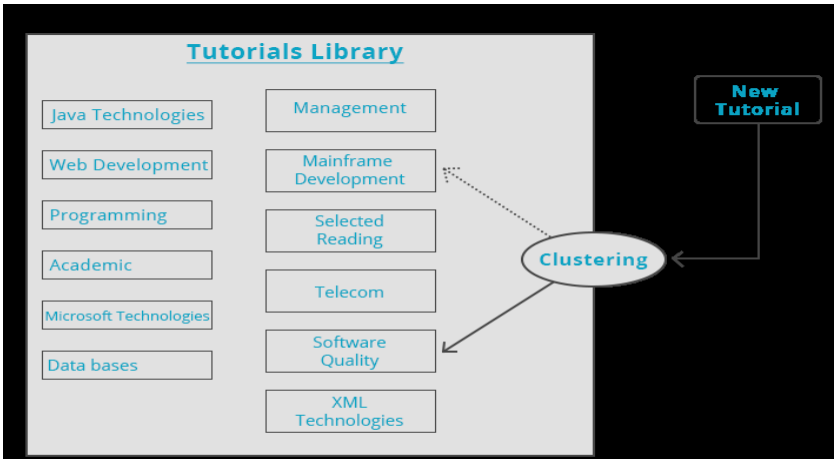
✍ خوشه‌بندی یا Clustering

از تکنیک خوشه‌بندی برای مرتب کردن گروه‌ها یا خوشه‌های داده شبیه به هم براساس خصوصیت‌های مشترک استفاده می‌شود. خوشه‌بندی یکی از تکنیک‌های یادگیری غیر نظارتی یا Unsupervised Learningمی‌باشد.

موتورهای جستجو نظیر گوگل و یاهو از خوشه‌بندی جهت گروه کردن داده‌ها براساس خصوصیات شبیه به هم استفاده می‌کند.

گروه‌های خبری یا Newsgroups از تکنیک خوشه‌بندی جهت گروه کردن مقالات مختلف براساس عناوین به هم مرتبط، استفاده می‌کنند.

موتور خوشه بند داده ورودی را بر اساس خصوصیاتش با دقت بررسی می‌کند و تصمیم می‌گیرد که داده جدید در کدام خوشه قرار بگیرد، به مثال زیر توجه کنید:



کتابخانه آموزش‌های ما شامل موضوعات مختلفی می‌باشد. وقتی ما یک آموزش جدید در tutiran قرار می‌دهیم، توسط یک موتور خوشه بند تصمیم گرفته می‌شود که این آموزش در کدام گروه قرار گرفته شود.

فصل چهارم

محیط ماهوت

در این بخش آموزش نصب ماهوت را فرا می‌گیرید. پیش‌نیازهای ماهوت جاوا و هدوپ می‌باشند. پس ما مرحله‌به‌مرحله سراغ نصب جاوا، هدوپ و ماهوت می‌رویم.

✍ تنظیمات قبل از نصب

قبل از نصب هدوپ روی محیط لینوکسی‌تان، ما نیاز دارید تا از طریق ssh لینوکستان را تنظیم کنید. برای این کار مراحل زیر را به ترتیب طی نمایید.

✍ ساخت یک کاربر

توصیه می‌شود حتماً یک یوزر جداگانه برای هدوپ جهت جدا کردن فایل سیستم هدوپ از فایل سیستم unix ایجاد کنید.

برای ساخت یک کاربر یا یوزر مراحل زیر را طی کنید:

به وسیله دستور su با کاربر root وارد محیط کامند شوید.

با استفاده از دستور زیر یک یوزر با نام دلخواه ایجاد نمایید:

```
useradd username
```

با استفاده از دستور زیر برای کاربر جدید کلمه عبور انتخاب نمایید:

```
passwd username
```

با استفاده از دستور زیر وارد یوزری که ساختید شوید:

```
su username
```

تنظیمات SSH و تولید کلید یا Key Generation

برای انجام عملیات مختلف روی خوشه‌ها نظیر استارت کردن، استپ کردن و توزیع کردن عملیات shell دمون، ما نیاز به تنظیم کردن SSH داریم.
برای احراز هویت کاربران مختلف هدوپ، اجباری است که یک کلید خصوصی یا عمومی برای کاربر هدوپ ایجاد کرده و بین باقی کاربران به اشتراک بگذاریم.
از دستورات زیر جهت تولید زوج کلید و مقدار SSH استفاده می‌کنیم، کلید عمومی را از id_rsa.pub داخل authorized_keys کپی کنید و پس از آن مجوزهای owner و خواندن و نوشتن را به authorized_keys بدهید.

```
$ ssh-keygen -t rsa
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
$ chmod 0600 ~/.ssh/authorized_keys
```

تائید ssh

با استفاده از دستور زیر SSH را verify یا تائید می‌کنیم:

```
ssh localhost
```

نصب جاوا (JAVA)

جاوا اصلی‌ترین پیش‌نیاز هدوپ و HBase محسوب می‌شود. قبل از هر چیز بررسی کنید آیا جاوا در سیستم شما نصب است یا خیر، برای این کار از دستور زی استفاده نمایید:

```
$ java -version
```

خروجی:

```
java version "1.7.0_71"
Java(TM) SE Runtime Environment (build 1.7.0_71-b13)
Java HotSpot(TM) Client VM (build 25.0-b02, mixed mode)
```

خروجی اگر مانند جواب زیر بود یعنی جاوا نصب است در غیر این صورت مراحل زیر را برای نصب جاوا طی کنید.

مرحله ۱:

جاوا (JDK <latest version> - X64.tar.gz) را از لینک زیر دانلود کنید:

http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html

برای مثال jdk-7u71-linux-x64.tar.gz برای شما دانلود می‌شود.

مرحله ۲:

عموماً، شما فایل دانلود شده را می‌توانید در پوشه Downloads پیدا نمایید. آن را پیدا کرده و با استفاده از دستور زیر extract نمایید.

```
$ cd Downloads/
$ ls
jdk-7u71-linux-x64.gz

$ tar zxf jdk-7u71-linux-x64.gz
$ ls
jdk1.7.0_71 jdk-7u71-linux-x64.gz
```

مرحله ۳:

برای این‌که جاوا برای تمامی کاربران در دسترس باشد، شما باید آن را به پوشه /usr/local انتقال دهید. برای این کار از دستورات زیر استفاده نمایید:

```
$ su
password:
# mv jdk1.7.0_71 /usr/local/
# exit
```

مرحله ۴:

برای تنظیم متغیرهای محیطی PATH و JAVA_HOME، خط‌های زیر را در فایل ~/.bashrc اضافه نمایید.

```
export JAVA_HOME=/usr/local/jdk1.7.0_71
export PATH= $PATH:$JAVA_HOME/bin
```

حالا، دوباره java -version را در محیط کامند امتحان کنید و ببینید جاوا به درستی نصب شده یا خیر

✍ دانلود هدوپ (Hadoop)

بعد از نصب جاوا، نوبت به نصب هدوپ می‌رسد. با استفاده از دستور "Hadoop version" نصب یا غیر نصب بودن هدوپ را بررسی می‌کنیم:

```
hadoop version
```

اگر هدوپ نصب باشد خروجی مانند زیر است:

```
Hadoop 2.6.0
Compiled by jenkins on 2014-11-13T21:10Z
Compiled with protoc 2.5.0
From source with checksum 18e43357c8f927c0695f1e9522859d6a
This command was run using /home/hadoop/hadoop/share/hadoop/common/hadoop-common-2.6.0.jar
```

اگر سیستم شما نتوانست هدوپ را پیدا کند، باید هدوپ را دانلود کرده و نصب کنید.
با استفاده از دستورات زیر هدوپ را از بنیاد نرم‌افزارهای آزاد آپاچی دانلود می‌کنیم:

```
$ su
password:
# cd /usr/local
# wget http://mirrors.advancedhosters.com/apache/hadoop/common/hadoop-2.6.0/hadoop-2.6.0-src.tar.gz
# tar xzf hadoop-2.6.0-src.tar.gz
# mv hadoop-2.6.0/* hadoop/
# exit
```

✍ نصب هدوپ (Hadoop)

هدوپ را می‌بایست در هر حالت یا mode ای که می‌خواهید نصب کنید. در اینجا، ما کارکردهای HBase را در حالت pseudo-distributed به شما نشان می‌دهیم؛ بنابراین هدوپ را در حالت یا مد psedu-distributed نصب کنید.
برای نصب Hadoop 2.4.1 روی سیستم مراحل زیر را طی نمایید.

مرحله ۱: تنظیمات هدوپ

شما می‌توانید متغیرهای محیطی هدوپ را با اضافه کردن خطوط زیر به فایل ~/.bashrc تنظیم کنید.

```
export HADOOP_HOME=/usr/local/hadoop export
HADOOP_MAPRED_HOME=$HADOOP_HOME export
HADOOP_COMMON_HOME=$HADOOP_HOME export
HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME export
HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native export
PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
export HADOOP_INSTALL=$HADOOP_HOME
```

حالا باید تغییرات را به وسیله دستور زیر اعمال کنید:

```
$ source ~/.bashrc
```

مرحله ۲: پیکربندی هدوپ

شما می‌توانید تمام فایل‌های تنظیمات هدوپ را در آدرس "$HADOOP_HOME/etc/hadoop" پیدا کنید. شما باید با توجه به شالوده هدوپ در این فایل‌ها تغییرات ایجاد کنید.

$ cd $HADOOP_HOME/etc/hadoop

به منظور گسترش برنامه‌های هدوپ در جاوا، شما باید متغیرهای محیطی جاوا را در hadoop-env.sh ریست کرده و JAVA_HOME درست را در آنجا قرار دهید.

export JAVA_HOME=/usr/local/jdk1.7.0_71

در زیر لیستی از تمام فایل‌هایی که شما برای تنظیم هدوپ باید تغییر دهید آمده است.

core-site.xml

فایل core-site.xml شامل اطلاعاتی از قبیل شماره پورت مورد استفاده برای یک نمونه از هدوپ، میزان حافظه اختصاص یافته برای فایل سیستم، محدودیت حافظه برای ذخیره داده و سایز بافر خواندن/نوشتن.

فایل core-site.xml را باز کنید و کدهای زیر را بین تگ‌های <configuration> و </configuration> قرار دهید:

```
<configuration>
<property> <name>fs.default.name</name> <value>hdfs://localhost:9000</value>
</property>
</configuration>
```

hdfs-site.xml

فایل hdfs-site.xml شامل اطلاعاتی از قبیل مقدار تکثیر داده‌ها، مسیر namenode و مسیرهای datanode فایل سیستم لوکال شما می‌باشد. به این معنی که این فایل جایی است که شما می‌خواهید شالوده یا Infrastructure هدوپ خود را ذخیره نمایید.

فرض کنیم داده‌های زیر در آن موجود می‌باشند:

```
dfs.replication (data replication value) = 1
 (In the below given path /hadoop/ is the user name.
hadoopinfra/hdfs/namenode is the directory created by hdfs file system.)
namenode path = //home/hadoop/hadoopinfra/hdfs/namenode
(hadoopinfra/hdfs/datanode is the directory created by hdfs file system.)
datanode path = //home/hadoop/hadoopinfra/hdfs/datanode
```

این فایل را باز کرده و مشخصات زیر را بین تگ‌های <configuration> و </configuration> قرار دهید:

```
<configuration>
<property>
<name>dfs.replication</name>
<value>1</value>
</property>
<property>
```

<name>dfs.name.dir</name>
<value>file:///home/hadoop/hadoopinfra/hdfs/namenode</value>
</property>
<property>
<name>dfs.data.dir</name>
<value>file:///home/hadoop/hadoopinfra/hdfs/datanode</value>
</property>
</configuration>

نکته: در فایل بالا، تمامی مقادیر مشخصه‌ها توسط کاربر تعریف می‌شوند. شما می‌توانید با توجه به ساختار هدوپ خود این مقادیر را تغییر دهید.

yarn-site.xml

از این فایل برای تنظیمات yarn توی هدوپ استفاده می‌شود. فایل yarn-site.xml را باز کرده و خصوصیات زیر را بین تگ‌های <configuration> و </configuration> قرار دهید:

<configuration>
<property>
<name>yarn.nodemanager.aux-services</name> <value>mapreduce_shuffle</value>
</property>
</configuration>

mapred-site.xml

از این فایل برای مشخص کردن این‌که از کدام چهارچوب یا فریم ورک mapReduce استفاده می‌کنیم، استفاده می‌شود. به صورت پیش‌فرض، هدوپ شامل قالب yarn-site.xml می‌باشد. اول از همه، باید محتویات فایل‌های mapred-site و xml.template را با استفاده از دستور زیر داخل mapred-site.xml بریزیم:

```
$ cp mapred-site.xml.template mapred-site.xml
```

سپس فایل mapred-site.xml را باز کرده و خصوصیات زیر را بین تگ‌های <configuration> و </configuration> قرار دهید:

<configuration>
<property>
<name>mapreduce.framework.name</name>
<value>yarn</value>
</property>
</configuration>

✍ بازرسی یا verfiy نصب هدوپ (Hadoop)

برای اطمینان از صحت نصب هدوپ مراحل زیر را به ترتیب طی نمایید:

مرحله ۱: تنظیم نام گره (name node)

namenode را با استفاده از دستور زیر تنظیم نمایید:

```
$ cd ~
$ hdfs namenode -format 14
```

جواب قابل انتظار برای ما باید مشابه خروجی زیر باشد:

```
10/24/14 21:30:55 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG: host = localhost/192.168.1.11
STARTUP_MSG: args = [-format]
STARTUP_MSG: version = 2.4.1
...
...
10/24/14 21:30:56 INFO common.Storage: Storage directory
/home/hadoop/hadoopinfra/hdfs/namenode has been successfully formatted. 10/24/14
21:30:56 INFO namenode.NNStorageRetentionManager: Going to retain 1 images
with txid >= 0
10/24/14 21:30:56 INFO util.ExitUtil: Exiting with status 0
10/24/14 21:30:56 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at localhost/192.168.1.11
************************************************************/
```

مرحله ۲: بررسی dfs هدوپ

از دستور زیر برای استارت کردن dfs استفاده می‌کنیم. این دستور فایل سیستم هدوپ را استارت می‌کند.

```
$ start-dfs.sh
```

جواب قابل انتظار برای ما باید مشابه خروجی زیر باشد:

```
10/24/14 21:37:56
Starting namenodes on [localhost]
localhost: starting namenode, logging to /home/hadoop/hadoop-2.4.1/logs/hadoop-
hadoop-namenode-localhost.out
localhost: starting datanode, logging to /home/hadoop/hadoop-2.4.1/logs/hadoop-
hadoop-datanode-localhost.out
Starting secondary namenodes [0.0.0.0]
```

مرحله ۳: بررسی اسکریپت Yarn

از دستور زیر برای استارت کردن اسکریپت Yarn استفاده می‌شود. اجرای این دستور باعث استارت شدن دمون Yarn می‌شود.

```
$ start-yarn.sh
```

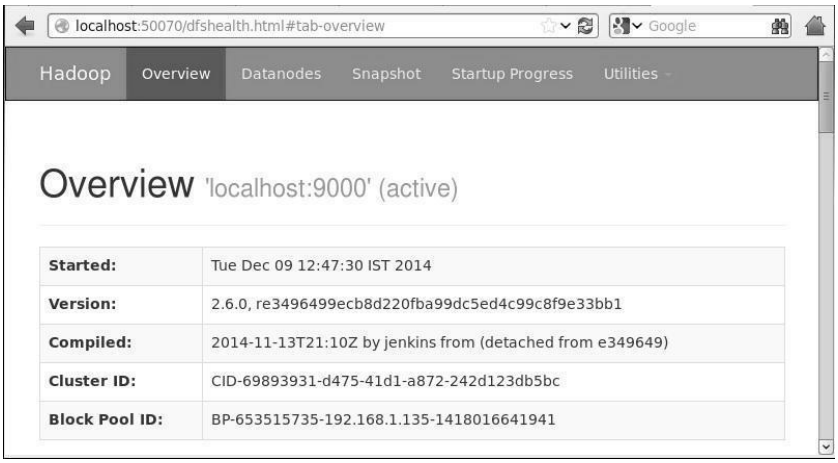
جواب قابل انتظار برای ما باید مشابه خروجی زیر باشد:

```
starting yarn daemons
starting resource manager, logging to /home/hadoop/hadoop-2.4.1/logs/yarn-hadoop-resourcemanager-localhost.out
localhost: starting node manager, logging to /home/hadoop/hadoop-2.4.1/logs/yarn-hadoop-nodemanager-localhost.out
```

مرحله ۴: دسترسی به هدوپ از طریق مرورگر

شماره پورت پیش‌فرض برای دسترسی به هدوپ ۵۰۰۷۰ می‌باشد. از URL زیر برای دریافت سرویس هدوپ روی مرورگر خود استفاده نمایید.

