به نام خدا

دانشگاه صنعتی امیرکبیر

Amirkabir University
of Technology

تکلیف چهارم- یادگیری ماشین کاربردی

استاد مربوطه: دکتر ناظرفرد

نام: زهرا اخلاقی

شماره دانشجویی: ۴۰۱۱۳۱۰۶۴

ایمیل: zahra.akhlaghi@aut.ac.ir

تابستان ۱۴۰۲

فهرست مطالب

Q1

(a

رویکرد کلی برای یافتن تعداد بهینه ویژگی های تصادفی به صورت زیر می باشد:
مجموعه داده ها به training و validation برای آموزش درختان تصمیم و برای ارزیابی عملکرد تقسیم میشود و میتوان با استفاده از آن عملکرد مدل را برای تعداد مختلف از ویژگی ها ارزیابی کرد.
برای این کار با تعداد کمی از ویژگی ها شروع و به تدریج تعداد آنها را به تعداد کل ویژگی ها افزایش می دهیم، برای هر مقدار کاندید، مراحل زیر را انجام میدهیم:
چندین درخت تصمیم را با استفاده از تعداد ویژگی های منتخب آموزش میدهیم و عملکرد آن را در مجموعه validation با استفاده از یک متریک مناسب (به عنوان مثال، دقت، امتیاز F1 و غیره) ارزیابی کرده و بهترین را انتخاب میکنیم مقدار ویژگی های تصادفی در آن درخت را به عنوان بهینه جواب در نظر گرفته میشود.
تعداد بهینه ویژگی‌های تصادفی می‌تواند به مجموعه داده و هدف بستگی داشته باشد. بنابراین، باید مقادیر مختلف را آزمایش و عملکرد را ارزیابی کرد تا بهترین ویژگی انتخاب شود.
در برخی مقاله ها و کتابها مقدار پیش فرض توصیه شده m=p/3 است برای مسایل رگرسیون $m=\sqrt{p}$ در نظر گرفته میشود ولی مقدار آن بستگی به هدف و داده ها دارد.

(b

الگوریتم های طبقه بندی مختلفی وجود دارد که هر کدام نقاط قوت و ضعف خاص خود را دارند. انتخاب بهترین الگوریتم به عوامل متعددی از جمله ماهیت مسئله، ویژگی داده ها، نیازهای محاسباتی، قابلیت تفسیر و منابع موجود بستگی دارد.
رگرسیون لجستیک: با مجموعه داده های کوچک به خوبی کار می کند و دارای سادگی و قابلیت تفسیر می باشد و در مسائل با قابلیت جداسازی خطی مفید میباشد ولی با الگوهای داده پیچیده به خوبی عمل نمیکند.
درختان تصمیم: درک و تفسیر آنها آسان است و می‌تواند ویژگی‌های عددی، دسته‌بندی، روابط غیرخطی، مقادیر از دست رفته و مقادیر پرت را کنترل کند ولی مستعد مشکل overfit هستند.
جنگل های تصادفی: مشکل overfit را کاهش می دهد و داده های با ابعاد بالا را مدیریت می کند و تخمینی از اهمیت ویژگی‌ها ارائه می دهد، ولی دارای معایب، پیچیدگی محاسباتی و عدم تفسیرپذیری در مقایسه با درخت تصمیم واحد است.
ماشین‌های بردار پشتیبانی (SVM): در داده های با ابعاد بالا موثر است، با مجموعه داده های کوچک تا متوسط به خوبی کار می کند و مرزهای تصمیم غیرخطی را میتواند با ترفند هایی کنترل می کند. ولی پیچیدگی محاسباتی برای مجموعه داده های بزرگ دارد و ممکن است برای داده هایی که نویز زیادی دارند عملکرد خوبی نداشته باشد.
Gaussian Naive Bayes: ساده و با داده های با ابعاد بالا به خوبی کار می کند، ویژگی های طبقه بندی را مدیریت

می کند ولی به دلیل فرض استقلال میان ویژگی ها، ممکن است روابط پیچیده را ثبت نکند.
برای انتخاب الگوریتم طبقه بندی،مناسب باید درک درستی نسبت به داده ها داشته باشیم و ویژگی آنها را تجزیه و تحلیل کنیم، مانند تعداد ویژگی ها، توزیع داده ها، وجود مقادیر دورافتاده یا مقادیر از دست رفته، و نیاز به مقیاس بندی ویژگی. درک پیچیدگی مسئله و اینکه به مرز تصمیم گیری خطی یا غیرخطی نیاز دارد و روابط بین ویژگی ها و متغیر هدف ساده یا پیچیده است مهم است، ارزیابی نیازهای محاسباتی و اهمیت تفسیرپذیری نیز از اهمیت زیادی برخوردار است.
میتوان با آزمایش روی الگوریتم های مختلف و مقایسه آنها بهترین را انتخاب کرد و میتوان از منحنی ROC برای ارزیابی عملکرد آنها استفاده کرد.

(c

قضیه No-Free Lunch بیان می‌کند که هیچ الگوریتم یادگیری ماشینی وجود ندارد که بتواند در تمام مسائل عملکرد بهینه داشته باشد. یادگیری گروهی (Ensemble learning) تکنیکی است که از ترکیب چند مدل برای بهبود عملکرد کلی و مقابله با معضل NFL استفاده می کند و به صورت زیر عمل میکند.
یادگیری گروهی چندین مدل را ترکیب می‌کند تا یک مدل «گروهی» قوی‌تر ایجاد کند. هر مدل پایه بر روی زیرمجموعه های مختلف داده آموزش داده می شود و با تجمیع مدل‌های متنوع، مجموعه طیف وسیع‌تری از الگوها و روابط درون داده‌ها را به تصویر می‌کشد.
مدل‌های پایه مختلف ممکن است نقاط قوت و ضعف متفاوتی داشته باشند و به طور جمعی می‌توانند جنبه‌های مختلفی از فضای مشکل را به تصویر بکشند. یادگیری گروهی به مدل ها اجازه می دهد تا با ترکیب پیش بینی های خود یکدیگر را تکمیل کنند و هدف آن دستیابی به عملکرد کلی بهتر است، استفاده از این روش حتی میتواند مشکل overfit را تا حد زیادی کاهش دهد.
به طور کلی، یادگیری گروهی مشکلات No-Free Lunch را با ترکیب چندین مدل برای بهره برداری از تنوع آنها، کاهش بیش از حد برازش، مدیریت روابط پیچیده، بهبود استحکام و دستیابی به عملکرد کلی بهتر در طیف وسیعی از حوزه های مشکل برطرف می کند.

(d

روش LASSO برای انتخاب ویژگی و منظم سازی در مدل های رگرسیون خطی استفاده می شود که عبارت جریمه را به تابع هدف رگرسیون خطی اضافه می کند و با کوچک کردن برخی ضرایب به صفر، sparse solutions را تشویق می کند. LASSO معمولاً به طور مستقیم برای Base Learner Selection در یادگیری گروهی استفاده نمی شود ولی به طور غیرمستقیم با روش های زیر میتواند بر آن تاثیر بگذارد:
انتخاب ویژگی: روش LASSO را می توان برای انتخاب ویژگی های مرتبط از مجموعه داده با کوچک کردن ضرایب ویژگی‌ها با اهمیت کمتر به صفر اعمال کرد که کاهش تعداد ویژگی‌ها، می‌تواند به ساده‌سازی فرآیند یادگیری گروهی کمک کند.

مرحله پیش پردازش: استفاده از LASSO به عنوان مرحله پیش پردازش برای انتخاب مهم ترین ویژگی ها استفاده میشود، سپس ویژگی‌های انتخاب‌شده می‌توانند به عنوان ورودی برای مدل‌های گروه مورد استفاده قرار گیرند و به طور غیرمستقیم بر انتخاب دانش‌آموز پایه تأثیر بگذارند. خاصیت منظم‌سازی LASSO را می‌توان برای الگوریتم های پایه در مجموعه اعمال کرد تا پیچیدگی آن‌ها را کنترل کرده و از برازش بیش از حد جلوگیری کند. تکنیک‌های منظم‌سازی می‌تواند در هر یادگیرنده پایه برای یافتن تعادل بین پیچیدگی مدل و عملکرد استفاده شود.
در حالی که LASSO معمولاً به طور مستقیم برای انتخاب یادگیرنده پایه در یادگیری گروهی استفاده نمی شود، می تواند به طور غیرمستقیم از طریق تکنیک های انتخاب ویژگی و منظم سازی بر روند تأثیر بگذارد و به بهبود عملکرد و قابلیت تفسیر مدل کمک کند.

(e

Bagging، Boosting و Sketching تکنیک‌های یادگیری گروهی هستند که هدفشان بهبود عملکرد مدل‌های یادگیری ماشینی با ترکیب چندین مدل پایه است. شباهت ها و تفاوت های آنها به صورت زیر میباشد:

شباهت ها:

۱. یادگیری گروهی: هر سه تکنیک شامل ترکیب چندین مدل پایه برای ایجاد یک مدل گروهی قدرتمندتر است.
۲. بهبود عملکرد: هدف اصلی آنها، افزایش عملکرد پیش بینی با استفاده از دانش جمعی از مدل های پایه است.
۳. کاهش اضافه برازش: با ترکیب مدل‌های پایه متنوع، هر سه روش با کاهش واریانس موجود در مدل‌های جداگانه، به کاهش اضافه برازش کمک می‌کنند.

تفاوت ها:

۱. آموزش مدل پایه: Bagging مدل‌های پایه را به طور مستقل بر روی زیر مجموعه‌های تصادفی مختلف داده‌های آموزشی با جایگزینی آموزش می‌دهد. هر مدل پایه به صورت موازی آموزش داده می شود و وزن یکسانی دارد. Boosting، مدل های پایه را به صورت متوالی آموزش می دهد و وزن نمونه‌های تمرینی را برای تاکید بر نمونه‌های دشوار تطبیق می‌دهد و هدف آن ایجاد یک مدل قوی با تصحیح متوالی اشتباهات مدل‌های قبلی است. Stacking شامل آموزش چندین مدل پایه است، اما به جای آموزش مستقل یا متوالی، آنها را به طور مشترک آموزش می دهد و پیش‌بینی‌های مدل‌های پایه را برای پیش‌بینی نهایی ترکیب کند.
۲. وزن مدل‌های پایه: در Bagging، به همه مدل‌های پایه وزن برابر داده می‌شود و Boosting وزن های مختلفی را به مدل های پایه بر اساس عملکرد آنها اختصاص می دهد. به مدل هایی که عملکرد بهتری دارند وزن بیشتری داده می شود و به مدل هایی که با نمونه های خاصی دست و پنجه نرم می کنند تمرکز بیشتری در تکرارهای بعدی داده می شود. Stacking یاد می گیرد پیش بینی های مدل های پایه را بر اساس عملکرد فردی آنها بسنجد با در نظر گرفتن نقاط قوت و ضعف مدل های پایه، پیش بینی ها را به طور موثر ترکیب میکند.
۳. پیچیدگی: Bagging و Stacking می تواند با مدل های پایه مختلف که به طور بالقوه پیچیده هستند، مانند درخت تصمیم، ماشین های بردار پشتیبان یا شبکه های عصبی کار کند و Boosting اغلب به یک مدل پایه ساده متکی

است و به طور مکرر عملکرد آن را بهبود می بخشد. Boosting بر ساخت یک مدل قوی با ترکیب چندین مدل ضعیف تمرکز دارد.

(f

در زمینه ماشین‌های بردار پشتیبان (SVM)، مسائل دوگانه و اولیه دو فرمول‌بندی ریاضی هستند که برای یافتن ابر صفحه بهینه برای طبقه‌بندی استفاده می‌شوند.

مسئله اولیه در SVM ها به دنبال یافتن ابر صفحه بهینه با بهینه سازی مستقیم وزن ها و بایاس های مدل خطی است و می توان آن را به عنوان یک مسئله بهینه سازی محدود فرموله کرد که معمولاً با هدف به حداکثر رساندن حاشیه بین دو کلاس در حالی که خطاهای طبقه بندی را به حداقل می رساند.

مسئله دوگانه، فرمول بندی مجدد مسئله اولیه است که شامل ضرب کننده های لاگرانژ است. با معرفی این ضرایب، مسئله دوگانه یک نمایش جایگزین از مسئله بهینه سازی را ارائه می دهد. می توان نشان داد که مسئله دوگانه دارای ویژگی محدب بودن است که حل آن را در برخی موارد آسان تر می کند.

ارتباط مسئله دوگانه در SVMها با استفاده از kernel اجازه می دهد تا داده ها را به طور ضمنی در یک فضای ویژگی با ابعاد بالاتر ترسیم کنند، که می تواند جداسازی داده های غیرخطی قابل جداسازی را با استفاده از یک ابر صفحه خطی ممکن کند. SVM می تواند به طور موثر در این فضای ویژگی با ابعاد بالاتر بدون محاسبه صریح تبدیل عمل کند. مسایل دوگانه همچنین بینشی را در مورد اهمیت بردارهای پشتیبانی در SVM ها ارائه می دهد. ضرب‌کننده‌های لاگرانژ به‌عنوان راه‌حل برای مسئله دوگانه، ارتباط هر نمونه آموزشی را در تعیین ابرصفحه بهینه نشان می‌دهد.

(g

به‌طور پیش‌فرض، طبقه‌بندی‌کننده‌های SVM به‌طور مستقیم امتیاز یا احتمال اطمینان ارائه نمی‌کنند. در SVM ها هدف یافتن یک ابر صفحه بهینه است که این دو کلاس را از هم جدا می کند. مرز تصمیم با موقعیت نقاط داده نسبت به ابر صفحه تعیین می شود. روش های زیر برای تخمین امتیاز یا احتمالات از خروجی SVM وجود دارد:

۱. فاصله: امتیاز اطمینان در SVM ها معمولاً با محاسبه فاصله از مرز تصمیم به دست می آید. مقادیر مثبت یک پیش بینی برای یک کلاس را نشان می دهد، در حالی که مقادیر منفی نشان دهنده یک پیش بینی برای کلاس دیگر است. بزرگی امتیاز نشان دهنده سطح اطمینان است. مقادیر مطلق بزرگتر نشان دهنده اطمینان بیشتر است، در حالی که مقادیر نزدیک به صفر نشان دهنده اطمینان کمتر است..

۲. Platt Scaling: تکنیکی است که خروجی SVM را برای تخمین احتمالات کالیبره می کند. این شامل آموزش یک مدل رگرسیون لجستیک بر روی خروجی های SVM به عنوان ورودی و استفاده از مدل رگرسیون لجستیک برای تخمین احتمالات کلاس است.

۳. پشتیبانی بردار احتمال: برخی از پیاده سازی های SVM، مانند کتابخانه LibSVM، روشی به نام "svm_predict_probability" ارائه می کنند که می تواند احتمالات را مستقیماً تخمین بزند. این روش به طور داخلی از یک تغییر مقیاس پلات برای تخمین احتمالات کلاس بر اساس فاصله تا ابر صفحه استفاده می کند.

SVM ها ذاتاً مدل های احتمالی مانند رگرسیون لجستیک نیستند. احتمالات یا امتیازات به‌دست‌آمده ممکن است معنای آماری مشابهی با مدل‌های احتمالی واقعی نداشته باشند. بنابراین بهتر است از مدل‌های دیگری که به صراحت برای خروجی احتمالی طراحی شده‌اند استفاده شود.

(h

زمانی که طبقه‌بندی‌کننده SVM با هسته RBF با مجموعه داده‌های آموزشی مناسب نیست، می‌توان پارامترهای γ (گاما) و C را تنظیم کرد تا به طور بالقوه عملکرد مدل را بهبود ببخشید.

فراپارامتر گاما (γ)

افزایش گاما باعث می شود هسته RBF نسبت به هر نقطه داده در مجموعه آموزشی حساس تر شود که منجر به مرزهای تصمیم گیری پیچیده تر می‌شود و ممکن است باعث overfit شود. برعکس، کاهش γ باعث می‌شود هسته RBF نسبت به نقاط داده فردی حساس‌تر نباشد و در نتیجه مرزهای تصمیم‌گیری هموارتر و عمومی‌تر شود که ممکن است اعث مشکل unserfit شود.

فراپارامتر C:

هایپرپارامتر C قدرت منظم سازی را در SVM ها کنترل می کند. مقدار بالاتر C به SVM اجازه می دهد تا نقض حاشیه کمتری داشته باشد، که به طور بالقوه منجر به مرز تصمیم گیری سخت تر و افزایش پیچیدگی می شود. اگر مرز تصمیم اولیه خیلی ساده باشد یا حاشیه زیادی داشته باشد، افزایش C می تواند به رفع عدم تناسب کمک کند. برعکس، کاهش C اثر منظم سازی را تقویت می کند و امکان نقض حاشیه بیشتر و مرز تصمیم گیری گسترده تر را فراهم می کند.

تأثیر این تنظیمات فراپارامتر ممکن است بسته به مجموعه داده و مشکل موجود متفاوت باشد.

(i

مسئله بهینه سازی فرموله شده توسط ماشین های بردار پشتیبان (SVM) یک مسئله برنامه نویسی درجه دوم است. تکنیک های مختلفی وجود دارد که می توان برای حل موثر این مشکل استفاده کرد.

Sequential Minimal Optimization: یک الگوریتم محبوب برای حل مسائل بهینه سازی SVM است که مسئله بزرگ برنامه نویسی درجه دوم را به یک سری از مسائل فرعی کوچکتر تقسیم می کند که می توانند به صورت تحلیلی حل شوند. این الگوریتم به طور مکرر دو ضریب لاگرانژ را انتخاب می کند و آنها را برای بهبود راه حل به روز می کند. این روند تا رسیدن به همگرایی ادامه می یابد.

Interior-Point Methods: این روش‌ها با حل رشته‌ای از مسائل که به مشکل اصلی نزدیک می‌شوند، کار می‌کنند و در عین حال اطمینان می‌دهند که راه‌حل‌ها امکان‌پذیر باقی می‌مانند. این روش SVM خطی و غیرخطی را مدیریت کنند.

Quadratic Programming Solvers: چندین بسته نرم افزاری تخصصی و کتابخانه های کارآمدی برای مسائل برنامه نویسی درجه دوم ارائه می کنند. این حل کننده ها از الگوریتم هایی استفاده می کنند که به طور خاص برای حل مسائل بهینه سازی درجه دوم طراحی شده اند و می توانند هر دو فرمول SVM خطی و غیرخطی را مدیریت کنند.

Stochastic Gradient Descent: یک الگوریتم بهینه سازی تکراری است که معمولاً برای آموزش SVM در مقیاس بزرگ استفاده می شود. این راه حل را با به روز رسانی پارامترهای مدل بر اساس یک زیرمجموعه تصادفی از نمونه های آموزشی در هر تکرار تقریبی می کند. SGD کارآمد و مقیاس پذیر است و برای مدیریت مجموعه داده های بزرگ مناسب است.

انتخاب تکنیک بهینه سازی به عواملی مانند اندازه مسئله، منابع محاسباتی و دقت مطلوب بستگی دارد. تکنیک‌های مختلف از نظر سرعت هم‌گرایی، نیازهای حافظه و مقیاس‌پذیری معاوضه‌های متفاوتی دارند.

(j

SVM ها، به طور پیش فرض، تخمین احتمالی مستقیمی را برای پیش بینی ها ارائه نمی دهند ولی رویکردهایی برای تخمین احتمالات از مدل های SVM، به ویژه برای مسائل طبقه بندی باینری وجود دارد. یکی از روش های رایج Platt scaling است که یک تکنیک پس از پردازش است. که به صورت زیر عمل میکند.

آموزش مدل SVM با استفاده از داده های آموزشی، بهینه سازی پارامترهای مدل با توجه به تابع هدف SVM، سپس فاصله هر مثال آموزشی را از مرز تصمیم SVM بدست آورده. این فاصله ها می توانند مثبت یا منفی باشند.

یک مجموعه اعتبار سنجی که جدا از داده های آموزشی است وجود دارد. برای هر مثال آموزشی، فاصله آن را از مرز تصمیم محاسبه کرده و از آن به عنوان ورودی مدل رگرسیون لجستیک استفاده میشود. مدل رگرسیون لجستیک را با استفاده از فواصل به عنوان ورودی و برچسب های کلاس مربوطه (0 یا 1) به عنوان هدف آموزش داده و مدل رگرسیون لجستیک را با بهینه‌سازی پارامترهای آن برای تناسب با مجموعه اعتبارسنجی کالیبره کنید تا تخمین های احتمالی را ارائه دهد که با احتمالات کلاس واقعی تناسب بهتری داشته باشد.