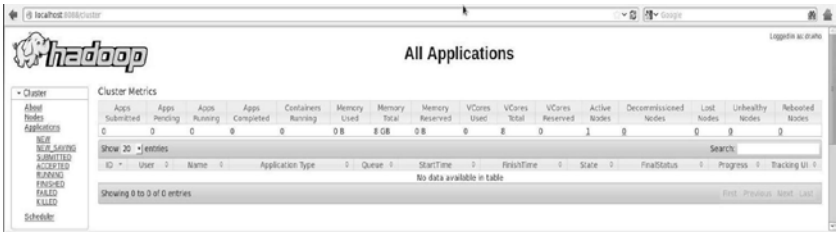
http://localhost:50070/



مرحله ۵: بررسی تمامی برنامه‌ها برای خوشه (Cluster)

شماره پورت پیش‌فرض برای دسترسی به هدوپ ۸۰۸۸ می‌باشد. از URL زیر برای دریافت سرویس هدوپ روی مرورگر خود استفاده نمایید.

http://localhost:8088/



✍ دانلود ماهوت (Mahout)

ماهوت از آدرس http://mahout.apache.org قابل دانلود می‌باشد. عکس زیر صفحه سایت آپاچی مرتبط با ماهوت می‌باشد.

مرحله ۱:

با استفاده از دستور زیر ماهوت را از لینک http://mirror.nexcess.net/apache/mahout دانلود کنید.

```
[Hadoop@localhost ~]$ wget
http://mirror.nexcess.net/apache/mahout/0.9/mahout-distribution-0.9.tar.gz
```

mahout-distribution-0.9.tar.gz در سیستم شما دانلود می‌شود.

مرحله ۲:

فایل دانلود شده‌ی mahout-distribution-0.9.tar.gz را با استفاده از دستور زیر Extract می‌کنیم:

```
[Hadoop@localhost ~]$ tar zxvf mahout-distribution-0.9.tar.gz
```

✍ مخزن مِیوِن (Maven Repository)

در صورتی که می‌خواهید ماهوت را از طریق Maven دریافت کنید درون فایل pom.xml در Eclipse IDE مقادیر زیر را قرار دهید:

```
<dependency>
<groupId>org.apache.mahout</groupId>
<artifactId>mahout-core</artifactId>
<version>0.9</version>
</dependency>
<dependency>
<groupId>org.apache.mahout</groupId>
<artifactId>mahout-math</artifactId>
<version>${mahout.version}</version>
</dependency>
<dependency>
<groupId>org.apache.mahout</groupId>
<artifactId>mahout-integration</artifactId> <version>${mahout.version}</version>
</dependency>
```

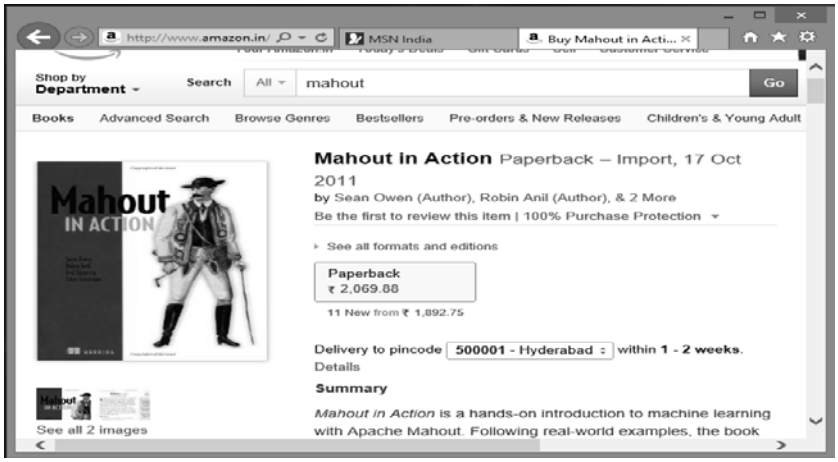
فصل پنجم

سیستم پیشنهاددهنده (Recommendation)

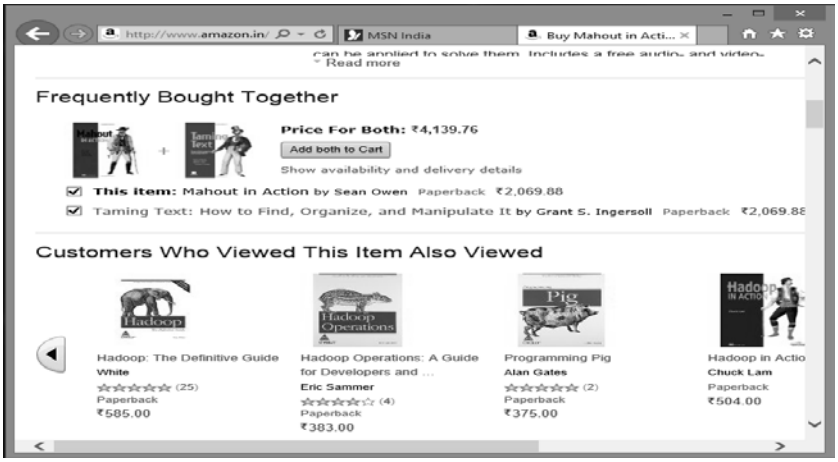
در این بخش می‌خواهیم یکی از بخش‌های خیلی محبوب یادگیری ماشین به نام "پیشنهاد" یا "Recommendation" را پوشش دهیم. همچنین مکانیزم پیشنهاد و این‌که چگونه برنامه‌ای بنویسید که سیستم پیشنهاد ماهوت را پیاده‌سازی کند را فرا خواهید گرفت.

✍ پیشنهاد (Recommendation)

تا به حال به این فکر کرده‌اید که آمازون چگونه لیستی از آیتم‌های پیشنهادی را برحسب آیتمی که انتخاب کرده‌اید به شما نمایش می‌دهد؟
فرض کنید شما می‌خواهید کتاب "Mahout in Action" را از آمازون خریداری کنید:



مانند تصویر زیر هنگامی که شما محصولی را در آمازون انتخاب می‌کنید، آمازون لیستی از آیتم‌های مرتبط انتخابی هم به شما نشان می‌دهد:



این مدل لیست‌های پیشنهادی با کمک موتورهای پیشنهاددهنده تولید می‌شوند. ماهوت موتورهای پیشنهاددهنده مختلفی برای ما فراهم می‌کند، از قبیل:

پیشنهاددهنده‌هایی براساس کاربر (user-based)

پیشنهاددهنده‌هایی براساس محصول (item-based)

و چندین الگوریتم دیگر...

موتور پیشنهاددهنده ماهوت (Mahout)

ماهوت یک موتور پیشنهاددهنده غیر وابسته به هدوپ و غیر توزیع شده دارد. شما می‌توانید یک فایل text حاوی اولویت‌های کاربر برای آیتم‌ها به سیستم بدهید و خروجی‌این موتور اولویت‌های تخمینی آن کاربر برای دیگر محصولات می‌باشد.

مثال

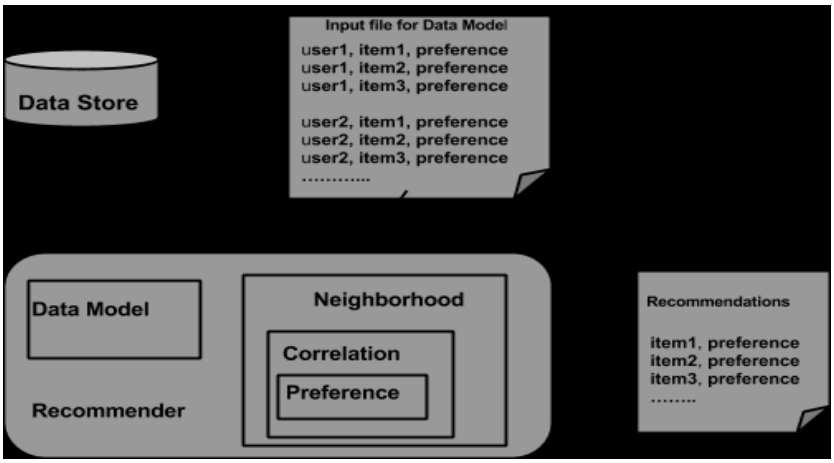
وب‌سایتی را در نظر بگیرید که کالاهایی نظیر موبایل، گجت و لوازم جانبی می‌فروشد. اگر ما بتوانیم ویژگی‌های ماهوت را در این سایت پیاده کنیم، آنگاه می‌توانیم یک موتور پیشنهاددهنده بسازیم. این موتور اطلاعات خریدهای قبلی کاربران را آنالیز کرده و براساس آن محصولات جدید را پیشنهاد می‌دهد.

کامپوننت هایی که ماهوت برای ساخت یک موتور پیشنهاددهنده ارائه می‌کند عبارت‌اند از:

- DataModel
- UserSimilarity
- ItemSimilarity
- UserNeighborhood
- Recommender

از داده ذخیره شده، مدل داده‌یا data model ساخته شده و به عنوان ورودی به موتور پیشنهاددهنده وارد می‌شود. موتور پیشنهاددهنده پیشنهادهایی را برای کاربر تولید می‌کند. در ادامه معماری موتور پیشنهاددهنده را می‌بینید.

معماری موتور پیشنهاددهنده



ساخت یک پیشنهاددهنده به استفاده از ماهوت

مراحل ساخت یک سیستم پیشنهاددهنده ساده در زیر آمده است:

مرحله ۱: ساخت شی مدل داده (Data Model Object)

تابع سازنده کلاس PearsonCorrelationSimilarity به صورت اجباری یک شی از مدل داده را می‌خواهد که در اینجا شامل فایلی می‌شود که درون آن جزییات کاربران، محصولات و اولویت‌های محصولات می‌شود.

یک نمونه فایل مدل داده شبیه زیر می‌باشد:

1,00,1.0

```
1,01,2.0
1,02,5.0
1,03,5.0
1,04,5.0
2,00,1.0
2,01,2.0
2,05,5.0
2,06,4.5
2,02,5.0
3,01,2.5
3,02,5.0
3,03,4.0
3,04,3.0
4,00,5.0
4,01,5.0
4,02,5.0
4,03,0.0
```

شی DataModel به شی file نیاز دارد که آدرس فایل را به آن می‌دهیم. آبجکت DataModel را به صورت زیر می‌سازیم:

```
DataModel datamodel = new FileDataModel(new File("input file"));
```

مرحله ۲: ساخت شی UserSimilarity

شی UserSimilarity را با استفاده از کلاس PearsonCorrelationSimilarity مانند خط زیر بسازید:

```
UserSimilarity similarity = new PearsonCorrelationSimilarity(datamodel);
```

مرحله ۳: ساخت شی UserNeighborhood

این شی یک "neighborhood" یا "همسایگی" کاربرانی شبیه کاربر گرفته شده را محاسبه می‌کند. دو نوع همسایگی وجود دارد:

NearestNUserNeighborhood: این کلاس همسایگی را براساس n کاربری که به کاربر داده شده نزدیک‌تر می‌باشند را محاسبه می‌کند."Nearest" توسط UserSimilarity تعریف شده است.

ThresholdUserNeighborhood: این کلاس همسایگی را براساس تمام کاربرانی که نسبت به کاربر داده شده شبیه‌تر می‌باشند یا به یک حد آستانه مشخصی رسیده‌اند را محاسبه می‌کند."Similarity" توسط UserSimilarity تعریف شده است.

در اینجا ما از ThresholdUserNeighborhood استفاده می‌کنیم و حد اولویت را روی 3.0 تنظیم می‌کنیم.

```
UserNeighborhood neighborhood
= new ThresholdUserNeighborhood(3.0, similarity, model);
```

مرحله ۴: ساخت شی Recommender

شی UserbasedRecomender را بسازید و تمام شی هایی که تا الان ساختیم را مانند نمونه زیر به تابع سازنده این کلاس بدهید.

```
UserBasedRecommender recommender
= new GenericUserBasedRecommender(model, neighborhood, similarity);
```

مرحله ۵: پیشنهاد آیتم‌ها به کاربر

با استفاده از متد ()recommend از اینترفیس Recommender می‌خواهیم یک سری محصول را به کاربر پیشنهاد کنیم. این متد دو پارامتر می‌گیرد. اول شناسه یکتا کاربری (id) که می‌خواهیم پیشنهادات برایش ارسال شود و دومی تعداد پیشنهاداتی که می‌خواهیم برای آن کاربر بفرستیم.