برای یک نمونه جدید و دیده نشده، فاصله آن را از مرز تصمیم با استفاده از مدل SVM محاسبه کرده و از مدل رگرسیون لجستیک کالیبره شده برای ترسیم فاصله تا تخمین احتمال استفاده کنید.

تخمین های احتمال به دست آمده باید با احتیاط تفسیر شوند. SVMها اصولاً برای به حداکثر رساندن حاشیه بین کلاس‌ها طراحی شده‌اند و تخمین‌های احتمالی که از مقیاس‌بندی Platt به دست می‌آیند، تقریب‌های تعقیبی هستند. این تخمین ها ممکن است معنای آماری مشابهی با مدل های احتمالی واقعی مانند رگرسیون لجستیک نداشته باشند. برخی از پیاده سازی های SVM، مانند LIBSVM، توابع یا گزینه های داخلی را برای فعال کردن تخمین احتمال ارائه می دهند.

(k

الف)

برای به دست آوردن هسته $K(x, z) = cK1(x, z)$، می توانیم از مفهوم مقیاس بندی در فضای هسته استفاده کنیم. از آنجایی که $K1(x,z)$ یک هسته است، مربوط به نگاشت ویژگی $\varphi1(x)$ و $\varphi1(z)$ در فضایی با ابعاد بالاتر است، هسته در این رابطه با ضرب کرنل $K1(x, z)$ در یک ثابت c بدست میاید. که در فضای ویژگی، مربوط به مقیاس بندی نگاشت

ویژگی φ1(x) و φ1(z)) با √c است. بنابراین، نگاشت ویژگی φ(x) برای هسته مقیاس شده را می توان به صورت زیر تعریف کرد:

φ(x) = √c * φ1(x)

بنابراین، با استفاده از φ1 برای نگاشت فضای ورودی Rn به فضای ویژگی Rd و مقیاس بردارهای مشخصه حاصل با √c، هسته را به دست می آوریم.

ب)

برای به دست آوردن نگاشت هسته K(x, z) = K1(x, z)K2(x, z) می توانیم از مفهوم حاصلضرب استفاده کنیم. به ترتیب برای هسته های K1 و K2. این نگاشت ها فضای ورودی Rn را به فضای Rd مرتبط با هر دو هسته ترسیم میکنند. حاصلضرب هسته‌های K1 و K2 را می‌توان به‌عنوان ضرب بین نگاشت‌های ویژگی φ1(x) و φ1(z) برای K1 و φ2(x) و φ2(z) برای K2 نشان داد. به عبارت دیگر:

K(x, z) = K1(x, z)K2(x, z) = ⟨φ1(x), φ1(z)⟩ ⟨φ2(x), φ2(z)⟩

برای به دست آوردن نگاشت ویژگی φ(x) برای هسته K(x, z) = K1(x, z)K2(x, z)، می‌توانیم آن را به عنوان ضرب عنصری φ1 و φ2 تعریف کنیم:

φ(x) = φ1(x) ⊙ φ2(x)

در عبارت بالا، ⊙ نشان دهنده ضرب عنصر است. نگاشت ویژگی حاصل φ(x) فضای ورودی Rn را به فضای بعدی Rd بالاتر مرتبط با هسته K(x, z) = K1(x, z)K2(x, z) ترسیم می کند. بنابراین، با استفاده از نگاشت ویژگی‌های φ1 و φ2 و ضرب عنصری خروجی‌های آنها، می‌توانیم هسته K(x, z) = K1(x, z)K2(x, z) را بدست آوریم.

(l

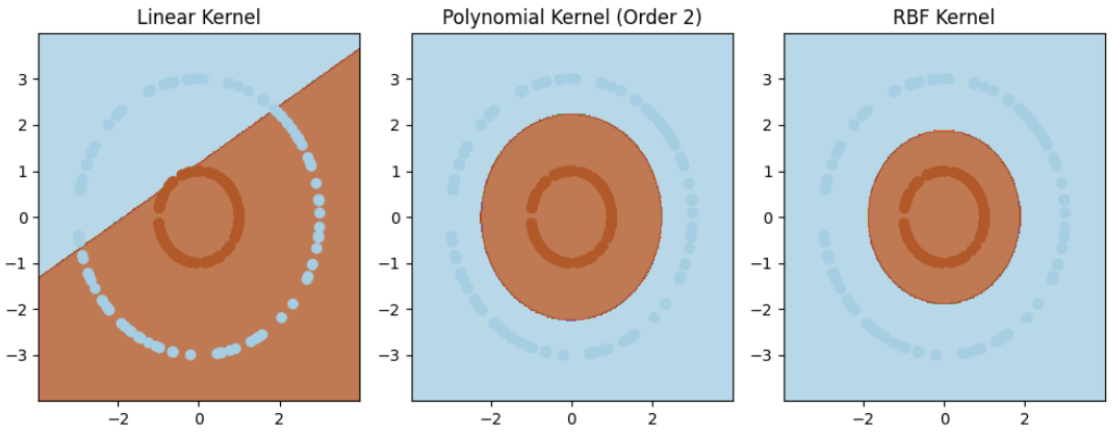
در این شکلها دو کلاس با مرکز یکسان، اما با شعاع های متفاوت داریم. یک کلاس دارای شعاع 1 و کلاس دیگر دارای شعاع 3 است.

هسته خطی ساده ترین هسته است و یک مرز تصمیم گیری خطی را در نظر می گیرد. می توان آن را به عنوان یک خط مستقیم در یک فضای دو بعدی تجسم کرد. با این حال، در این مورد، با کلاس‌هایی که شکل‌های دایره‌ای دارند، هسته خطی ممکن است نتواند ماهیت غیرخطی داده‌ها را به طور دقیق ثبت کند. مرز تصمیم ایجاد شده توسط طبقه بندی کننده SVM خطی نمی‌تواند این دو کلاس را به طور موثر جدا کند.

هسته چند جمله ای مرتبه 2 مرزهای تصمیم گیری غیر خطی را امکان پذیر می کند. ویژگی های ورودی را به فضایی با ابعاد بالاتر ترسیم می کند، هسته چند جمله ای می تواند اشکال دایره ای دو کلاس را بهتر از هسته خطی به تصویر بکشد. مرز تصمیم یک خط منحنی است که بهتر می تواند نقاط داده را بر اساس شعاع آنها جدا کند.

هسته RBF (تابع پایه شعاعی) می تواند با در نظر گرفتن شباهت بین نقاط داده در یک فضای با ابعاد بالا، مرزهای تصمیم گیری پیچیده را به تصویر بکشد. هسته RBF وزن هایی را به نقاط داده همسایه اختصاص می دهد و نقاط

نزدیکتر وزن بیشتری دارند. در این مورد، هسته RBF می تواند به طور موثر مرز تصمیم گیری دایره ای بین دو کلاس را یاد بگیرد و داده ها را بر اساس شعاع آنها جدا می کند.



Q2

(a

```
make_classification(
```

```
    n_samples=1000, # 1000 observations
    n_features=5, # 5 total features
    n_informative=3, # 3 'useful' features
    n_classes=2, # binary target/label
    random_state=999 # if you want the same results as mine
)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   X1      1000 non-null   float64
 1   X2      1000 non-null   float64
 2   X3      1000 non-null   float64
 3   X4      1000 non-null   float64
 4   X5      1000 non-null   float64
 5   y       1000 non-null   int64
dtypes: float64(5), int64(1)
memory usage: 47.0 KB
```

```
Train X:(800, 5) Y:(800,)
Val X:(200, 5) Y:(200,)
```

(b

```
param_grid = {
    'C': [0.1, 1, 10],
    'gamma': [0.1, 0.01, 0.001],
    'kernel': ['linear','poly','rbf']
}
```

```
Best Parameters: {'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
Best Score: 0.8675
```

(c

Multiple Kernel Learning تکنیکی است که در ماشین‌های بردار پشتیبان (SVM) برای ترکیب چند هسته برای بهبود عملکرد طبقه‌بندی استفاده می‌شود. در SVM سنتی، یک تابع هسته واحد برای اندازه گیری شباهت بین نقاط داده

استفاده می شود. با این حال، MKL به ما اجازه می دهد تا چندین توابع هسته را ترکیب کنیم تا به نتایج طبقه بندی بهتری دست یابیم. MKL انعطاف پذیری را با اجازه دادن به هسته های مختلف به صورت وزن دار برای تشکیل یک هسته ترکیبی ارائه می دهد. وزن های اختصاص داده شده به هر هسته سهم آنها را در مرز تصمیم نهایی تعیین می کند. پارامترها و انواع MKL به شرح زیر است:

توابع هسته: MKL استفاده از توابع مختلف هسته را امکان پذیر می کند. برخی از توابع کرنل که معمولاً مورد استفاده قرار می‌گیرند عبارتند از خطی، چند جمله‌ای، گاوسی (RBF)، سیگموئید و ....

وزن هسته: وزن های تخصیص یافته به هر کرنل اهمیت آنها را در مرز تصمیم گیری نهایی مشخص می کند. این وزن ها را می توان ثابت یا از داده ها در طول فرآیند بهینه سازی یاد گرفت.

روش های ترکیبی: MKL روش های مختلفی را برای ترکیب توابع جداگانه هسته ارائه می دهد. برخی از روش های ترکیبی رایج عبارتند از ترکیبات خطی ساده، ترکیبات محدب و ترکیبات غیر خطی.

استفاده از چندین هسته در MKL، می‌تواند از نقاط قوت هر هسته بهره برده و ویژگی‌های متفاوتی از داده‌ها را ثبت کند. این می تواند توانایی مدل را برای مدیریت الگوهای پیچیده و بهبود عملکرد طبقه بندی بهبود بخشد.