در زیر نمونه استفاده از متد ()Recommender را می‌بینید:

```
List<RecommendedItem> recommendations = recommender.recommend(2, 3);
for (RecommendedItem recommendation: recommendations) {
System.out.println(recommendation);
}
```

✍ نمونه کامل برنامه

در زیر نمونه کامل یک برنامه پیشنهاددهنده آمده است. در این مثال پیشنهاداتی برای کاربری که شناسه‌اش ۲ می‌باشد فراهم می‌کنیم:

```
import java.io.File;
import java.util.List;
import org.apache.mahout.cf.taste.impl.model.file.FileDataModel;
import
org.apache.mahout.cf.taste.impl.neighborhood.ThresholdUserNeighborhood;
import
org.apache.mahout.cf.taste.impl.recommender.GenericUserBasedRecommender;
import org.apache.mahout.cf.taste.impl.similarity.PearsonCorrelationSimilarity;
import org.apache.mahout.cf.taste.model.DataModel;
import org.apache.mahout.cf.taste.neighborhood.UserNeighborhood;
import org.apache.mahout.cf.taste.recommender.RecommendedItem;
import org.apache.mahout.cf.taste.recommender.UserBasedRecommender;
import org.apache.mahout.cf.taste.similarity.UserSimilarity;
```

```
public class Recommender {
public static void main(String args[]){
try{
//Creating data model
DataModel datamodel = new FileDataModel(new File("data")); //data
 //Creating UserSimilarity object.
UserSimilarity usersimilarity =
new PearsonCorrelationSimilarity(datamodel);
//Creating UserNeighbourHHood object.
UserNeighborhood userneighborhood
= new ThresholdUserNeighborhood(3.0, usersimilarity, datamodel);
//Create UserRecomender
UserBasedRecommender recommender
= new GenericUserBasedRecommender(datamodel, userneighborhood,
usersimilarity);
List<RecommendedItem> recommendations = recommender.recommend(2, 3);
for (RecommendedItem recommendation: recommendations) {
System.out.println(recommendation);
}
}catch(Exception e){ }
}
}
```

با استفاده از دستور زیر برنامه را اجرا می‌کنیم:

```
javac Recommender.java
java Recommender
```

و باید خروجی زیر را مشاهده نماییم:

```
RecommendedItem [item:3, value:4.5]
RecommendedItem [item:4, value:4.0]
```

فصل ششم

خوشه‌بندی (Clustering)

خوشه‌بندی یک پروسه برای مرتب کردن عناصر یا آیتم‌های یک مجموعه داده شده، به گروه‌هایی است که برحسب شباهت بین آیتم‌ها از هم جدا شده‌اند. برای مثال، برنامه‌های مرتبط با انتشار خبرهای آنلاین، مقاله‌های خبری را بر اساس عنوان آن‌ها خوشه‌بندی می‌کند.

✍ برنامه‌های خوشه‌بندی

خوشه‌بندی به طور گسترده در بسیاری از برنامه‌ها نظیر تحقیق بازار، تشخیص الگو، آنالیز داده و پردازش تصویر استفاده می‌شود.

خوشه‌بندی می‌تواند به سهامداران جهت تشخیص گروه‌های یکتا از پایه خریداران کمک کند؛ و آن‌ها می‌توانند گروه‌های خریدار را برحسب الگوهای خریدشان توصیف کنند.

در زمینه بیولوژی یا زیست‌شناسی، از خوشه‌بندی برای مشتق کردن گیاهان و طبقه‌بندی کردن حیوانات، دسته‌بندی کردن ژن‌ها برحسب کارکردشان و کسب بینش داخل ساختارهای اصلی در جوامع بشری.

خوشه‌بندی به شناسایی منطقه‌های مشابه سطح کره زمین از طریق پایگاه داده مشاهده کره زمین کمک می‌کند.

خوشه‌بندی همچنین به طبقه‌بندی کردن فایل‌ها و مستندات وب برای کشف اطلاعات کمک می‌کند.

خوشه‌بندی همچنین در برنامه‌های تشخیص outlier مثل کلاه‌برداری کارت اعتباری استفاده می‌شود.

به عنوان یک کاربرد در داده‌کاوی، آنالیز خوشه‌ای این امکان را به ما می‌دهد که برای مشاهده خصوصیات هر خوشه چیزی مثل ابزار برای دریافت بینش داخل توزیع داده را داشته باشیم.

به وسیله ماهوت، می‌توانیم مجموعه‌ای از داده‌ها را خوشه‌بندی کنیم. مراحل انجام این کار به این ترتیب است:
الگوریتم (Algorithm): شما به انتخاب الگوریتم مناسب خوشه‌بندی برای گروه‌بندی کردن عناصر به عنوان یک خوشه نیاز دارید.
شباهت و عدم شباهت (Similarity and Dissimilarity): شما در جایی که باید شباهت را بین دو عنصر جدید و عناصر موجود در گروه‌ها وارسی کنید، به قانون (rule) نیاز دارید.
شرط توقف (Stopping Condition): برای تعریف جایی که خوشه‌بندی لازم نمی‌باشد استفاده از شرط توقف الزامی می‌باشد.

✍ پروسه‌ی خوشه‌بندی

برای خوشه‌بندی کردن داده‌های دریافتی به موارد زیر نیاز دارید:
سرویس هدوپ را استارت کنید.
دایرکتوری‌های الزامی را برای ذخیره فایل‌ها در فایل سیستم هدوپ ایجاد کنید. (یک دایرکتوری برای فایل ورودی، فایل sequence و خروجی خوشه‌بندی شده در مورد canopy)
کپی کردن فایل ورودی از فایل سیستم Unix به فایل سیستم هدوپ
ایجاد فایل زنجیره یا sequence از فایل ورودی
اجرای یکی از الگوریتم‌های خوشه‌بندی
دریافت داده‌های خوشه‌بندی شده.

استارت هدوپ

ماهوت با هدوپ کار می‌کند، بنابراین مطمئن شوید که سرور هدوپ بالا است و کار می‌کند.

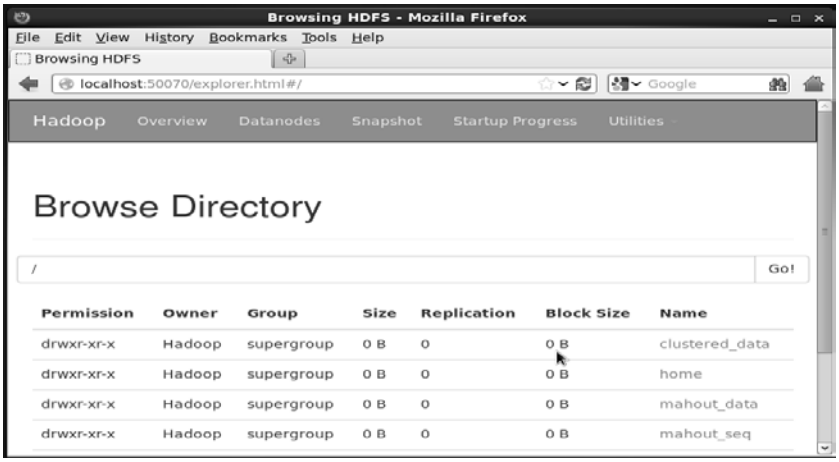
```
$ cd HADOOP_HOME/bin
$ start-all.sh
```

ایجاد دایرکتوری فایل ورودی

برای ذخیره فایل ورودی، فایل زنجیره و فایل خروجی دایرکتوری‌هایی را در فایل سیستم هدوپ با دستورات زیر بسازید:

```
$ hadoop fs -p mkdir /mahout_data
$ hadoop fs -p mkdir /clustered_data
$ hadoop fs -p mkdir /mahout_seq
```

شما همچنین می‌توانید از طریق واسط وب هدوپ به آدرس http://localhost:50070 دایرکتوری‌ها و پوشه‌ها را بررسی و چک کنید.
خروجی مانند تصویر زیر می‌باشد.



کپی کردن فایل‌های ورودی روی HDFS

حالا، فایل داده ورودی را از فایل سیستم لینوکسی خودروی دایرکتوری mahout_data در فایل سیستم هدوپ کپی کنید.
فرض می‌کنیم نام فایل ورودی mydata.txt و نام دایرکتوری /home/Hadoop/data/ می‌باشد.

```
$ hadoop fs -put /home/Hadoop/data/mydata.txt /mahout_data/
```

ایجاد فایل زنجیره یا sequence

ماهوت برای تبدیل فایل ورودی به فرمت فایلی که کار روی آن انجام می‌شود (Sequence File)، ابزاری را آماده کرده است. این ابزار دو پارامتر می‌خواهد.
دایرکتوری فایل ورودی که فایل اصلی در آن قرار دارد.
دایرکتوری فایل خروجی که داده خوشه‌بندی شده در آن ذخیره می‌گردد.
قسمت کمکی دستور ابزار seqdirectory خروجی‌ای مانند زیر دارد:

```
[Hadoop@localhost bin]$ ./mahout seqdirectory --help
Job-Specific Options:
--input (-i) input Path to job input directory.
```

--output (-o) output The directory pathname for output.
--overwrite (-ow) If present, overwrite the output directory

برای ایجاد فایل sequence با استفاده از این ابزار باید سینتکس زیر را رعایت کنید:

mahout seqdirectory -i <input file path> -o <output directory>

مثال

mahout seqdirectory
-i hdfs://localhost:9000/mahout_seq/
-o hdfs://localhost:9000/clustered_data/

الگوریتم‌های خوشه‌بندی

ماهوت برای خوشه‌بندی از دو الگوریتم اصلی استفاده می‌کند:

خوشه‌بندی canopy

خوشه‌بندی k-means

خوشه‌بندی Canopy

خوشه‌بندی canopy یا خوشه‌بندی سایبانی یک تکنیک ساده و سریع برای خوشه‌بندی است که توسط ماهوت استفاده می‌شود. با آبجکت ها در این در این حالت مانند امتیازهایی در یک فضای خام رفتار می‌شود. این تکنیک اغلب به عنوان اولین مرحله در تکنیک‌های دیگر خوشه‌بندی مانند k-means استفاده می‌شود.

شما می‌توانید یک برنامه canopy را با استفاده از دستور زیر اجرا کنید:

mahout canopy -i <input vectors directory>
-o <output directory>
-t1 <threshold value 1>
-t2 <threshold value 2>

برنامه canopy به یک دایرکتوری فایل ورودی یا input با یک فایل sequenceو یک دایرکتوری خروجی که داده‌های خوشه‌بندی شده در آنجا ذخیره شود، نیاز دارد.

مثال

mahout canopy -i hdfs://localhost:9000/mahout_seq/mydata.seq
-o hdfs://localhost:9000/clustered_data
-t1 20
-t2 30

شما می‌توانید داده‌های خوشه‌بندی شده به وجود آمده را در دایرکتوری خروجی مشاهده نمایید.

خوشه‌بندی k-means

خوشه‌بندی k-means یک الگوریتم مهم خوشه‌بندی است.k در الگوریتم k-means به معنای تعداد خوشه‌هایی است که داده‌ها در آن‌ها تقسیم می‌شوند. برای مثال، در این برنامه مقدار k برابر ۳ قرار دارد، پس الگوریتم داده‌ها را به ۳ خوشه تقسیم می‌کند.
هر آبجکت به عنوان یک بردار یا vector در فضا در نظر گرفته می‌شود. در ابتدا مقدار k توسط الگوریتم و به صورت اتفاقی یا random تعیین شده و به عنوان مرکز از آن استفاده می‌شود، هر آبجکتی که به مرکز نزدیک‌تر است خوشه‌بندی می‌شود. الگوریتم‌های زیادی برای اندازه‌گیری فاصله وجود دارند که کاربر می‌تواند یکی از آن‌ها را انتخاب نماید.

ساخت فایل‌های بردار یا vector

برخلاف الگوریتم canopy، الگوریتم k-means از فایل‌های برداری جهت ورودی استفاده می‌کند، بنابراین شما باید فایل‌های برداری بسازید.
برای تولید فایل‌های برداری از فرمت فایل sequence، ماهوت ابزاری به نام seq2parse را ایجاد کرده است.
در زیر چند خصوصیت از ابزار seq2parse را می‌بینید. با استفاده از این خصوصیت‌ها فایل‌های بردار را بسازید.

```
$MAHOUT_HOME/bin/mahout seq2sparse
--analyzerName (-a) analyzerName The class name of the analyzer
--chunkSize (-chunk) chunkSize The chunkSize in MegaBytes.
--output (-o) output The directory pathname for o/p
--input (-i) input Path to job input directory.
```

بعد از ساخت بردارها، با الگوریتم k-means پیش می‌رویم. برای این‌که برنامه k-means را اجرا کنیم از کد زیر استفاده می‌کنیم:

```
mahout kmeans -i <input vectors directory>
-c < input clusters directory >
-o < output working directory >
-dm < Distance Measure technique >
-x < maximum number of iterations >
-k < number of initial clusters >
```

الگوریتم خوشه‌بندی k-means به یک دایرکتوری بردار ورودی، دایرکتوری خوشه‌های خروجی، اندازه فاصله، ماکزیمم عدد تکرار و عدد integer که نشان می‌دهد داده‌ها به چه تعدادی تقسیم می‌شوند، نیاز دارد.

فصل هفتم

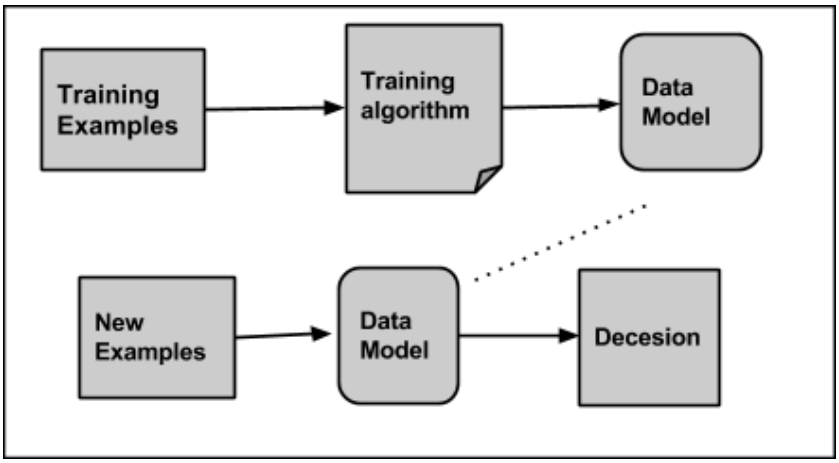
طبقه‌بندی (Classification)

✍ طبقه‌بندی (Classification) چیست؟

طبقه‌بندی یک تکنیک یادگیری ماشین است که از داده‌های شناخته شده برای این‌که بداند داده‌ی جدید در کدام گروه قرار بگیرد، استفاده می‌کند. برای مثال
برنامه iTunes از طبقه‌بندی برای آماده کردن لیست پخش یا playlists استفاده می‌کند.
شرکت‌های ارائه‌دهنده سرویس ایمیل مانند یاهو و گوگل از این تکنیک برای تشخیص این‌که ایمیل جدید باید در گروه اسپم‌ها قرار گیرد یا خیر، استفاده می‌کند. الگوریتم دسته‌بندی خود را با عادت‌های کاربر آموزش می‌دهد به این صورت که می‌بیند کاربر چه ایمیل‌هایی را اسپم در نظر می‌گیرد. بر اساس آن، طبقه‌بندی کننده تصمیم می‌گیرد که ایمیل جدید در inbox قرار گیرد یا در spam.

✍ طبقه‌بندی (Classification) چگونه کار می‌کند؟

در هنگام طبقه‌بندی کردن یک مجموعه داده، سیستم طبقه‌بندی کننده فعالیت‌های زیر را انجام می‌دهد:
در ابتدا مدل داده جدید باید برای استفاده از هرگونه الگوریتم یادگیرنده آماده شود.
سپس باید مدل داده آماده شده تست شود.
بعد از آن، از این مدل داده جهت ارزیابی داده جدید و تصمیم درباره کلاس آن استفاده می‌شود.



✍ برنامه‌های طبقه‌بندی (Classification)

تشخیص کلاه‌برداری کارت‌های اعتباری:
از مکانیزم طبقه‌بندی برای پیش‌بینی کلاه‌برداری‌های کارت‌های اعتباری استفاده می‌شود. با استفاده از اطلاعات کلاه‌برداری‌های تاریخی یا historical که در گذشته رخ داده است، طبقه‌بندی کننده می‌تواند پیش‌بینی کند که تراکنش‌های آینده می‌توانند کلاه‌برداری باشند.
ایمیل‌های اسپم:
با توجه به ویژگی‌های ایمیل‌های اسپم قبلی، طبقه‌بندی کننده تصمیم می‌گیرد که ایمیل جدید را در پوشه اسپم قرار دهد یا خیر.

✍ طبقه‌بندی کننده Naive Bayes

ماهوت از الگوریتم طبقه‌بندی کننده Naive Bayes استفاده می‌کند. دو پیاده‌سازی این الگوریتم:

- طبقه‌بندی Naive Bayes توزیع شده
- طبقه‌بندی Naive Bayes تکمیلی یا متمم (Complementary)

Naive Bayes یک تکنیک ساده برای ساخت طبقه‌بندی کننده است. این یک الگوریتم تکی برای آموزش طبقه بندها نیست، اما از همان خانواده می‌باشد. یک طبقه‌بندی کننده Bayes برای طبقه‌بندی کردن نمونه‌های مسئله، مدل‌هایی را می‌سازد. این طبقه‌بندی کننده‌ها از داده‌های موجود به وجود می‌آیند.

یکی از مزایای Naive Bayes این است که فقط به مقدار کمی از داده آموزشی برای این‌که بتواند پارامترهای لازم برای طبقه‌بندی را تخمین بزند، نیاز دارد.
برای برخی از انواع مدل‌های احتمال، طبقه بند Naive Bayes می‌تواند با استفاده از تنظیمات آموزش نظارتی (supervised) به طرز خیلی مؤثری آموزش ببیند.
با وجود این فرض‌های ساده گیرانه، طبقه بند Naive Bayes می‌تواند بسیاری از مسائل پیچیده دنیای واقعی را حل کند.

✍ پروسه طبقه‌بندی (Classification)

برای پیاده‌سازی طبقه‌بندی مراحل زیر را باید دنبال کنیم:
تولید داده برای مثال (داده تستی)
ساخت فایل‌های sequence از روی این داده‌ها
تبدیل فایل‌های sequence به بردارها
آموزش بردارها
تست بردارها
در ادامه هر مرحله را با دستورات و آپشن های آن توضیح می‌دهیم.

مرحله ۱: تولید داده تستی

داده‌هایی که می‌خواهید طبقه‌بندی کنید تولید و یا دانلود کنید. برای مثال، شما می‌توانید از لینک زیر ۲۰ خبرنامه تستی را دانلود کنید:
http://people.csail.mit.edu/jrennie/20Newsgroups/20news-bydate.tar.gz
برای ذخیره داده‌ها یک دایرکتوری بسازید؛ مانند زیر مثال را دانلود نمایید:

```
$ mkdir classification_example
$ cd classification_example
$tar xzvf 20news-bydate.tar.gz
wget http://people.csail.mit.edu/jrennie/20Newsgroups/20news-bydate.tar.gz
```

مرحله ۲: ساخت فایل‌های sequence

با استفاده از ابزار seqdirectory فایل sequence لازم را درست کنید. دستور این کار به صورت زیر است:

```
mahout seqdirectory -i <input file path> -o <output directory>
```

مرحله ۳: تبدیل فایل‌های sequence به بردارها

با استفاده از ابزار seq2parse فایل‌های sequence را به فایل‌های بردار تبدیل کنید. آپشن های مختلف این دستور در زیر آمده است:

```
$MAHOUT_HOME/bin/mahout seq2sparse
--analyzerName (-a) analyzerName The class name of the analyzer
--chunkSize (-chunk) chunkSize The chunkSize in MegaBytes.
--output (-o) output The directory pathname for o/p
--input (-i) input Path to job input directory.
```

مرحله ۴: آموزش بردارها

بردارهای تولید شده را با استفاده از ابزار trainnb آموزش دهید. آپشن های مختلف این دستور در زیر آمده است:

```
mahout trainnb
-i ${PATH_TO_TFIDF_VECTORS}
-el
-o ${PATH_TO_MODEL}/model
-li ${PATH_TO_MODEL}/labelindex
-ow
-c
```

مرحله ۵: تست بردارها

بردارهای تولید شده را با استفاده از ابزار testnb تست کنید. آپشن های مختلف این دستور در زیر آمده است:

```
mahout testnb
-i ${PATH_TO_TFIDF_TEST_VECTORS}
-m ${PATH_TO_MODEL}/model
-l ${PATH_TO_MODEL}/labelindex
-ow
-o ${PATH_TO_OUTPUT}
-c
-seq
```

فصل هشتم

معرفی اسپارک

اسپارک آپاچی چیست؟

اسپارک محاسباتی خوشه‌ای سبک و سریع برای محاسبات سریع طراحی شده است.
اسپارک در لایه بالایی Hadoop MapReduce می‌باشد و مدل MapReduce را برای مؤثر بودن انواع بیشتری از محاسباتی که شامل کوئری های تعاملی (Interactive Queries) و جریان پردازش (Stream Processing) می‌باشد، گسترش می‌دهد.
این آموزش شامل توضیحات مقدماتی برنامه‌نویسی هسته اسپارک می‌باشد.
مخاطبان
مخاطبان این آموزش کسانی هستند که می‌خواهند اصول اولیه آنالیز اطلاعات بزرگ (بیگ دیتا) با استفاده از فریم ورک اسپارک را یاد گرفته و گسترش‌دهنده اسپارک باشند.
همچنین این آموزش می‌تواند برای آنالیز حرفه‌ای و برنامه‌نویسان ETL نیز مفید باشد.
پیش‌نیاز
قبل از این‌که این آموزش را شروع کنید، ما فرض می‌کنیم شما یک پیش‌زمینه قبلی در زمینه برنامه‌نویسیScala ، اصول پایگاه داده (Database Concepts) و یکی از سیستم‌عامل‌های لینوکس دارید.

تکامل اسپارک

اسپارک یکی از زیر پروژه‌های Hadoop است که در سال ۲۰۰۹ در آزمایشگاه AMPlab برکلی توسط Matei Zaharia ساخته شد.

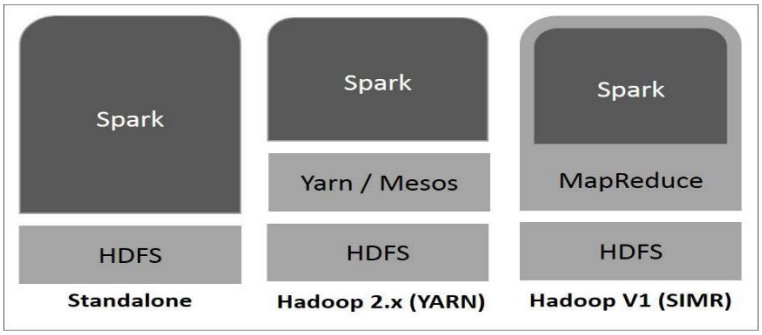
در سال ۲۰۱۰ تحت لیسانس BSD عضو متن باز (Open Source) شد.
در سال ۲۰۱۳ تحت حمایت بنیاد نرم‌افزاری آپاچی قرار گرفت و حالا پروژه اسپارک در فوریه ۲۰۱۴ در بالاترین سطح پروژه‌های آپاچی قرار گرفت.

ویژگی‌های اسپارک

از ویژگی‌های اسپارک می‌توان به نکات زیر اشاره کرد:
سرعت:
اسپارک کمک می‌کند تا برنامه‌ها در خوشه‌ی Hadoop اجرا شوند، این یعنی بیش از ۱۰۰ برابر سرعت اجرا در حافظه و بیش از ۱۰ برابر سرعت اجرا روی disk .
این اتفاق به وسیله کم کردن تعداد عملیات خواندن / نوشتن روی دیسک مسیر می‌شود.
همچنین پردازش داده‌های متوسط (Intermediate) را در حافظه ذخیره می‌کند.
پشتیبانی از چندین زبان:
اسپارک API های از پیش تعیین‌شده‌ای برای java، Scala یا Python دارد.
بنابراین شما می‌توانید به زبان‌های مختلف برنامه‌تان را بنویسید، اسپارک دارای ۸۰ دستور سطح بالا (high-level) برای کوئری های تعاملی می‌باشد.
آنالیزهای حرفه‌ای:
اسپارک نه تنها از "MAP" و "Reduce" پشتیبانی می‌کند بلکه از کوئری های SQL، جریان‌های داده‌ای، ماشین یادگیرنده (Machine learning ML) و الگوریتم‌های گراف هم پشتیبانی می‌کند.

اسپارک های ساخته شده روی hadoop

نمودار زیر سه روشی که اسپارک می‌تواند به قسمت‌های Hadoop وصل شود را نشان می‌دهد.



تفسیر سه روش گسترش اسپارک:

۱- Standalone (به تنهایی):

گسترش تنهای اسپارک به این معنی است کـه اسـپارک جـایی در بـالای HDFS(Hadoop Distribued File System)مستقر می‌شود و به وضوح فضا برای HDFS اختصاص داده شده است.

۲- Hadoop Yarn:

گسترش Hadoop Yarn ساده است، اسپارک بدون هیچ نصب قبلی یا دسترسـی کامـل (Root) روی Yarn اجرا می‌شود.

این حالت این امکان را می‌دهد که بقیه قسمت‌ها بالای پشته (Stack) اجرا شوند.

۳- Spark In MapReduce (SIMR):

اسپارک در MapReduce برای اجرای کارها (Job) استفاده می‌شود.

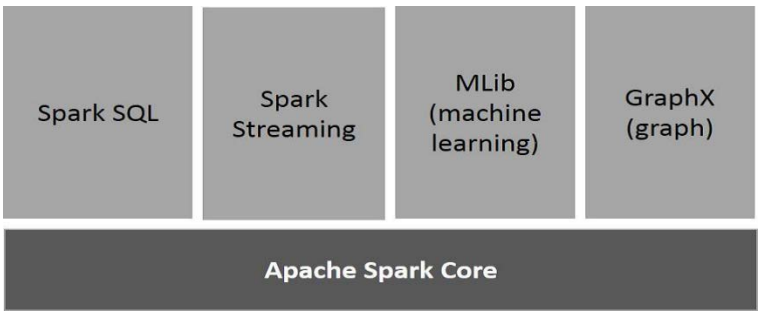
با استفاده ازSIMR ، کاربر می‌تواند اسپارک را شروع (Start) کند و از پوسته (Shell) بدون هیچ دسترسی خاصی (Administrative Access) استفاده کند.

بخش‌های اسپارک

در تصویر زیر بخش‌های مختلف اسپارک را مشاهده می‌کنید.

Apache Spark Core (هسته‌ی اسپارک آپاچی)

هسته اسپارک یک موتور اجرایی عمومی اساسی برای پلتفرم اسـپارک اسـت کـه تمـام امکانـات دیگر روی آن ساخته شده است.



بدین وسیله محاسبات داخل حافظه (In-Memory) فراهم می‌شود و مجموعه داده‌ها به سیستم‌های ذخیره خارجی ارجاع داده می‌شوند.

Spark SQL

Spark SQLیک بخش روی هسته اسپارک است که داده‌های انتزاعی جدید New Data) (Abstractionکه Schema RDD نامیده می‌شوند را معرفی می‌کند که پشتیبانی از داده‌های ساخت یافته (Structured-Data) و شبه ساخت یافته را فراهم می‌کند.

Spark Streaming

Spark Streamingاز قابلیت زمان‌بندی سریع هسته‌های اسپارک برای فراهم کردن آنالیزهای جریانی (Streaming Analytics) استفاده می‌کند.
Ingestها داده رو به گروه‌های کوچک (Mini-Batches) و انتقالات RDD را روی آن گروه‌های کوچک داده فراهم می‌کند.

MLib (Machine Learning Library)

MLibیک فریم ورک توزیع شده ماشین یادگیرنده روی اسپارک است زیرا معماری اسپارک بر پایه حافظه (Memory Base) و توزیع شده است.
MLibدر اسپارک ۹ برابر سریع‌تر از ورژن Disk-Base را Apache Mahout می‌باشد) .قبل از این‌که Mahout اینترفیس اسپارک را بگیرد(.

GraphX

Graphxیک فریم ورک پردازش گراف توزیع شده روی اسپارک است.
Graphxیک API برای محاسبه گراف Expressing فراهم می‌کند که توانایی مدل کردن گرافی که کاربر تعریف می‌کند را با استفاده از Pregel Abstraction API را دارد.
علاوه بر این زمان اجرا را برای سطوح انتزاع بهینه می‌کند.

فصل نهم

دیتاست های توزیع شده ارتجاعی

✍ دیتاست های توزیع شده ارتجاعی

مجموعه داده‌های توزیع شده ارتجاعی (RDD) از پایه‌ای‌ترین ساختار داده‌های اسپارک هستند.
RDD ها کالکشن توزیع شده و غیرقابل تغییر (Immutable) اشیاء هستند.
هر دیتاست (DataSet) در RDD به پارتیشن‌های منطقی تقسیم می‌شوند که می‌توانند در گره‌های مختلف خوشه محاسبه شوند.
RDD ها شامل هر شی از جاوا، پایتون، اسکالا و کلاس‌هایی که کاربر تعریف می‌کند، می‌شوند.
معمولاً RDD، Read-Only (فقط خواندنی) بوده و مجموعه‌ای پارتیشن‌بندی شده از رکوردهاست.
RDD ها می‌توانند از طریق عملیات قطعی بر داده‌ها روی حافظه ثابت یا دیگر RDD ها ساخته شوند.
RDD ها کالکشن هایی از المان‌هایی هستند که Fault-Tolerant (توانایی تحمل خطا) داشته که می‌توانند به صورت موازی عمل کنند.
دو راه برای ساخت RDD ها وجود دارد:
۱- موازی‌سازی کالکشن موجود در درایو برنامه شما
۲- ارجاع دادن یک دیتاست به سیستم حافظه خارجی مثل سیستم اشتراک فایل، HDFS، HBase یا هر منبع داده‌ای با فرمت ورودی HADoop .

اسپارک عملیات MapReduce را با استفاده از الگو RDD (Concept) ها سریع‌تر و مؤثرتر می‌سازد.
بگذارید اول بحث کنیم که MapReduce ها چطور کار می‌کنند و چرا خیلی مؤثر و مفید نیستند.