(d

عملکرد هر یک از کرنل ها به صورت مجزا:

Kernel: linear

	precision	recall	f1-score	support
0	0.90	0.19	0.31	102
1	0.54	0.98	0.69	98
accuracy			0.57	200
macro avg	0.72	0.58	0.50	200
weighted avg	0.72	0.57	0.50	200

Kernel: poly

	precision	recall	f1-score	support
0	0.94	0.44	0.60	102
1	0.62	0.97	0.76	98
accuracy			0.70	200
macro avg	0.78	0.71	0.68	200
weighted avg	0.78	0.70	0.68	200

Kernel: rbf

	precision	recall	f1-score	support
0	0.89	0.75	0.81	102
1	0.77	0.91	0.84	98
accuracy			0.82	200
macro avg	0.83	0.83	0.82	200
weighted avg	0.84	0.82	0.82	200

ترکیب کرنل ها با یکدیگر با استفاده از mkl

	precision	recall	f1-score	support
0	0.88	0.75	0.81	102
1	0.78	0.89	0.83	98
accuracy			0.82	200
macro avg	0.83	0.82	0.82	200
weighted avg	0.83	0.82	0.82	200

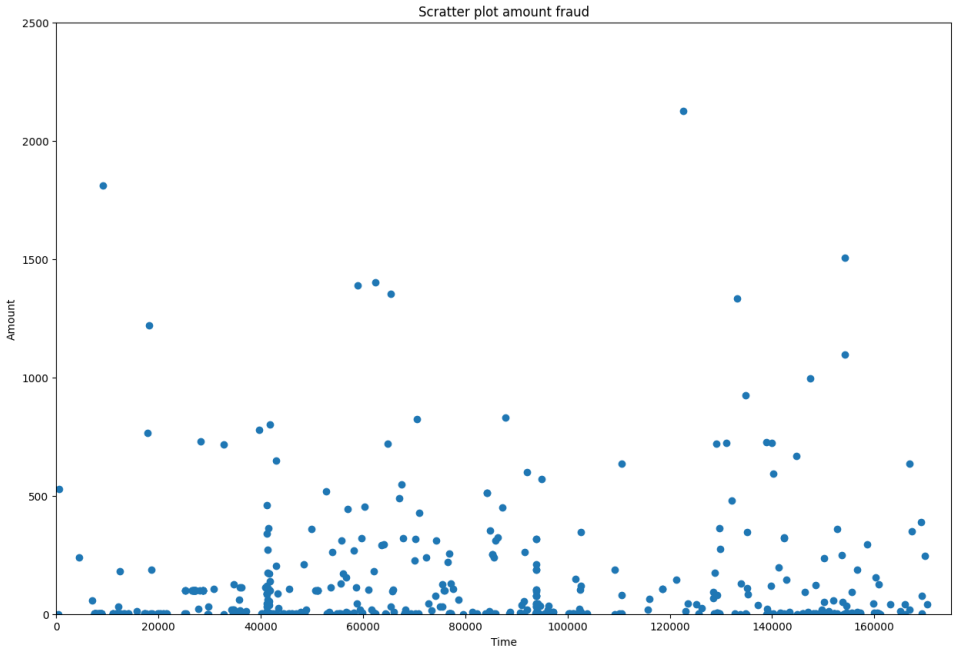
Q3

(a

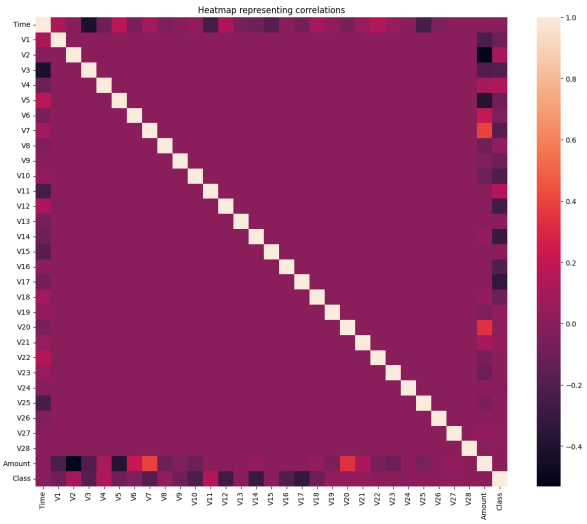
	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	...	2.848070e+05	2.848070e+05
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16	1.487313e-15	-5.556467e-16	1.213481e-16	-2.406331e-15	...	1.654067e-16	-3.568593e-16
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00	1.098632e+00	...	7.345240e-01	7.257016e-01
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01	-1.343407e+01	...	-3.483038e+01	-1.093314e+01
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01	-6.430976e-01	...	-2.283949e-01	-5.423504e-01
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02	-5.142873e-02	...	-2.945017e-02	6.781943e-03
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01	5.971390e-01	...	1.863772e-01	5.285536e-01
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01	1.559499e+01	...	2.720284e+01	1.050309e+01

8 rows × 31 columns

شکل زیر تقلب را با توجه به زمان آنها نمایش دهید



شکل زیر همبستگی داده ها را نسبت به یکدیگر نشان میدهد.



مجموعه داده دارای مشکل عدم تعادل می باشد، زیرا استفاده از داده‌ها آنطور که هست ممکن است منجر به رفتار ناخواسته در supervised classifier می‌شود. به این معنی که اگر همه داده ها به‌عنوان غیر تقلبی برچسب‌گذاری شود، الگوریتم دقت بالایی را خواهد داشت. برای این مشکل از under-sampling استفاده شده است.
برای این کار، ۸۰ درصد داده ها به عنوان آموزش در نظر گرفته میشود و از این مجموعه داده های تقلب و غیر تقلب جداشده و متناسب با داده های تقلب، نمونه ای از داده های غیر تقلب در نظر گرفته میشود تا داده آموزش دارای نسبت تقریبا برابر داده تقلب و غیر تقلب باشد و آموزش مدل عملکرد بدی نداشته باشد.

```
best_parems = {'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}
              precision    recall  f1-score   support

           0       1.00      0.99      0.99     56887
           1       0.09      0.85      0.16        75

    accuracy                           0.99     56962
   macro avg       0.55      0.92      0.58     56962
weighted avg       1.00      0.99      0.99     56962
```

(b

One-Class SVM نمونه‌ای از الگوریتم SVM است که به‌طور خاص برای تشخیص موارد پرت طراحی شده است. بر خلاف SVM های سنتی که برای وظایف طبقه بندی باینری استفاده می شوند، One-Class SVM بر روی مجموعه ای از داده ها آموزش داده می شود که تنها یک کلاس را نشان می دهد، که معمولاً کلاس اکثریت (نمونه های معمولی) است. هدف، ایجاد مرزی است که نمونه های عادی را در بر می گیرد و به طور موثری موارد پرت را به عنوان نمونه هایی که خارج از این مرز قرار می گیرند، شناسایی می کند.

```
Number of outliers detected in training data: 37
```

(c

برای ارزیابی اینکه مدل با چه دقتی داده‌های پرت را روی مجموعه داده اعتبارسنجی تشخیص می‌دهد، می‌توان از معیارهای مختلفی مانند precision, recall, F1-score accuracy استفاده کرد. در زمینه تشخیص موارد پرت، precision و recall معیارهای بسیار مفیدی هستند:
Precision: دقت بالا نشان می دهد که مدل به درستی درصد بالایی از نقاط پرت واقعی را از نمونه های پیش بینی شده به عنوان نقاط پرت شناسایی می کند.

recall: نشان می دهد که مدل در شناسایی بخش بزرگی از نقاط پرت واقعی موثر است.

در زمینه تشخیص پرت، دقت و یادآوری اغلب در یک رابطه از اهمیت بالایی برخوردار است. افزایش آستانه برای طبقه‌بندی نمونه‌ها به عنوان موارد پرت می‌تواند منجر به دقت بالاتر اما یادآوری کمتر شود و بالعکس.

انتخاب بین دقت و فراخوانی بستگی به الزامات خاص در تشخیص موارد پرت دارد:

- اگر اولویت به حداقل رساندن موارد Fp است (به عنوان مثال، کاهش شانس طبقه‌بندی اشتباه نمونه‌های عادی به عنوان موارد پرت)، پس دقت یک معیار بسیار مهم است. دقت بالا تضمین می‌کند که نمونه‌هایی که به‌عنوان موارد پرت شناسایی می‌شوند، به احتمال زیاد موارد پرت واقعی هستند.

- اگر اولویت این است که موارد پرت واقعی به تصویر کشیده شوند، حتی اگر به معنای پذیرش برخی موارد FP باشد، پس یادآوری مهمتر است. یک یادآوری بالا نشان می دهد که بخش بزرگی از نقاط پرت واقعی توسط مدل شناسایی می شوند.

```
Precision: 0.0011586671816298584
Recall: 0.0011586671816298584
F1-score: 0.0011586671816298584
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_cla
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_cla
  warnings.warn(
```

Q4

(a

برای پیش پردازش نیاز است ابتدا اطلاعاتی درباره دیتاست داشته باشیم، که به صورت زیر است:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3755 entries, 0 to 3754
Data columns (total 11 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   work_year           3755 non-null   int64 
 1   experience_level    3755 non-null   object
 2   employment_type     3755 non-null   object
 3   job_title           3755 non-null   object
 4   salary              3755 non-null   int64 
 5   salary_currency     3755 non-null   object
 6   salary_in_usd       3755 non-null   int64 
 7   employee_residence  3755 non-null   object
 8   remote_ratio        3755 non-null   int64 
 9   company_location    3755 non-null   object
 10  company_size        3755 non-null   object
dtypes: int64(4), object(7)
memory usage: 322.8+ KB
```

	count	mean	std	min	25%	50%	75%	max
work_year	3755.0	2022.373635	0.691448	2020.0	2022.0	2022.0	2023.0	2023.0
salary	3755.0	190695.571771	671676.500508	6000.0	100000.0	138000.0	180000.0	30400000.0
salary_in_usd	3755.0	137570.389880	63055.625278	5132.0	95000.0	135000.0	175000.0	450000.0
remote_ratio	3755.0	46.271638	48.589050	0.0	0.0	0.0	100.0	100.0

```
: df.nunique()
```

```
: work_year             4
  experience_level      4
  employment_type       4
  job_title            93
  salary              815
  salary_currency      20
  salary_in_usd      1035
  employee_residence   78
  remote_ratio          3
  company_location     72
  company_size          3
  dtype: int64
```