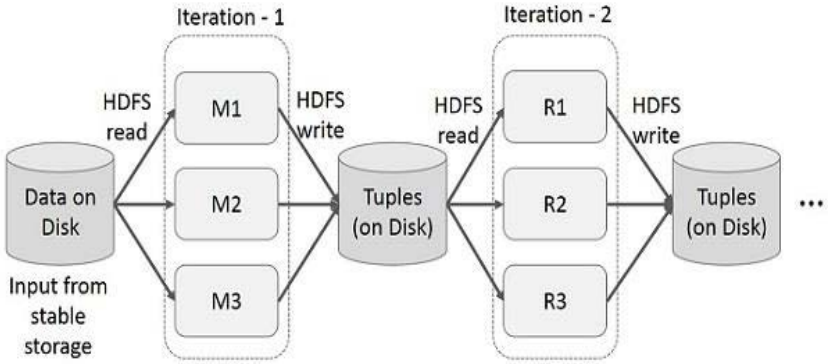
سرعت کم اشتراک داده درMapReduce

MapReduce به صورت گسترده برای پردازش و تولید دیتاست های بزرگ با موازی‌سازی و الگوریتم‌های توزیع شده روی یک خوشه استفاده می‌شود.
این امکان به کاربران اجازه می‌داد با استفاده از مجموعه سطح بالای عملیات محاسبات موازی بنویسند بدون داشتن نگرانی درباره توزیع‌پذیری کار و تحمل‌پذیری خطا.
متأسفانه، در بیشتر فریم ورک های موجود، تنها راه برای استفاده مجدد از داده بین محاسبه کننده‌ها (مثال: بین دو MapReduce Jobs) نوشتن آن داده روی یک سیستم حافظه خارجی ثابت بود (مثل: HDFS) با اینکه این فریم ورک تعداد زیادی انتزاع (Abstractions) برای دسترسی به منابع خوشه‌های محاسبه کننده فراهم کرده است، کاربران هنوز چیزهای بیشتری می‌خواهند.
هم برنامه‌های تکراری و هم برنامه‌های تعاملی نیاز دارند تا اشتراک داده بین کارها موازی را سریع‌تر انجام دهند.
اشتراک داده‌ها در MapReduce در موارد تکثیر (Replication)، موازی‌سازی (Serialization) و ورودی و خروجی دیسک (Disk IO) کند می‌باشد.
درباره سیستم حافظه، اغلب برنامه‌های Hadoop، بالای ٪۹۰ زمان را صرف عملیات خواندن / نوشتن HDFS می‌کنند.

عملیات تکراری روی MapReduce

استفاده مجدد از نتایج میانی بین چند محاسبه کننده در برنامه‌های چند طبقه (Multi-Stage) را گویند.
تصویر زیر نشان می‌دهد که فریم ورک فعلی چگونه کار می‌کند وقتی از عملیات تکراری روی MapReduceاستفاده می‌کند.

این خسارت قابل توجه هزینه‌هایی دارد که در حین تکثیر، ورودی و خروجی دیسک و موازی‌سازی سیستم را کند می‌کند.

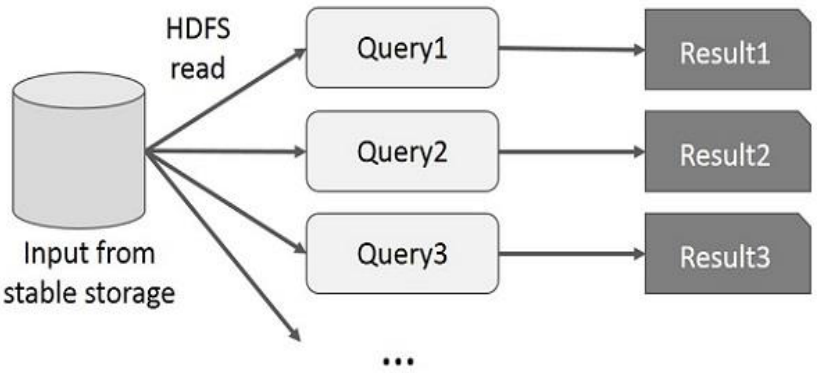


عملیات تعاملی روی MapReduce

کاربران کوئری های موردی) اختصاصی، (Ad-Hocرا روی زیر مجموعه‌های مساوی از داده می‌زنند

هر کوئری باید عملیات ورودی و خروجی روی دیسک را روی حافظه ثابت ثبت کند که همین می‌تواند زمان اجرای برنامه را افزایش دهد.

تصویر زیر نحوه کار کوئری های تعاملی در فریم ورک فعلی را نشان می‌دهد.



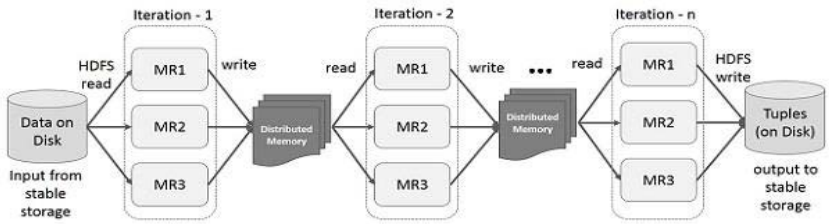
اشتراک داده با اسپارکRDD

اشتراک داده هنگام تکثیر، موازی‌سازی و ورودی و خروجی دیسک در MapReduce کند می‌باشد.

اغلب برنامه‌های Hadoop بالای ٪۹۰ زمان خود را صرف عملیات خواندن / نوشتن HDFS می‌کنند.
با شناسایی این مشکل، محققان یک فریم ورک ویژه به نام "اسپارک آپاچی" را گسترش دادند.
ایده کلیدی اسپارک دیتاست های توزیع شده ارتجاعی (RDD) می‌باشد که محاسبات پردازشی داخل حافظه را فراهم می‌کند.
این به این معنی است که موقعیت حافظه را به عنوان یک شی بین کارها (Jobs) ذخیره می‌کند و این شی بین آن کارها به اشتراک گذاشته می‌شود.
اشتراک داده داخل حافظه بین ۱۰ تا ۱۰۰ برابر سریع‌تر از اشتراک داده در شبکه و دیسک می‌باشد.
در قسمت بعد متوجه می‌شویم که عملیات تکراری و تعاملی چطور در اسپارک RDD جا می‌گیرند.

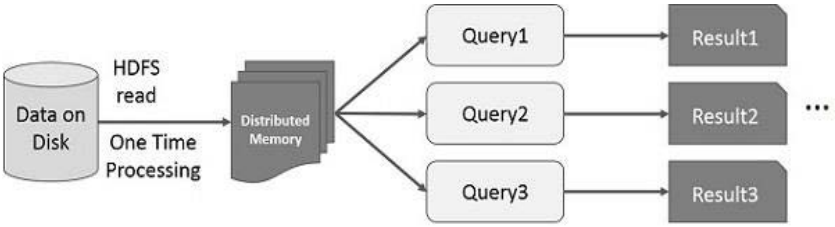
✍ عملیات تکراری روی اسپارکRDD

شکل زیر نحوه اجرا شدن عملیات تکراری روی اسپارک RDD را نشان می‌دهد
در این حالت نتایج میانی به جای حافظه ثابت (دیسک)، روی حافظه‌های توزیع شده Memory) (ذخیره می‌شوند که این اتفاق سیستم را سریع‌تر می‌کند.
نکته: اگر حافظه توزیع شده (RAM) برای ذخیره نتایج میانی (حالت Job ها) کافی باشد، اون موقع نتایج رو روی دیسک ذخیره می‌کنند.



✍ عملیات تعاملی روی اسپارکRDD

شکل زیر نحوه اجرا شدن عملیات تعاملی روی اسپارک RDD را نشان می‌دهد.
اگر کوئری های مختلف به صورت پشت سرهم روی یک مجموعه از داده اجرا شوند، این مجموعه داده برای بهتر شدن زمان اجرا می‌تواند در حافظه نگهداری شود.



به صورت پیش‌فرض، هر بار که اتفاقی روی هر RDD تغییر کرد و رخ دهد، RDDمی‌تواند دوباره محاسبه کند. به‌هرحال شاید شما RDD ثابت (Persist) در حافظه بخواهید، در این مورد اسپارک می‌تواند برای دسترسی سریع‌تر المان‌ها را اطراف خوشه‌ای نگه داره که شما دفعه بعدی روی آن کوئری می‌زنید.

همچنین این حالت، RDDهای ثابت روی دیسک یا تکثیر شدن بین چندین گره (Node) را پشتیبانی می‌کند.

فصل دهم

نصب اسپارک

✍ چک کردن نصب جاوا

اسپارک یکی از زیر پروژه‌های Hadoop می‌باشد، بنابراین بهتر است اسپارک روی یک سیستم با پایه‌ی لینوکسی نصب گردد. در ادامه مراحل نصب اسپارک آپاچی را مرور می‌کنیم.
برای نصب اسپارک اولین قدم ضروری نصب جاوا است. برای چک کردن نصب جاوا دستور زیر را اجرا کنید:

```
$ java-version
```

خروجی

```
java version "1.7.0_71"
Java(TM) SE Runtime Environment (build 1.7.0_71-b13)
Java HotSpot(TM) Client VM (build 25.0-b02, mixed mode)
```

اگر پیغام بالا را مشاهده کردید یعنی جاوا نصب شده است و می‌توانید به مرحله بعدی بروید، در غیر این صورت قبل از رفتن به مرحله بعد جاوا را نصب کنید.

✍ چک کردن نصب اسکالا

شما برای پیاده‌سازی اسپارک به زبان اسکالا (Scala) نیاز دارید.
برای این‌که چک کنید که آیا Scala نصب هست یا خیر دستور زیر را وارد نمایید:

```
$ Scala –version
```

اگر اسکالا روی سیستم شما نصب باشد پیامی مانند پیام زیر را می‌بینید:

خروجی

Scala code runner version 2.11.6 -- Copyright 2002-2013, LAMP/EPFL

اگر شما اسکالا را نصب نکرده‌اید مراحل بعد را دنبال کنید.

دانلود اسکالا

آخرین ورژن اسکالا را از اینجا دانلود کنید.

http://www.scala-lang.org/download

برای این آموزش ما از اسکالا ورژن Scala-2.11.6 استفاده کرده‌ایم.

بعد از دانلود فایل tar را از فولدر دانلود خود برداشته و اسکالا را طبق مرحله بعدی نصب کنید.

نصب اسکالا

برای نصب اسکالا مراحل زیر را دنبال نمایید:

-فایل tar اسکالا را از حالت فشرده خارج کنید.

برای این کار می‌توانید از دستور زیر استفاده کنید:

```
-$tar xvf Scala-2.11.6.tgz
```

-تغییر محل فایل‌های نرم‌افزار:Scala

از دستور زیر برای منتقل کردن فایل‌های اسکالا به (/usr/local/scala) استفاده نمایید:

```
$ su –
Password:
# cd /home/Hadoop/Downloads/
# mv scala-2.11.6 /usr/local/scala
# exit
```

-تنظیم PATH برای اسکالا:

با استفاده از دستور زیر متغیر فراگیر PATH را برای اسکالا تنظیم نمایید:

```
$ export PATH = $ PATH:/usr/local/scala/bin
```

-چک کردن نصب اسکالا:

پس از اتمام مراحل فوق جهت چک کردن نصب اسکالا از دستور زیر استفاده نمایید:

```
$ Scala-version
```

اگر تمام مراحل را به درستی طی کرده باشید و اسکالا درست نصب و تنظیم شده باشد، پیامی مانند پیام زیر مشاهده می‌کنید:

خروجی

Scala code runner version 2.11.6 -- Copyright 2002-2013, LAMP/EPFL

دانلود اسپارک

آخرین ورژن اسپارک را از اینجا دانلود کنید.
https://spark.apache.org/downloads.html
در این آموزش ما از ورژن Spark-1.3.1-bin-hadoop2.6 استفاده می‌کنیم.
بعد از دانلود فایل tar را از فولدر دانلود خود برداشته و اسپارک را طبق مرحله بعدی نصب کنید.

نصب اسپارک

برای نصب اسپارک مراحل زیر را دنبال نمایید:
-فایل tar اسپارک را از حالت فشرده خارج کنید.
برای این کار می‌توانید از دستور زیر استفاده کنید:

```
$tar xvf Spark-1.3.1-bin-hadoop2.6.tgz
```

-تغییر محل فایل‌های نرم‌افزار:Spark
از دستور زیر برای منتقل کردن فایل‌های اسپارک به (/usr/local/ Spark) استفاده نمایید:

```
$ su –
Password:
# cd /home/Hadoop/Downloads/
# mv spark-1.3.1-bin-hadoop2.6 /usr/local/spark
# exit
```

-تنظیم متغیر فراگیر (Environment) برای اسپارک:
خط زیر را به فایل ~/.bashrc اضافه کنید. به این معنی که اضافه کردن محلی که فایل نـرم‌افـزار اسپارک متغیر PATH را پیدا می‌کند.

```
export PATH = $ PATH:/usr/local/ Spark /bin
```

از دستور زیر برای سورس کردن فایل ~/.bashrc استفاده کنید:

```
$ Source ~/.bashrc
```

چک کردن نصب اسپارک

برای باز کردن پوسته (Shell) اسپارک از دستور زیر استفاده نمایید:

```
$ spark-shell
```

اگر اسپارک به درستی نصب شده باشد پیامی مانند پیام زیر را نشان داده می‌شود:
خروجی

```
Spark assembly has been built with Hive, including Datanucleus jars on classpath
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
```

15/06/04 15:25:22 INFO SecurityManager: Changing view acls to: hadoop
15/06/04 15:25:22 INFO SecurityManager: Changing modify acls to: hadoop
15/06/04 15:25:22 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(hadoop); users with modify permissions: Set(hadoop)
15/06/04 15:25:22 INFO HttpServer: Starting HTTP Server
15/06/04 15:25:23 INFO Utils: Successfully started service 'HTTP class server' on port 43292.
Welcome to

```
____ __
/ __/__ ___ _____/ /__
_\ \/ _ \/ _ `/ __/ '_/
/___/ .__/\_,_/_/ /_/\_\ version 1.4.0
/_/
```

Using Scala version 2.10.4 (Java HotSpot(TM) 64-Bit Server VM, Java 1.7.0_71)
Type in expressions to have them evaluated.
Spark context available as sc
scala>

فصل یازدهم

برنامه‌نویسی هسته اسپارک

پوسته اسپارک

هسته‌ی اسپارک نقش اصلی را در این پروژه ایفا می‌کند که وظیفه‌ی آن پخش وظایف (task) های توزیع شده، زمان‌بندی و عملیات پایه‌ای) I/O ورودی و خروجی) است. اسپارک از ساختار داده ویژه‌ی خودش که به نام RDD شناخته می‌شود استفاده می‌کند که وظیفه آن جمع‌آوری منطقی داده تقسیم (Partitioned) شده بین ماشین‌ها است.

RDD ها به دو طریق ساخته می‌شوند، نوع اول با ارجاع دیتاست ها در سیستم‌های حافظه خارجی و نوع دوم با اعمال تغییر حالت)مثلmap :، fitter، reducer، (joinروی RDD های موجود.

سطح انتزاعی RDD از طریق language-integrated API (API های وابسته به زبان (مشخص می‌شود، با این روش پیچیدگی‌های برنامه‌نویسی ساده‌تر می‌شود، چون تغییر RDD های برنامه‌ها راحت‌تر از تغییر مجموعه داده‌های محلی (Local) است.

اسپارک دارای یک پوسته تعاملی است، یک ابزار قوی برای آنالیز داده به صورت تعاملی که در زبان Python یا Scala موجود می‌باشد. اولین سطح انتزاع اسپارک مجموعه‌ای توزیع شده از آیتم‌هاست که به آن "دیتاست های توزیع شده ارتجاعی (RDD) "می‌گویند.

RDDها یا با فرمت ورودی) Hadoop مثل فایل‌های(HDFS ، یا با تغییر دیگرRDD ها ساخته می‌شوند.

باز کردن پوسته اسپارک(Open Spark Shell)

برای باز کردن پوسته اسپارک از دستور زیر استفاده کنید:

```
$ Spark-shell
```

ساخت یک RDD ساده

ابتدا یک RDD ساده از روی فایل متنی (Text File) می‌سازیم. با دستور زیر یک RDD ساده می‌سازیم.

```
Scala > val inputfile = sc.textFile("input.text")
```

خروجی دستور بالا نوشته زیر است:

خروجی

```
inputfile: org.apache.spark.rdd.RDD[String] = input.txt MappedRDD[1] at textFile at:12
```

تحول RDD

RDD Transformations اشاره‌گرهایی به RDD جدید را برمی‌گرداند (Return می‌کند) و اجازه می‌دهد بین RDD ها وابستگی‌هایی ایجاد کنیم. هر RDD در زنجیر وابستگی (رشته وابستگی) برای محاسبه Data خودش یک تابع دارد و برای Parent خود یک اشاره‌گر (Pointer) دارد.

اسپارک تنبله، به این معنی که هیچ چیز اجرا نمی‌شود مگر این‌که شما یک Transformation یا Action را صدا کنید؛ که در این صورت یک Trigger Job ساخته و اجرا می‌شود. به قطعه کد زیر برای مثال word-count دقت کنید.

بنابراین، RDD Transformations یک مجموعه Data نیست اما یک مرحله در برنامه است (شاید تنها مرحله) که به اسپارک می‌گوید که چگونه Data را دریافت کند و با آن چکار کند.

در زیر لیست RDD Transformations ها را می‌بینید.

۱. Map(func)

یک دیتاست توزیع شده جدید را برمی‌گرداند، به وسیله پاس دادن هر المنت منبع از طریق تابع func

۲. Filter(func)

یک دیتاست جدید به وسیله انتخاب المنت‌های منبع وقتی که خروجی func، True باشد، برمی‌گرداند.

۳. FlatMap(func)

شبیه به Map است اما هر آیتم ورودی می‌تواند نگاشت (Map) شود به ۰ یا چند آیتم خروجی (func می‌تواند یک seq برگرداند به جای یک آیتم تکی)

۴. MapPartitions(func)

شبیه به Map است اما روی هر بلاک RDD به صورت جداگانه اجرا می‌شود. برای همین نوع func باید به شکل=> Iterator <T> <Iteraator<u> باشد وقتی روی RDD نوع T اجرا می‌شود.

۵. Map Partitioned with Index(func)

شبیه به Map Partitions ولی یک داده عددی به عنوان index پارتیشن (بلاک) هم می‌گیرد.

برای همین نوع func باید به شکل Interator<u =>(Int,Interator<T>)> باشد وقتی روی RDD نوع T اجرا می‌شود.

۶. Sample (Replacement,Fraction,Seed):

نمونه‌برداری از بخشی از داده، "با" یا "بی" جایگزین (Replacement)، با استفاده از Seed تولیدکننده اعداد تصادفی.

۷. Union (Other Dataset):

یک دیتاست جدید که شامل اجتماع (پیوند) دیتاست های آرگومان‌های منبع (ورودی) است، برمی‌گرداند.

۸. Intersection (Other Dataset):

یک دیتاست جدید که شامل اشتراک دیتاست های آرگومان‌های منبع (ورودی) است، برمی‌گرداند.

۹. Distinct ([numTasks]):

یک دیتاست جدید که از آرگومان‌های ورودی به صورت متمایز (غیر تکرار) برمی‌گرداند.

۱۰. Reduce By Key (fund, [numTasks]):

وقتی روی زوج دیتاست (k,V) اجرا شود، دیتاستی از زوج (K, V) برمی‌گرداند به طوری که مقدار هر Key جمع شماری شده (Aggregated) از تابع Reduce گرفته شده در پارامتر اول (Func) که باید؟ از (v,v)=>v باشد، شبیه groupByKey تعداد Reduce Task ها از طریق پارامتر دوم که اختیاری است تغییر می‌کند.

۱۱. groupByKey([numTasks]):

وقتی روی زوج دیتاست (k,V) اجرا شود، دیتاستی از زوج (K,Iterable) برمی‌گرداند.

نکته: اگر شما به منظور عملیات جمعی روی هر key از group استفاده می‌کنید (مثل Sum یا Average)، بهتر است از Reduce By Key یا Aggregated By Key استفاده کنید چون کارایی بالاتری دارد.

۱۲. Aggregated By Key (ZeroValue) (SeqOP,ComOP, [numTasks]):

وقتی روی زوج دیتاست (k,V) اجرا شود، خروجی دیتاستی از زوج (K, U) است که مقدار هر کلید (Key) عملیات جمعی روی Combine Function و مقدار "صفر" (Zero) مطلق است.
این اجازه داده می‌شود که نوع مقدار جمعی بتواند متفاوت باشد از نوع متغیر ورودی، (بدون اختصاص فضای اضافی). مثل groupByKey، تعداد Reduce Task ها از طریق پارامتر دوم که اختیاری است تغییر می‌کند.

۱۳. Sort By Key ([ascending], [numTasks]):

وقتی روی دیتاست زوج (k, v) که k به صورت ordered (ترتیبی) پیاده شده است، اجرا می‌شود، خروجی دیتاستی از (k, v) است که با key، sort (مرتب) شده است (یا asc یا desc) که به وسیله پارامتر ascending boolean، مشخص می‌شود.

۱۴. Cogroup (OtherDataset, [numTasks]):

وقتی روی دیتاست های (k, v) و (k, w) اجرا شود، خروجی دیتاستی از (k, (Iterable<v>, Iterable<w>)) است. این عملیات گاهی "group with" هم گفته می‌شود.

۱۵. Join (OtherDataset, [numTasks]):

وقتی روی دیتاست های (k, v) و (k, w) اجرا شود، خروجی دیتاستی از (k, (v,w)) است با تمام مقادیر برای هر key.
Outer join ها هم از طریق leftouterjoin و rightouterjoin و fullouterjoin پشتیبانی می‌شوند.

۱۶. Cartesian (OtherDataset):

وقتی روی دیتاست نوع T و U اجرا می‌شود، جواب دیتاستی از (T,U) است.

۱۷. Pipe (Command, [envVars]):

هر پارتیشن RDD رو از طریق کامند shell، Pipe (موازی) می‌کند مثلاً با Perl یا bash اسکریپت. المنت‌های RDD برای پردازش‌های stdin نوشته می‌شوند و خط‌های خروجی Stdout به عنوان رشته‌های RDD برمی‌گردند.

۱۸. Coalesce (numPartitions):

تعداد پارتیشن‌های RDD را به اندازه numPartitions کاهش می‌دهد. بعد از filter down کردن یک دیتاست بزرگ اجرا شدن این دستور خیلی عملکرد را بهبود می‌بخشد.

۱۹. RePartition (numPartitions):

داده‌ی RDD ها را به صورت رندوم تغییر می‌دهد برای ساخت تعداد بیش‌تر یا کمتر یا بالانس کردن پارتیشن‌ها. این عملیات معمولاً کل داده روی شبکه را به هم می‌ریزد.

۲۰. RePartition And Sort Within Partitions (Partitioner):

با توجه به partitioner گرفته شده دوباره پارتیشن‌بندی می‌کند و با هر پارتیشن جدید key های آن را sort می‌کند.

این کار خیلی بهتر از این است که اول repartition کنیم و بعد sort چون که این sort را وقتی انجام می‌دهد که تمام data ها به هم ریخته هستند.

✍ فعالیت‌ها

در زیر لیستی از Action ها را می‌بینید:

۱- Reduce(func):

المنت‌های dataset را با استفاده از تابع func (که دو آرگومان می‌گیرد و یک خروجی می‌دهد) تابع می‌تواند مبادله‌ای (commutative) و انجمنی (associative) باشد که می‌تواند به صورت موازی و به درستی آن را محاسبه کرد.

۲- Collect():

کل المان‌های دیتاست را به عنوان ارائه (array at the driver program) برمی‌گرداند.

معمولاً بعد از فیلتر (filter) یا عملیاتی که قسمت کوچکی از داده را برمی‌گرداند استفاده از collect مفید است.

۳- Count():

تعداد المان‌های دیتاست را برمی‌گرداند.

۴- First():

تعداد المان دیتاست را برمی‌گرداند، مانند take(۱) عمل می‌کند.

۵- Take(n):

یک ارائه با اولین ایندکس n، از المان‌های دیتاست را برمی‌گرداند.

۶- Take Sample(WithReplacement,num, [Seed]):

ارائه‌ای با مقدار رندوم num از المان‌های دیتاست برمی‌گرداند که "با" یا "بدون" (replacrment) (تعویض) است.

همچنین می‌توان با استفاده از پارامتر اختیاری seed الگوی رندوم را انتخاب کرد.

۷- Take Ordered(n, [ordering]):

تعداد n المنت اول RDD را به صورت خنثی و یا مرتب ([ordering]) برمی‌گرداند.

۸- Save As Text File(Path):

المنت‌های دیتاست را در مسیر گرفته شده (path) در هارد یا HDFS یا هر فایل سیستمی که hadoop ساپورت می‌کند، می‌نویسد.

اسپارک برای نوشتن هر خط روی فایل سیستم، tostring را صدا می‌زند.

۹- Save As Sequence File(Path)(Java and Scala):

المنت‌های dataset را به عنوان یک Sequence File هادوپ در مسیر گرفته شده می‌نویسد.

این کار این امکان را می‌دهد که RDD های زوج (value,name) بتوانند واسط (interface) قابل نوشتن هادوپ را پیاده کند.

همچنین روی انواعی که به صورت ضمنی قابل تغییر به نوشتنی (writeable) باشند این امکان در دسترس است.

(اسپارک تبدیلات را برای انواع ابتدایی نوع داده مانند: Int و Double و string پشتیبانی می‌کند.)

۱۰- Save As Object File(Path)(Java and Scala):

المنت‌های دیتاست را با فرمت ساده‌ی Java Serialization ذخیره می‌کند که این فایل‌ها می‌تواند با SparkContext.objectFile() لود (خوانده) می‌شود.

۱۱- Count By Key():

فقط برای RDD های نوع (k,v) است، خروجی آن یک hashmap از زوج‌های (k,Int) است که تعداد هر key در آن مشخص است.

۱۲- Foreach(func):

تابع func را روی هر المنت دیتاست اجرا می‌کند.

معمولاً برای آپدیت حافظه‌ها یا ارتباط با سیستم‌های حافظه خارجی استفاده می‌شود.

نکته: تغییر متغیرها در بیرون از حلقه foreach در حافظه‌ها امکان دارد یک رفتار غیر تعریف شده تلقی شود.

برای اطلاعات بیش‌تر مطلب Understanding Closures را ببینید.

برنامه‌نویسی باRDD

به کمک یک مثال می‌خواهیم actions ها و transformations های RDD را پیاده‌سازی کنیم.

مثال:

یک مثال "تعداد کلمات" را در نظر بگیرید: به این صورت که هر کلمه‌ای که در داکیومنت اضافه می‌شود شمرده می‌شود.

فرض کنید متن زیر را به عنوان ورودی در home directory تحت نام input.txt ذخیره می‌کنیم.

Input.txt: input file.

People are not as beautiful as they look,
as they walk or as they talk.
They are only as beautiful as they love,
as they care as they share.

برای انجام این تمرین مراحل زیر را انجام دهید.

Open-Spark-Shell:

برای این‌که فضای پوسته اسپارک را باز کنیم از دستور استفاده می‌کنیم.

معمولاً، اسپارک با اسکالا ساخته می‌شود، بنابراین برنامه اسپارک روی محیط اسکالا اجرا می‌شود.

```
$ spark-shell
```

اگر پوسته (shell) اسپارک به درستی باز شود، پیامی مانند پیام زیر مشاهده می‌کنید.

به خط آخر خروجی دقت کنید:

"Spark Context available as SC"به این معنی که محفظه‌ی اسپارک به طور خودکار شی contextاسپارک را با نام SC می‌سازد. قبل از شروع مرحله اول برنامه، شی SparkContextباید ساخته شود.

خروجی

```
Spark assembly has been built with Hive, including Datanucleus jars on classpath
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
15/06/04 15:25:22 INFO SecurityManager: Changing view acls to: hadoop
15/06/04 15:25:22 INFO SecurityManager: Changing modify acls to: hadoop
15/06/04 15:25:22 INFO SecurityManager: SecurityManager: authentication disabled;
ui acls disabled; users with view permissions: Set(hadoop); users with modify
permissions: Set(hadoop)
15/06/04 15:25:22 INFO HttpServer: Starting HTTP Server
15/06/04 15:25:23 INFO Utils: Successfully started service 'HTTP class server' on
port 43292.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.2.0
```

/_/
Using Scala version 2.10.4 (Java HotSpot(TM) 64-Bit Server VM, Java 1.7.0_71)
Type in expressions to have them evaluated.
Spark context available as sc
scala>

Create on RDD (ساخت RDD):

قدم اول این است که باید با Spark-Scala API فایل ورودی را بخوانیم و RDD را بسازیم.
دستور زیر برای خواندن فایل از محل (location) گرفته شده به کار می‌رود.
در اینجا RDD جدید تحت نام inputfile ساخته می‌شود. رشته گرفته شده در آرگومان (“”)textFileیک absolute Path برای ورودی است؛ بنابراین وقتی فقط اسم یک فایل گرفته می‌شود یعنی فایل در همین مکان هست که اسکالا هست.

```
scala> val inputfile = sc.textFile("input.txt")
```

اجرای transformation برنامه "تعداد کلمات"

هدف ما نگه‌داشتن تعداد کلمات روی فایل است، اول یک flat map برای جدا کردن هر خط به کلمات نیاز داریم((“”)flatMap(line=>line.split) :
بعد، هر کلمه را کلیدی حساب می‌کنیم که مقدار آن "۱" است(<word,1> =<key,value>) ، با استفاده از تابع((word,1))map(word=>
در آخر، تعداد کلیدها را با تطبیق کلیدهای شبیه هم کاهش می‌دهیم((- + -)reduceByKey)
برای پیاده‌سازی منطق برنامه "تعداد کلمات" دستورات زیر را ببینید. بعد از اجرای این دستورات شما خروجی مشاهده نخواهید کرد، چون شما هیچ Actions ای را صدا نکرده‌اید بلکه یک transformationصدا کرده‌اید.
یا باید به RDD جدید اشاره کنید یا به اسپارک بگویید با داده جدید گرفته شده چکار کند؟

```
Scala > Val Counts =
inputfile.flatMap(line=>line.split(“”)).map(word=>(word,1)).reduceByKey(- + -);
```

RDD جدید (Current RDD):

اگر حین کار با RDD خواستید اطلاعات بیش‌تری از آن بدانید از دستور زیر استفاده نمایید، این دستور توضیحاتی درباره RDD می‌دهد همچنین اطلاعاتی درباره وابستگی‌هایش(dependencies) (برای خطایابی به دست می‌آورید.

```
scala> counts.toDebugString
```

Caching The Transfrmations:

شما می‌توانید RDD را با استفاده از توابع ()Persist یا ()Cache ثابت (Persist)کنید.