```
: df.isnull().sum()
```

```
: work_year             0
  experience_level      0
  employment_type       0
  job_title             0
  salary                0
  salary_currency       0
  salary_in_usd         0
  employee_residence    0
  remote_ratio          0
  company_location      0
  company_size          0
  dtype: int64
```

- برای پیش پردازش داده ها برای فهم بهتر در رسم نمودار ها مقدار برخی ستون های categorical تغییر یافت (این کار تاثیری روی کیفیت داده ها ندارد و صرفا برای فهم بهتر است).

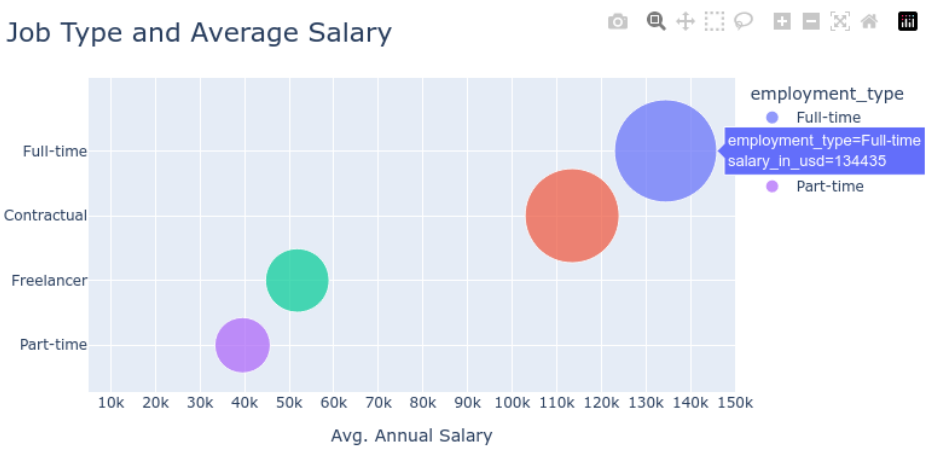
	work_year	experience_level	employment_type	job_title	salary	salary_currency	salary_in_usd	employee_residence	remote_ratio	company_location	co
1091	2023	Mid-level	Full-time	Deep Learning Engineer	150000	USD	150000	US	On-site	US	
1983	2022	Mid-level	Full-time	Data Analyst	102640	USD	102640	US	On-site	US	
826	2023	Senior	Full-time	Data Analyst	161500	USD	161500	US	On-site	US	
2464	2022	Senior	Full-time	Data Scientist	198440	USD	198440	US	Remote	US	
802	2023	Senior	Full-time	Data Analyst	138000	USD	138000	US	On-site	US	

- ردیف های تکراری در مجموعه داده حذف شده
- ستون stranger به صورت زیر به مجموعه داده اضافه شده

```
df.loc[(df['employee_residence']!=df['company_location']),'stranger']=1
df.loc[(df['employee_residence']==df['company_location']),'stranger']=0
```

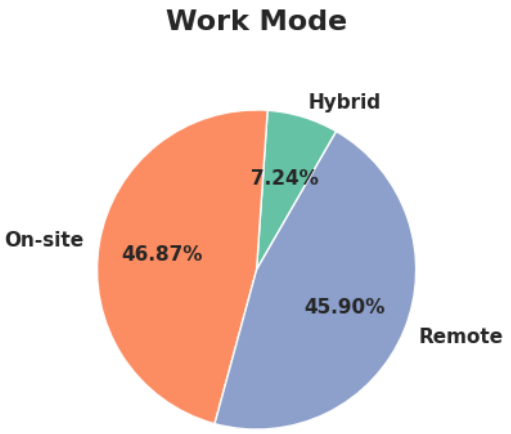
- ستون های ['salary_currency', 'salary'] از مجموعه داده حذف شده
- بعد از نمایش اطلاعات داده ها(b) ستون های categorical به مقادیر عددی برای آموزش مدل تغییر پیدا کردند.
- داده های پرت در salary_in_usd حذف شدند (c)

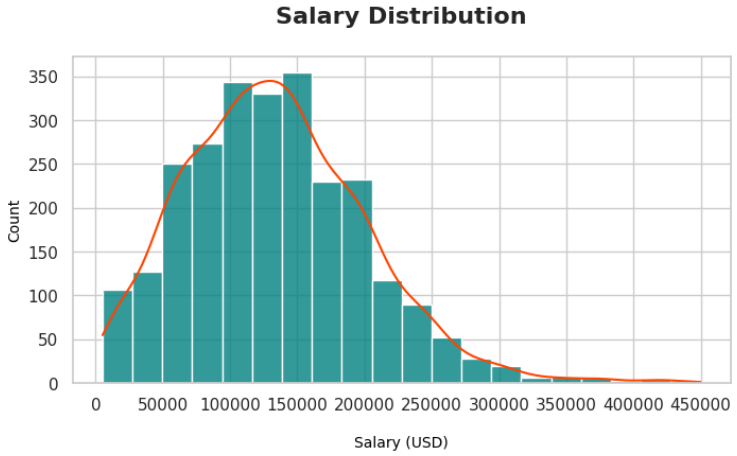
(b

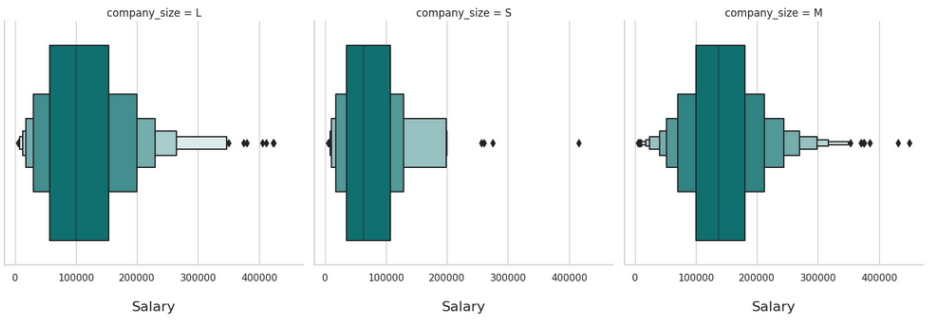


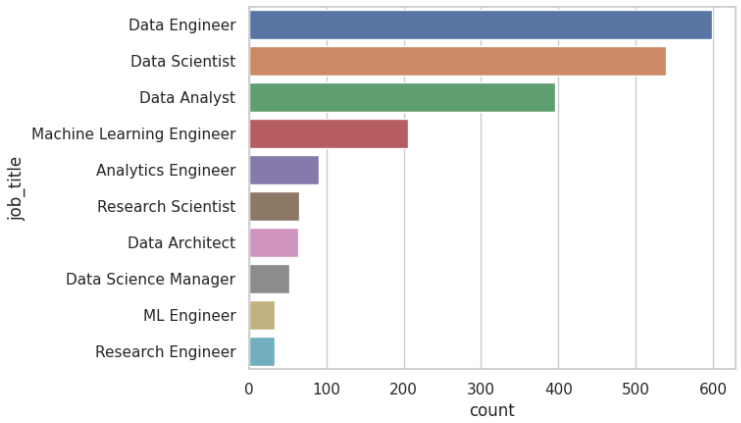
employment_type	salary_in_usd
Full-time	134435.0
Contractual	113447.0
Freelancer	51808.0
Part-time	39534.0

company_size	salary_in_usd
L	113202.24
M	141474.51
S	78364.28


Work Mode
Hybrid
7.24%
On-site
46.87%
45.90%
Remote


Salary Distribution
Count
Salary (USD)


company_size = L
company_size = S
company_size = M
Salary
Salary
Salary


Data Engineer
Data Scientist
Data Analyst
Machine Learning Engineer
Analytics Engineer
Research Scientist
Data Architect
Data Science Manager
ML Engineer
Research Engineer
job_title
count

(c

داده ها به دو دسته train , test تقسیم شدند

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

(d

ابتدا از مدل درخت به عمق ۲ و سپس از RandomForestClassifier بر روی برگ های درخت توسعه داده میشود.

```
tree = DecisionTreeClassifier(max_depth=2)
tree.fit(X_train, y_train)
```

```
DecisionTreeClassifier(max_depth=2)
```

```
y_pred_tree = tree.predict(X_test)
accuracy_tree = accuracy_score(y_test, y_pred_tree)
print("Decision Tree accuracy:", accuracy_tree)
```

```
Decision Tree accuracy: 0.030947775628626693
```

Extract leaf data

```
leaf_indices = tree.apply(X_train)  # Get the leaf indices
leaf_data = X_train[leaf_indices == tree.apply(X_train)]  # Extract leaf data
leaf_targets = y_train[leaf_indices == tree.apply(X_train)]  # Extract corresponding targets
```

```
]: rf = RandomForestClassifier()
   rf.fit(leaf_data, leaf_targets)
```

(e

```
Accuracy: 0.0038684719535783366
Precision: 0.0038684719535783366
Recall: 0.0038684719535783366
```

(f

ابتدا از درخت تصمیم با عمق ۲ استفاده میشود، سپس در داده های موجود در برگی های درخت RandomForest اجرا میشود، با انجام این کار دقت درخت تا حدودی افزایش میابد، برای توسعه بعد از تعریف کردن درخت با عمق ۲ داده آموزشی بر روی آن اجرا میشود و داده های موجود در هر برگ استخراج میشود و بر روی آن RandomForest اجرا میشود.

ابتدا SVM را به عنوان دومین مدل برای توسعه روی برگ های درخت انتخاب کرده بودم که با اعمال آن دقت نسبت به درخت با عمق ۲ حتی کاهش میافت.

```
Accuracy: 0.02321083172147002
Precision: 0.02321083172147002
Recall: 0.02321083172147002
F1 score: 0.023210831721470024
```

Q5

(a

هر خط در نقطه شروع و پایان شامل قیمت و زمان میباشد که استفاده از آن برای توسعه مدل k means مناسب نیست و باید از فیچر مناسب استفاده شود.

برای توسعه مدل روی این دیتا از قیمت شروع و پایان هر خط و خط های قبلی آن (برای بررسی وضعیت) استفاده میشود.

(b

```
num = len(zigzag_lines)
data=[]
for i in range(num):
    start_time , start_price = zigzag_lines[i][0]
    end_time , end_price = zigzag_lines[i][1]
    data.append([start_time,end_time,start_price,end_price,trends[i]])

data[:2]
```

داده در datasets ذخیره میشود که کلید آن تعداد leg را مشخص میکند، برای هر لگ قیمت شروع خط های قبل، نقطه پایان و trend آن خط ذخیره شده است.

```
datasets={}
previous = [2,3,4]
for p in previous:
    d1 = []
    for i in range(p , num):
        d2 = []
        for j in range (p-1,-1,-1):
            d2+=data[i-j][2:3]
        d2+=data[i][3:]
        d1.append(d2)
    datasets[p] = d1
```

```
datasets[3][:2]
```

```
[[1985.665, 1978.324, 2019.248, 2011.115, 'b'],
 [1978.324, 2019.248, 2011.115, 2019.465, 'b']]
```

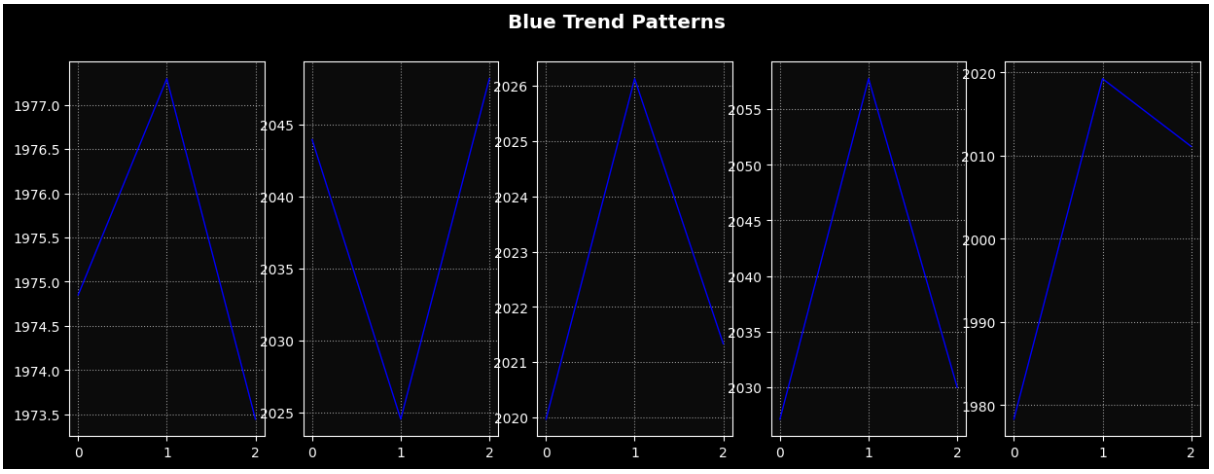
(c

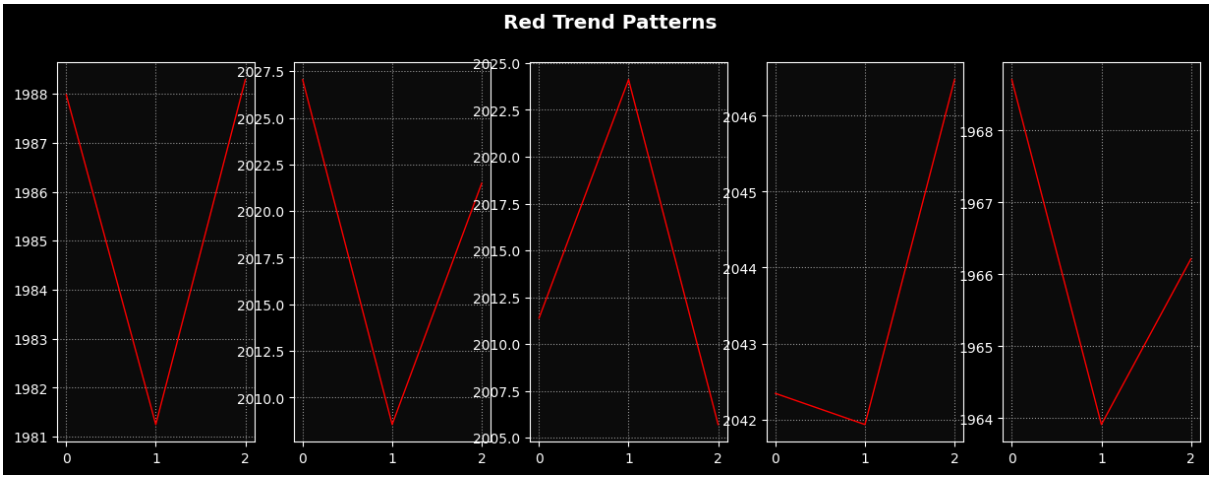
در این قسمت از تابع زیر استفاده شده:

```
]: def KMeans_imp(n,n2,cluster=5):
    data = np.array(datasets[n])

    b_trend_data = data[data[:, n2] == 'b']
    r_trend_data = data[data[:, n2] == 'r']

    r_X = r_trend_data[:, :n2]
    r_kmeans = KMeans(n_clusters=cluster)
    r_labels = r_kmeans.fit_predict(r_X)
    r_centers = r_kmeans.cluster_centers_

    b_X = b_trend_data[:, :n2]
    b_kmeans = KMeans(n_clusters=cluster)
    b_labels = b_kmeans.fit_predict(b_X)
    b_centers = b_kmeans.cluster_centers_


    fig, axs = plt.subplots(1, cluster, figsize=(15, 5
    fig.suptitle(f'Blue Trend Patterns')
    i=0
    for c in b_centers:
            axs[i].plot(c,color='b')
            i+=1
    plt.show()