بار اول این به عنوان Action محاسبه می‌شود که می‌تواند نگه داشته شود در حافظه روی یک گره.(node)
برای ذخیره نتایج میانی در حافظه از دستور زیر استفاده نمایید:

```
Scala > caunts.cashe( )
Applying The Action:
```

اعمال یکAction ، مثل ذخیره تمام نتایج transformations روی یک فایل است. آرگومان رشته‌ای برای تابع save AsTextFile(“”) یک مسیر مطلق (absolute) روی فولدر خروجی است.
برای ذخیره نتایج از دستور زیر استفاده کنید. در این مثال فولدر خروجی همین مکان فعلی است.

```
Scala > counts.saveAsTextFile( “output” )
```

چک کردن خروجی(checking the output)

یک ترمینال دیگر باز کنید و به home directory بروید (جایی که اسپارک در آن یک ترمینال اجرا می‌شود.)
از دستور زیر برای چک کردن خروجی استفاده کنید:

```
[ hadoop @localhost ~ ] $cd output/
[ hadoop @localhost output ] $ls -1
part-00000
part-00001
```

برای دیدن خروجی part-00000 از دستور زیر استفاده کنید:

```
[hadoop@localhost output]$ cat part-00000
```

خروجی

```
(people,1)
(are,2)
(not,1)
(as,8)
(beautiful,2)
(they, 7)
(look,1)
```

از دستور زیر هم برای دیدن خروجی part-00001 استفاده می‌کنیم:

```
[hadoop@localhost output]$ cat part-00001
```

خروجی

```
(walk,1 )
(or, 1 )
(talk, 1 )
(only, 1 )
(love, 1 )
```

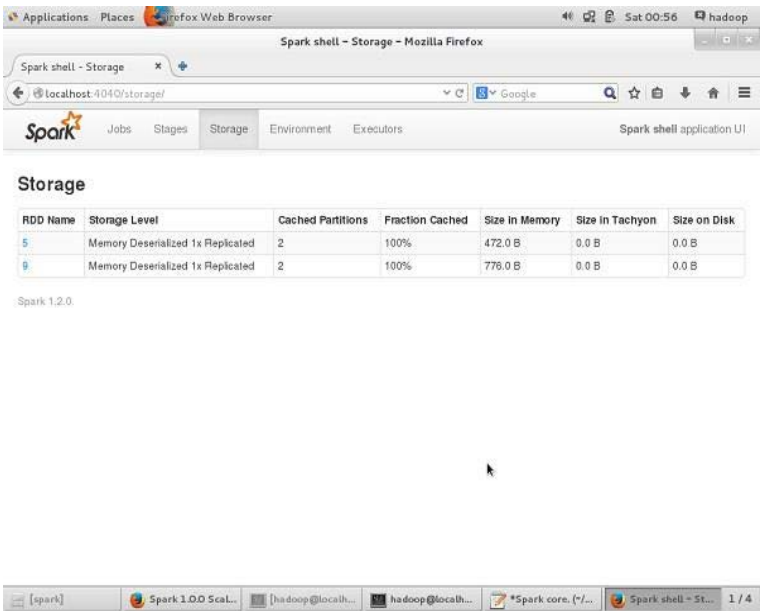
(care, 1)
(share, 1)

ذخیره‌سازی ثابتUN

قبل ازUN-Persisting ، اگر می‌خواهید فضایی که برای این برنامه اختصاص یافته را ببینید، Url زیر را در مرورگر اجرا کنید:

http://localhost: 4040

نتیجه مانند تصویر زیر است که فضاهای خالی و سایز حافظه را می‌بیند کـه روی shell اسـپارک اجرا می‌شوند.



اگر می‌خواهید UN-Persist کنید فضای ذخیـره یـک RDD بـه‌خصـوص را، بایـد از دسـتور زیـر استفاده کنید:

Scala > Counts.unpersist()

شما خروجی زیر را می‌بینید:

خروجی

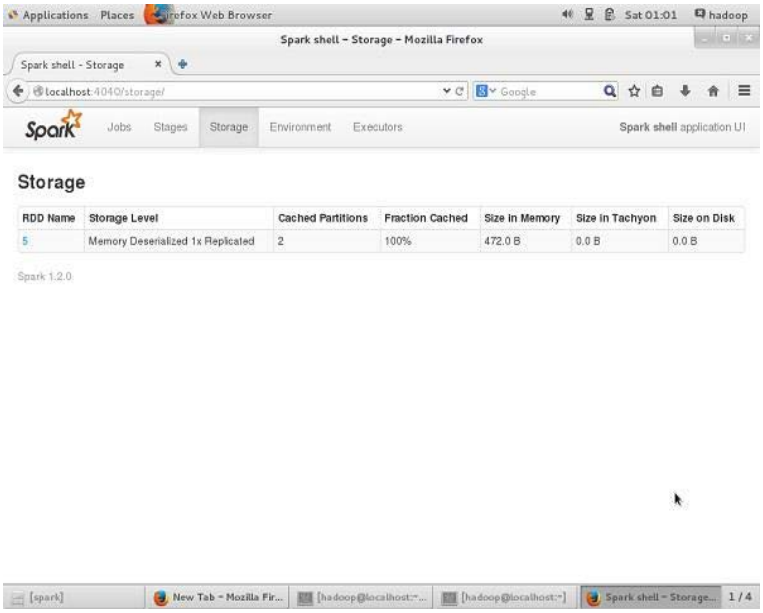
15/06/27 00:57:33 INFO ShuffledRDD: Removing RDD 9 from persistence list
15/06/27 00:57:33 INFO BlockManager: Removing RDD 9

15/06/27 00:57:33 INFO BlockManager: Removing block rdd_9_1
15/06/27 00:57:33 INFO MemoryStore: Block rdd_9_1 of size 480 dropped from memory (free 280061810)
15/06/27 00:57:33 INFO BlockManager: Removing block rdd_9_0
15/06/27 00:57:33 INFO MemoryStore: Block rdd_9_0 of size 296 dropped from memory (free 280062106)
res7: cou.type = ShuffledRDD[9] at reduceByKey at:14

دوباره برای چک کردن url زیر را در مرورگر بزنید:

http://localhost: 4040

خروجی مانند تصویر زیر است:



فصل دوازدهم

گسترش اسپارک

✍ گسترش اسپارک

برنامه اسپارک، برای گسترش خودروی خوشه (cluster) از Spark-submit که یک دستور روی Shellاست، استفاده می‌کند.

برای مدیریت خوشه‌های مربوطه از واسط uniform (یک شکل) استفاده می‌کند.

بنابراین، لازم نیست برای هر خوشه برنامه را config کنید.

مثال:

برگردیم به همان مثال تعداد کلمات و از دستورات shell استفاده کنیم برای روشن‌تر شدن موضوع، نمونه ورودی متن زیر ورودی ما است که در فایل in.txt ذخیره شده است.

People are not as beautiful as they look,
as they walk or as they talk.
They are only as beautiful as they love,
as they care as they share.

به برنامه زیر نگاه کنید:

SparkWordCount.scala

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark._
object SparkWordCount {
def main(args: Array[String]) {
val sc = new SparkContext("local", "Word Count", "/usr/local/spark", Nil, Map(), Map())
/* local = master URL; Word Count = application name; */
/* /usr/local/spark = Spark Home; Nil = jars; Map = environment */
/* Map = variables to work nodes */
/*creating an inputRDD to read text file (in.txt) through Spark context*/
val input = sc.textFile("in.txt")
```

```
24
/* Transform the inputRDD into countRDD */
val count=input.flatMap(line=>line.split(" "))
.map(word=>(word, 1))
.reduceByKey(_ + _)
/* saveAsTextFile method is an action that effects on the RDD */
count.saveAsTextFile("outfile")
System.out.println("OK");
}
}
```

برنامه بالا را در فایلی به نـام SparkWordCount.Scala ذخیـره کـرده و آن را در فولـدر Spark-application بریزید.

نکته: هنگام transforming از inutRDD به داخل CountRDD از ()flatMap بـرای تـوکن کـردن خط (lines) به کلمات استفاده می‌کنیم.

از تابع ()map برای شمارش تعداد تکرار کلمات و از تابع reduceByKey برای شمارش تکرار هـر کلمه استفاده می‌کنیم.

برای ثبت این برنامه مراحل زیر را طی کنید، تمام مراحل را از طریـق ترمینـال واقـع در پوشـه Spark-applicationاجرا کنید.

مرحله اول: دانلود:Spark jar

برای کامپایل Spark Core Java الزامی است، بنـابراین Spark-Core-2.10-1.3.0.jar را از لینـک http://mvnrepository.com/artifact/org.apache.spark/Spark-Core-2.10-1.3.0را دانلـــود کرده و فایل jar را داخل فولدر Spark-application ذخیره کنید.

مرحله دوم: کامپایل برنامه

با استفاده از دستور زیر برنامه‌ای که نوشتیم را کامپایل می‌کنیم، این دستور هـم بایـد از فولـدر Spark-applicationاجرا شود. در اینجا

/usr/local/spark/lib/spark-assembly-1.4.0-hadoop2.6.0.jar

یک jar با قابلیت ساپورت hadoop از کتابخانه Spark است.

```
$ scalac -classpath "spark-core_2.10-1.3.0.jar:/usr/local/spark/lib/spark-assembly-1.4.0-hadoop2.6.0.jar" SparkPi.scala
```

مرحله سوم: ساختJAR

فایل jar برنامه را از طریق دستور زیر می‌سازیم.

در اینجا wordcount نام فایل jar خروجی می‌باشد.

```
jar -cvf wordcount.jar SparkWordCount*.class spark-core_2.10-1.3.0.jar /usr/local/spark/lib/spark-assembly-1.4.0-hadoop2.6.0.jar
```

مرحله چهارم: ثبت (Submit) برنامه اسپارک:

با دستور زیر می‌توانید برنامه اسپارک را ثبت کنید.

```
spark-submit --class SparkWordCount --master local wordcount.jar
```

اگر دستور بالا با موفقیت اجرا شود شما باید خروجی زیر را بگیرید.

واژه‌ی ok در خروجی برای شناسایی کاربر است و آخرین خط برنامه است.

اگر با دقت به خروجی برنامه زیر دقت کنید، نکات مختلفی در آن پیدا می‌کنید، مانند:

-اسپارک با موفقیت روی پورت ۴۲۹۵۴ استارت خورد.

MemoryStore -با حافظه ۲۶۷,۳ MBاستارت خورد.

SparkUI -روی آدرس http://192.168.1.217:4040 استارت خورد.

Jar -فایل اضافه شد در/home/hadoop/piapplication/count.jar :

ResultStage -اول در ۰,۵۶۶ s تمام شد.

SparkUI -روی آدرس http://192.168.1.217:4040 استاپ شد.

MemoryStore -آزاد شد(Clear) .

خروجی

```
15/07/08 13:56:04 INFO Slf4jLogger: Slf4jLogger started
15/07/08 13:56:04 INFO Utils: Successfully started service 'sparkDriver' on port 42954.
15/07/08 13:56:04 INFO Remoting: Remoting started; listening on addresses:[akka.tcp://sparkDriver@192.168.1.217:42954]
15/07/08 13:56:04 INFO MemoryStore: MemoryStore started with capacity 267.3 MB
15/07/08 13:56:05 INFO HttpServer: Starting HTTP Server
15/07/08 13:56:05 INFO Utils: Successfully started service 'HTTP file server' on port 56707.
15/07/08 13:56:06 INFO SparkUI: Started SparkUI at http://192.168.1.217:4040
15/07/08 13:56:07 INFO SparkContext:
 Added JAR file:/home/hadoop/piapplication/count.jar at http://192.168.1.217:56707/jars/count.jar with timestamp 1436343967029
```

15/07/08 13:56:11 INFO Executor: Adding file:/tmp/spark-45a07b83-42ed-42b3-b2c2-823d8d99c5af/userFiles-df4f4c20-a368-4cdd-a2a7-39ed45eb30cf/count.jar to class loader
15/07/08 13:56:11 INFO HadoopRDD:
Input split: file:/home/hadoop/piapplication/in.txt:0+54
15/07/08 13:56:12 INFO Executor: Finished task 0.0 in stage 0.0 (TID 0). 2001 bytes result sent to driver
(MapPartitionsRDD[5] at saveAsTextFile at SparkPi.scala:11), which is now runnable
15/07/08 13:56:12 INFO DAGScheduler: Submitting 1 missing tasks from ResultStage 1 (MapPartitionsRDD[5] at saveAsTextFile at SparkPi.scala:11)
15/07/08 13:56:13 INFO DAGScheduler: ResultStage 1 (saveAsTextFile at SparkPi.scala:11) finished in 0.566 s
15/07/08 13:56:13 INFO DAGScheduler: Job 0 finished: saveAsTextFile at SparkPi.scala:11, took 2.892996 s
OK
15/07/08 13:56:13 INFO SparkContext: Invoking stop() from shutdown hook
15/07/08 13:56:13 INFO SparkUI: Stopped Spark web UI at http://192.168.1.217:4040
15/07/08 13:56:13 INFO DAGScheduler: Stopping DAGScheduler
15/07/08 13:56:14 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
15/07/08 13:56:14 INFO Utils: path = /tmp/spark-45a07b83-42ed-42b3-b2c2-823d8d99c5af/blockmgr-ccdda9e3-24f6-491b-b509-3d15a9e05818, already present as root for deletion.
15/07/08 13:56:14 INFO MemoryStore: MemoryStore cleared
15/07/08 13:56:14 INFO BlockManager: BlockManager stopped
15/07/08 13:56:14 INFO BlockManagerMaster: BlockManagerMaster stopped
15/07/08 13:56:14 INFO SparkContext: Successfully stopped SparkContext
15/07/08 13:56:14 INFO Utils: Shutdown hook called
15/07/08 13:56:14 INFO Utils: Deleting directory /tmp/spark-45a07b83-42ed-42b3-b2c2-823d8d99c5af
15/07/08 13:56:14 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!

مرحله پنجم: چک کردن خروجی

بعد از اجرای موفقیت‌آمیز برنامه، شما باید یک فولدر با نام outfile در فولدر Spark-application پیدا کنید.

دستورات زیر برای باز کردن و چک کردن لیست فایل‌ها درون فولدر outfile استفاده می‌شوند:

```
$ cd outfile
$ ls
Part-00000 part-00001 _SUCCESS
```

دستورات برای چک کردن فایل:part-00000

```
$ cat part-00000
(people,1)
(are,2)
(not,1)
(as,8)
(beautiful,2)
(they, 7)
(look,1)
```

دستورات برای چک کردن فایل:part-00001

```
$ cat part-00001
(walk, 1)
(or, 1)
(talk, 1)
(only, 1)
(love, 1)
(care, 1)
(share, 1)
```

برای این‌که بیشتر با دستور "Spark-Submit" آشنا شوید، بخش بعد را حتماً مطالعه کنید.

✍ سینتکس Spark-submit

spark-submit [options] [app arguments]

لیست دستورات:

لیست دستورات (options) و تعریف آن‌ها در زیر آمده است:

--masterSpark://host:part,mesos://host:part,yarn,or local

--deploy-mode می‌خواهیم درایور برنامه را به صورت locally یا "Client" اجرا کنیم یا روی یک ماشین worker داخل خوشه "Cluster" (پیش‌فرض: Client)

--class کلاس main برنامه شما (برای برنامه‌های جاوای اسکالا)

--name نام برنامه شما

--jars یک لیست جدا شده با کاما (،) از jar های محلی روی درایور و ClassPath های اجراکننده.

--packages یک لیست جدا شده با کاما (،) از مختصات maven را jar فایل‌ها روی درایور و ClassPath های اجراکننده.

--repositories یک لیست جدا شده با کاما (،) برای ریپوزیتوری های اضافی راه دور (Remote)، برای جستجو روی مختصات maven گرفته شده با - - Packages .

py-files-- یک لیست جدا شده با کاما (،) از فایل‌های .Zip و .egg یا .Py برای قرار دادن Python Path برای برنامه‌های پایتون

files-- یک لیست جدا شده با کاما (،) از فایل‌هایی که در فولدرهای کاری (Working) هر اجراکننده (Executor) قرار می‌گیرند.

conf (prop=val)--زوج‌های ویژگی / مقدار برای تغییر تنظیمات اسپارک (اختیاری)

properties-file-- مسیری که فایل اضافی برای تنظیمات دارد، اگر انتخاب نشود Conf/Spark-defaults می‌باشد.

driver-memory-- حافظه برای درایور (برای مثال: G, 100 M۲) (پیش‌فرض: M۵۱۲)

driver-java-options-- تنظیمات اضافی جاوا برای پاس دادن به درایور.

driver-library-path-- مسیر کتابخانه اضافی وارد شده برای پاس دادن به درایور.

driver-class-path-- مسیر کلاس اضافی وارد شده برای پاس دادن به درایور.

نکته: توجه داشته باشید که jar های اضافه شده با jars- - به صورت خودکار داخل classpath ها includeمی‌شوند.

executor-memory-- حافظه به ازای اجراکننده (برای مثال: M۱۰۰۰ یا G۲) (پیش‌فرض: G۱)

proxy-user-- کاربر می‌تواند خودش را جای دیگری جا بزند وقتی برنامه Submit (ثبت) می‌شود.

help, -h-- همین متن کمک را نشان می‌دهد و خارج می‌شود.

verbose, -v-- خروجی بیشتری از اشکال‌زدایی (خطایابی) (Debug) را چاپ می‌کند.

version-- ورژن اسپارک موجود را چاپ می‌کند.

driver-cores NUM-- هسته‌ها برای درایور (پیش‌فرض: ۱)

supervise-- اگر داده شود، در زمان شکست، درایور را ریستارت می‌کند.

kill-- اگر داده شود، درایورهای مشخص‌شده را می‌بندد (kill).

status-- اگر داده شود، حالات (status) درایور مشخص‌شده را درخواست می‌دهد.

total-executor-cores-- کل هسته‌ها برای کل اجراکننده‌ها.

executor-cores-- تعداد هسته‌ها به ازای هر اجراکننده. (پیش‌فرض در حالت YARN: 1، در حالت standalone روی هر worker تمام هسته‌ها در دسترس است.)

فصل سیزدهم

برنامه‌نویسی پیشرفته اسپارک

✍ متغیرهای همگانی

✍ مقدمه

اسپارک دو نوع مختلف متغیرهای اشتراکی را شامل می‌شود، یکی متغیرهای انتشار Broadcast) (Variablesو دومی انباره‌ها:(Acccumlators)

Broadcast Variablesبه میزان کفایت استفاده می‌شود، توزیع می‌کند مقادیر بزرگ را.

Accumulators برای جمع کردن اطلاعات کالکشن های ویژه به کار می‌رود.

✍ متغیرهای همگانی

متغیرهای همگانی (Broadcast) به برنامه‌نویسان اجازه می‌دهند که متغیر فقـط خوانـدنی کـش شده را روی هر ماشین نگه دارند که سریع‌تر است از انتقال یک کپی آن با Task (کار) مربوطه.
برای مثال برای گرفتن هر گره(node) ، گرفتن یک کپی از یک دیتاست ورودی بزرگ، می‌توانند با یک روش کارآمد استفاده می‌شوند.
اسپارک همچنین اجازه می‌دهد که متغیرهای همگانی توزیع شده استفاده کنند از الگوریتم‌هـای همگانی برای کاهش هزینه‌ی ارتباطی.
Actionهای اسپارک از طریق مجموعه‌ای از Stage (طبقه) ها اجرا می‌شـوند و بـا دسـتورالعمل "Shuffle"توزیع شده جدا می‌شوند. اسپارک به وسیله هر استیج بـه طـور خودکـار Broadcast می‌کند داده‌های متداول را که Task ها از آن‌ها استفاده می‌کنند.
دیتا (داده) ی Broadcast شده در این راه به صورت موازی شـده (Serialized) هسـت و قبـل از اجرای هر Task غیر موازی (deSerialized) می‌شود. این بدین معنی است که ساخت متغیرهـای

همگانی به طور واضح، فقط زمانی مفید است که Task ها سرتاسر چندین طبقه (Stage) به داده‌های یکسان نیاز دارند یا زمانی که داده را به صورت غیر موازی کش می‌کنند مهم است.
متغیرهای همگانی از Variable V با صدا زدن تابع SparkContext.Broadcast(V) ساخته می‌شوند.
متغیر همگانی یک پوشش اطراف V است و این مقدار با صدا زدن تابع Value به دست می‌آید.
کد زیر این داستان را نشان می‌دهد.

```
scala> val broadcastVar = sc.broadcast(Array(1, 2, 3))
```

خروجی

```
broadcastVar: org.apache.spark.broadcast.Broadcast[Array[Int]] = Broadcast(0)
```

بعد از این‌که متغیر همگانی ساخته شد، روی تمام توابعی که روی خوشه اجرا می‌شوند به جای Valueاستفاده می‌شود؛ بنابراین V بیش از یک‌بار به گره‌ها (nodes) انتقال داده نمی‌شود.
به‌علاوه، آبجکت V نباید بعد از broadcastتغییر کند به دلیل این‌که مطمئن باشیم که تمام گره‌ها (nodes) مقدار یکسانی از متغیر همگانی دریافت کرده‌اند.

انباره‌ها

انباره‌ها متغیرهایی هستند که فقط در میان عملیات انجمنی" (Associative) اضافه می‌شوند " (added)
بنابراین، برای پشتیبانی در موازی‌سازی (Paraller) کارآمد هستند.
از انباره‌ها برای پیاده‌سازی شمارنده‌ها) (Counters) مانند (MapReduce یا جمع‌ها (Sums) استفاده می‌کنند.
اسپارک به طور ذاتی از انباره‌های نوع عددی (numeric) پشتیبانی می‌کند و برنامه‌نویسان برای انواع دیگر پشتیبانی آن را اضافه کنند. اگر انباره‌ها با یک نام ساخته شوند، در Spark UIنمایش داده می‌شوند.
این می‌تونه مفید باشد برای فهمیدن پیشرفت اجرای طبقات) (Stages) نکته: این هنوز در پایتون پشتیبانی نمی‌شود(.
یک انباره از متغیر اولیه V با صدا زدن تابع SparkContext.accumulator(V) ساخته می‌شود، Taskها روی خوشه‌ای اجرا می‌شوند که از طریق تابع add یا عملگرهای - + اضافه می‌شوند (در پایتون و اسکالا) به هرحال آن‌ها نمی‌توانند این مقدار را بخوانند، فقط درایور برنامه می‌تواند مقدار انباره را با استفاده از تابع Value بخواند.
کد زیر نشان می‌دهد یک انباره چطور المان‌های ارائه را اضافه (add) می‌کند.

```
scala> val accum = sc.accumulator(0)
scala> sc.parallelize(Array(1, 2, 3, 4)).foreach(x => accum += x)
```

اگر شما می‌خواهید خروجی کد بالا را ببینید باید از دستور زیر استفاده کنید:

```
scala> accum.value
```

خروجی

```
res2: Int = 10
```

✍ عملیات عددیRDD

اسپارک به شما اجازه می‌دهد که روی داده‌های عددی عملیات مختلف انجام دهید، با استفاده از تابع‌های API از قبل تعریف شده.

عملیات عددی اسپارک به وسیله الگوریتم جریان پیاده‌سازی می‌شوند که به شما اجازه ساخت مدل را می‌دهد (در هر لحظه یک المنت)

این عملیات محاسبه می‌شوند و به عنوان آبجکت Status Counter با صدا زدن تابع Status() برگردانده می‌شود.

لیست توابع عددی موجود در Status Counter در زیر آمده است:

count()

تعداد المنت‌ها در RDD

Mean()

میانگین المنت‌ها در RDD

Sum()

مقدار کل المنت‌ها در RDD

Max()

بیش‌ترین مقدار میان تمام المنت‌ها در RDD

Min()

کمترین مقدار میان تمام المنت‌ها در RDD

Variance()

واریانس تمام المنت‌ها.

Stdev()

انحراف استاندارد.

نکته: اگر شما می‌خواهید فقط از یکی از این توابع استفاده کنید، شما می‌توانید تابع مشابه را مستقیم روی RDD اجرا کنید.

✍ پیوست ۱: نمونه یک پروژه با Hadoop

با استفاده از دانش زبان جاوا یک نمونه پروژه شمارش حروف را با هم اجرا می‌کنیم:

ابتدا فایل ورودی معادل زیر آماده می‌کنیم:

This is the example text file for word count example also knows as hello world example of the Hadoop ecosystem.
This example is written for the examples article of java code geek
The quick brown fox jumps over the lazy dog.
The above line is one of the most famous lines which contains all the english language alphabets.

و آن را با نام input.txt ذخیره می‌کنیم.

سپس کلاس‌های جاوا را معادل زیر می‌سازیم:

کلاس MapClass

```
package com.javacodegeeks.examples.wordcount;
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
/**
 * Map Class which extends MaReduce.Mapper class
 * Map is passed a single line at a time, it splits the line based on space
 * and generated the token which are output by map with value as one to be consumed
 * by reduce class
 * @author Raman
 */
public class MapClass extends Mapper<LongWritable, Text, Text, IntWritable>{

 private final static IntWritable one = new IntWritable(1);
 private Text word = new Text();
/**
 * map function of Mapper parent class takes a line of text at a time
 * splits to tokens and passes to the context as word along with value as one
 */
 @Override
 protected void map(LongWritable key, Text value, Context context)
 throws IOException, InterruptedException {
 String line = value.toString();
 StringTokenizer st = new StringTokenizer(line," ");
```

```
 while(st.hasMoreTokens()){
 word.set(st.nextToken());
 context.write(word,one);
 }

 }
}
```

کلاس ReduceClass

```
package com.javacodegeeks.examples.wordcount;
import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
/**
 * Reduce class which is executed after the map class and takes
 * key(word) and corresponding values, sums all the values and write the
 * word along with the corresponding total occurances in the output
 *
 * @author Raman
 */
public class ReduceClass extends Reducer<Text, IntWritable, Text, IntWritable>{
/**
 * Method which performs the reduce operation and sums
 * all the occurrences of the word before passing it to be stored in output
 */
 @Override
 protected void reduce(Text key, Iterable<IntWritable> values,
 Context context)
 throws IOException, InterruptedException {
 int sum = 0;
 Iterator<IntWritable> valuesIt = values.iterator();
 while(valuesIt.hasNext()){
 sum = sum + valuesIt.next().get();
 }
 context.write(key, new IntWritable(sum));
 }
}
```

کلاس WordCount

```
package com.javacodegeeks.examples.wordcount;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
/**
 * The entry point for the WordCount example,
 * which setup the Hadoop job with Map and Reduce Class
 *
 * @author Raman
 */
public class WordCount extends Configured implements Tool{
/**
 * Main function which calls the run method and passes the args using ToolRunner
 * @param args Two arguments input and output file paths
 * @throws Exception
 */
 public static void main(String[] args) throws Exception{
 int exitCode = ToolRunner.run(new WordCount(), args);
 System.exit(exitCode);
 }
/**
 * Run method which schedules the Hadoop Job
 * @param args Arguments passed in main function
 */
 public int run(String[] args) throws Exception {
 if (args.length != 2) {
 System.err.printf("Usage: %s needs two arguments <input> <output> files\n",
 getClass().getSimpleName());
 return -1;
 }
//Initialize the Hadoop job and set the jar as well as the name of the Job
 Job job = new Job();
 job.setJarByClass(WordCount.class);
 job.setJobName("WordCounter");
//Add input and output file paths to job based on the arguments passed
 FileInputFormat.addInputPath(job, new Path(args[0]));
 FileOutputFormat.setOutputPath(job, new Path(args[1]));
 job.setOutputKeyClass(Text.class);
 job.setOutputValueClass(IntWritable.class);
 job.setOutputFormatClass(TextOutputFormat.class);
```

```
//Set the MapClass and ReduceClass in the job
 job.setMapperClass(MapClass.class);
 job.setReducerClass(ReduceClass.class);
//Wait for the job to complete and print if the job was successful or not
 int returnValue = job.waitForCompletion(true)? 0:1;
 if(job.isSuccessful()) {
 System.out.println("Job was successful");
 } else if(!job.isSuccessful()) {
 System.out.println("Job was not successful");
 }

 return returnValue;
 }
}
```

در نهایت پس از اجرا خروجی مانند زیر می‌باشد:

```
Hadoop 1
The 2
This 2
above 1
all 1
alphabets. 1
also 1
article 1
as 1
brown 1
code 1
contains 1
count 1\
dog. 1
ecosystem. 1
english 1
example 4
examples 1
famous 1
file 1
for 2
fox 1
geek 1
hello 1
is 3
java 1
jumps 1
knows 1
language 1
lazy 1
line 1
lines 1
most 1
of 3
one 1
```

over 1
quick 1
text 1
the 6
which 1
word 1
world 1
written 1