    fig, axs = plt.subplots(1, cluster, figsize=(15, 5
    fig.suptitle(f'Red Trend Patterns')
    i=0
    for c in r_centers:
            axs[i].plot(c,color='r')
            i+=1
    plt.show()
```

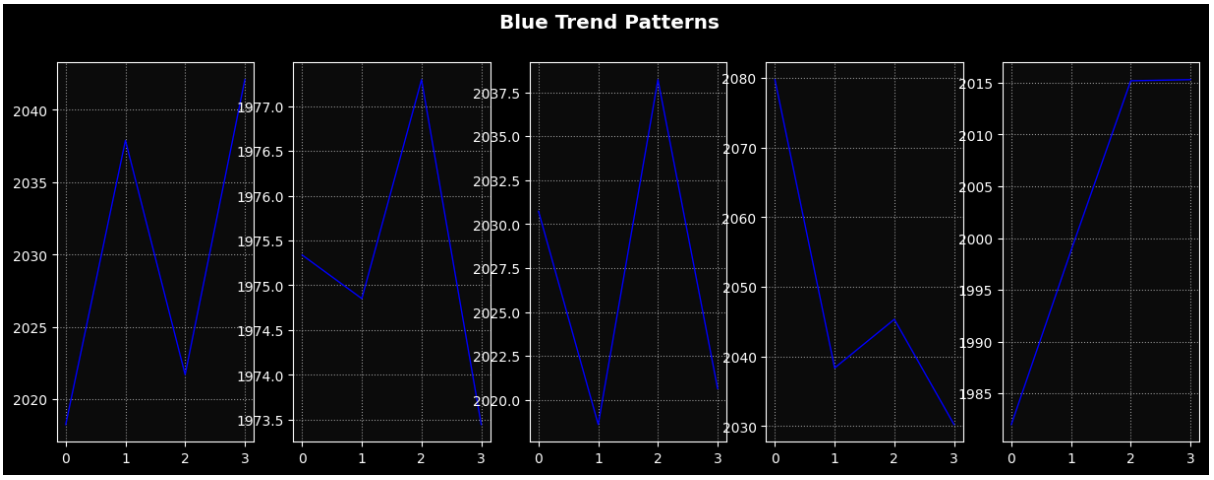
ورودی n=leg و n2 اندیس trend را مشخص میکند، داده ها به ازای trend های مختلف از یکدیگر جدا شده اند و روی هر قسمت الگوریتم k means اجرا شده و در نهایت مرکز خوشه ها رسم شده است.

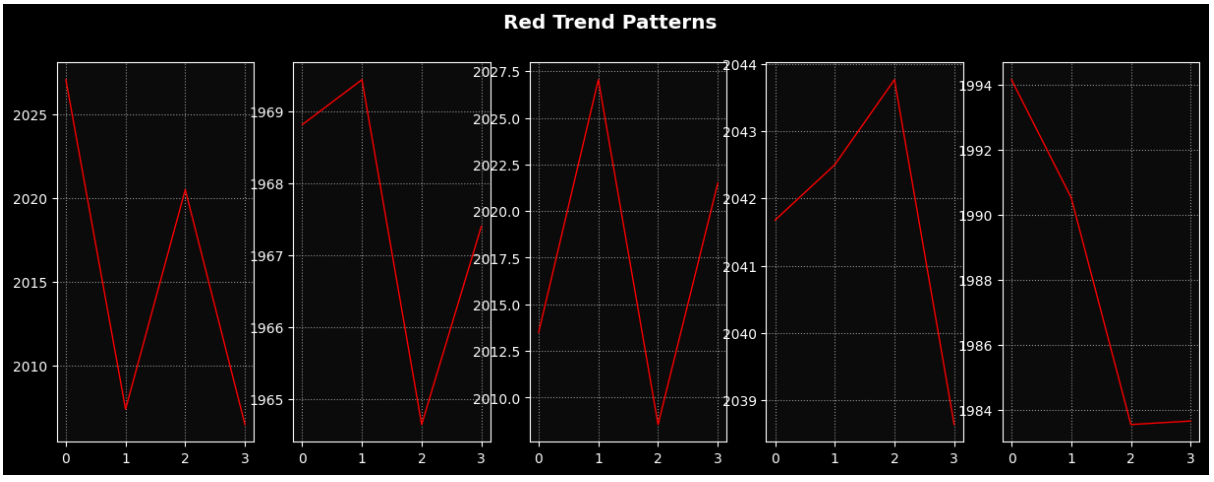
leg = 2



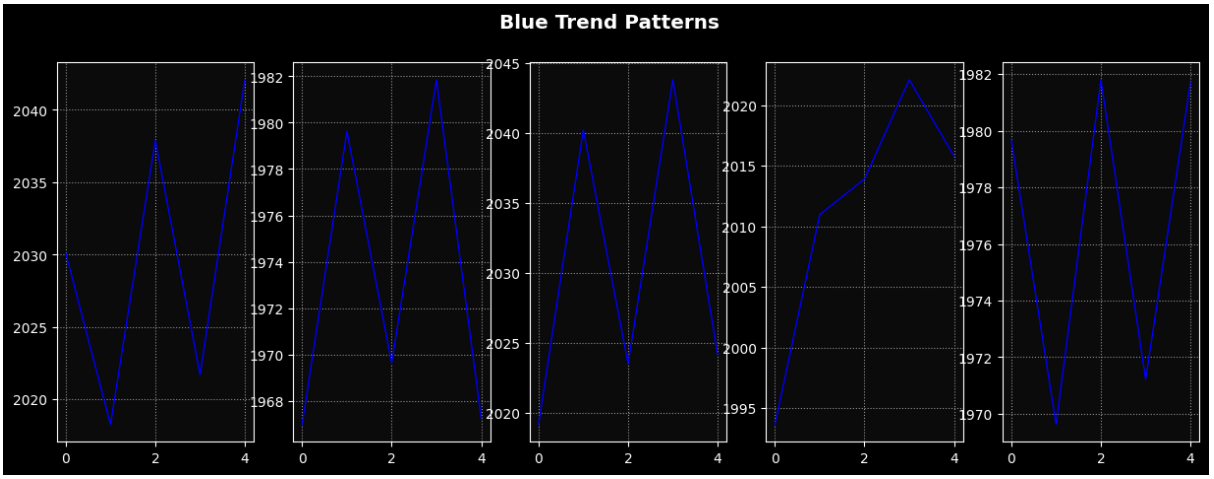


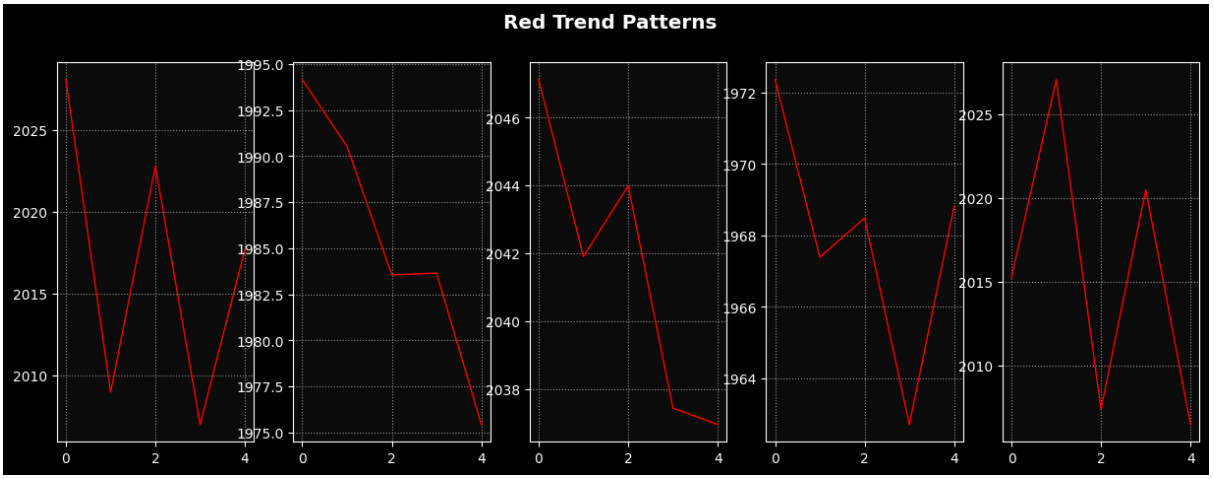
leg = 3





leg=4



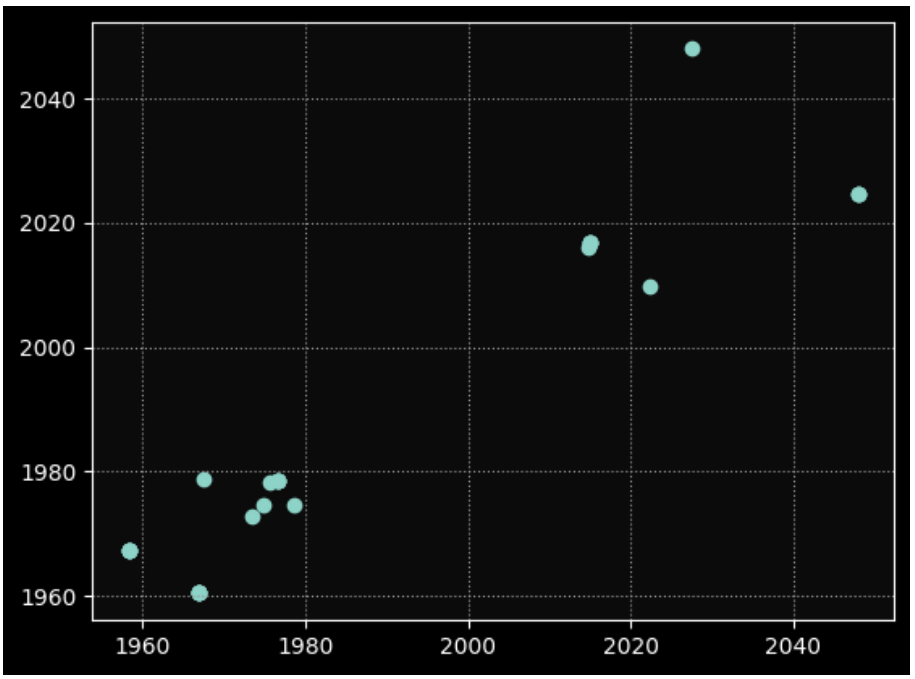


(d

برای نشان دادن حرکت مرکز خوشه ها از افزایش max-iter و ذخیره مقادیر مرکز خوشه ها استفاده شده است.

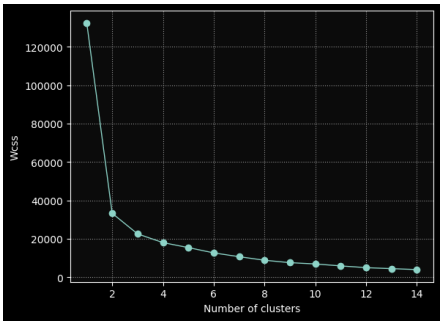
```
X = np.array(datasets[2])[:, 1:3]
centershist=[]
for i in range(1, 30):
    kmeans = KMeans(n_clusters=5, max_iter=i)
    labels = kmeans.fit_predict(X)
    centers = kmeans.cluster_centers_
    centershist.append(centers)
```

برای مثال، حرکت مرکز خوشه اول به صورت زیر میباشد:

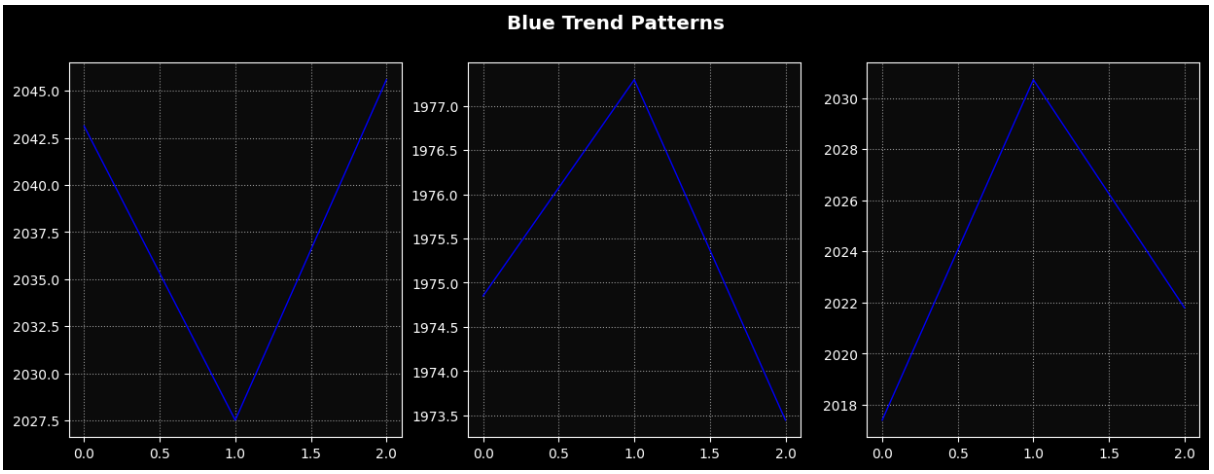


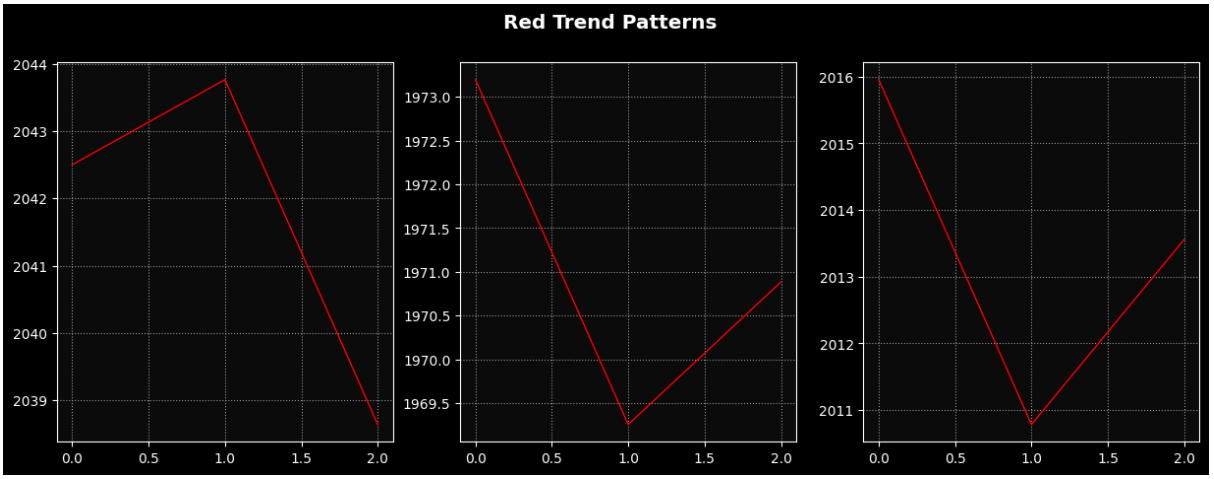
(e

leg = 2

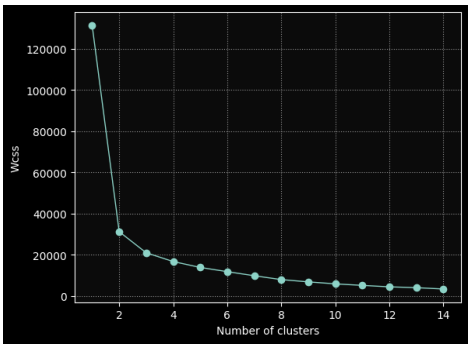


num_cluster = 3

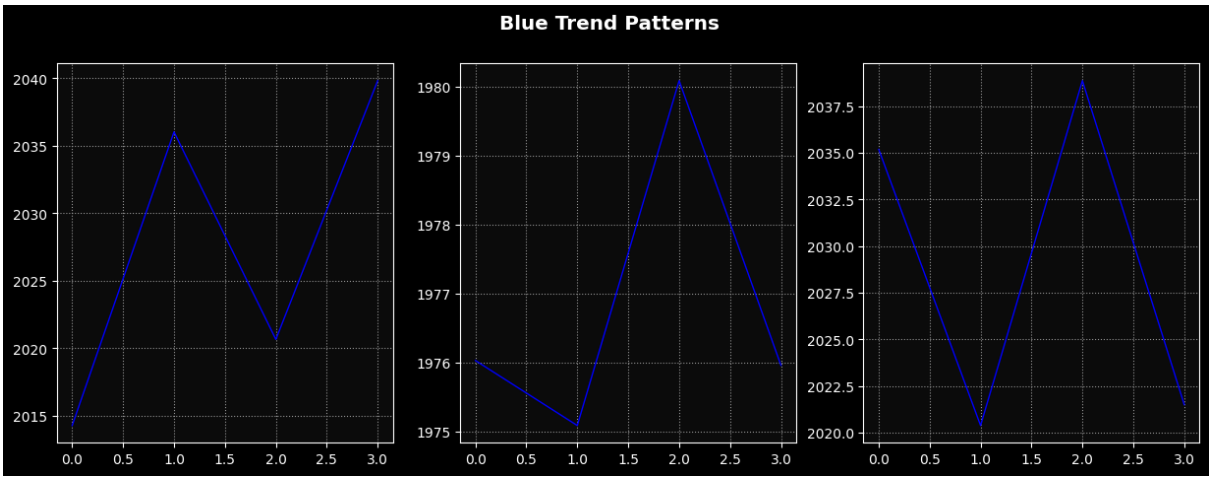


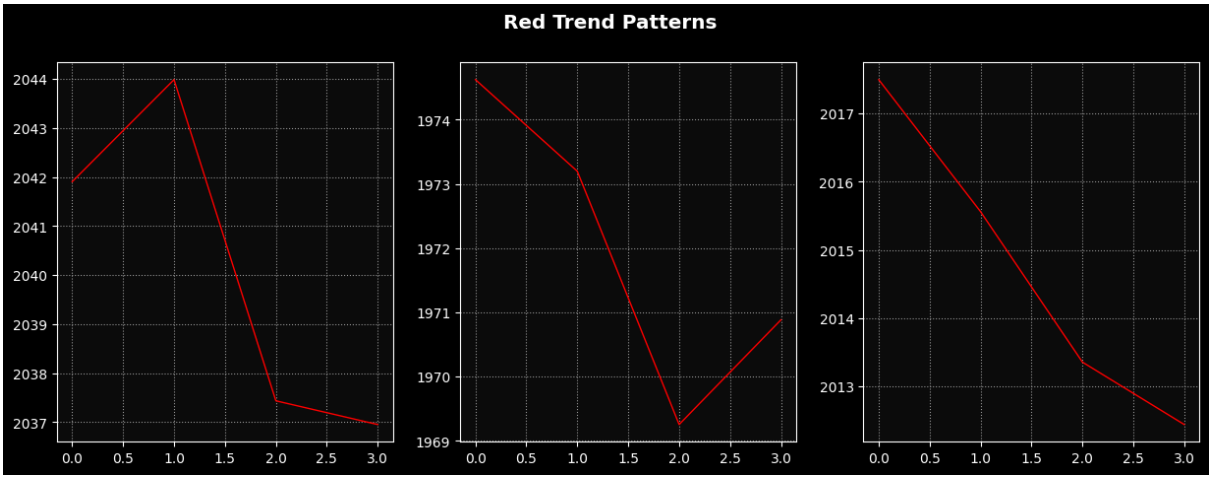


leg = 3

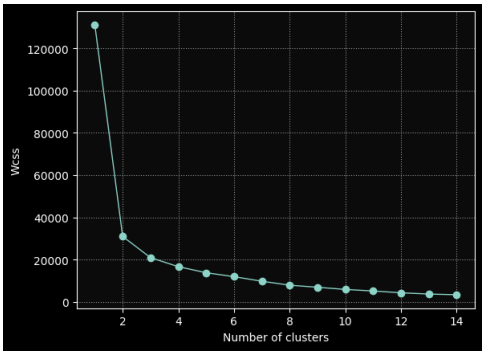


num_cluster = 3

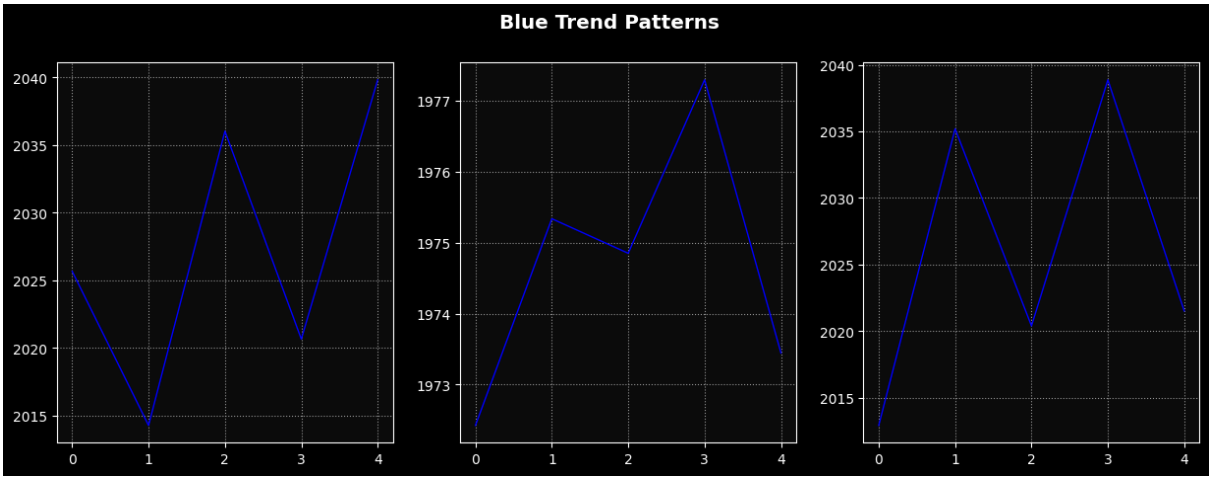


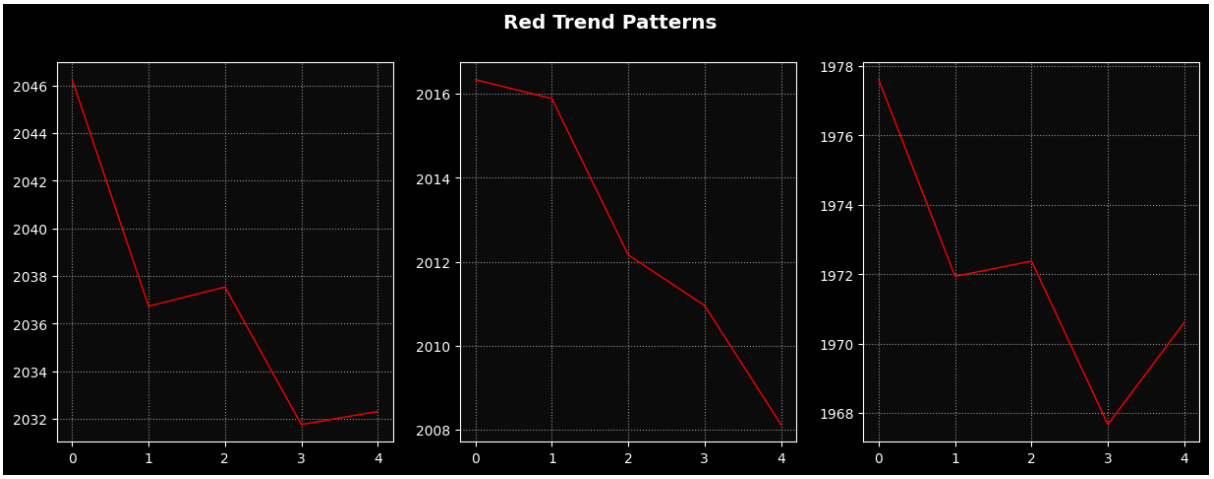


leg = 4



num_cluster = 3





(f

با تعیین n-cluster =3 در نمودار ها با اتصال نقاط min و ,max محلی به یکدیگر، در الگوهای قرمز رنگ (trend='r) داده ها به صورت نزولی و در آبی ها به صورت صعودی میباشد و در leg هایی با تعداد بیشتر این الگو بهتر مشخص است.

در حالی که هنگام استفاده از n-cluster = 5 گاهی اوقات خلاف این پیش بینی هم بود.

در leg=2 نمودار قرمز و آبی برعکس یکدیگرند اگر یکی صعودی باشد و دارای max دیگری نزولی و دارای min است، ولی در بقیه leg ها (۲و۳)در قرمز رنگ روند نزولی و در آبی صعودی میباشد.

Q6

(a

```
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler

X, y = make_blobs(n_samples=1000, centers=3, random_state=42)

scaler = StandardScaler()
X_std = scaler.fit_transform(X)
```

make_blobs برای ساختن دیتاست استفاده شده است و پارامتر های آن به صورت زیر تعریف شده‌اند:

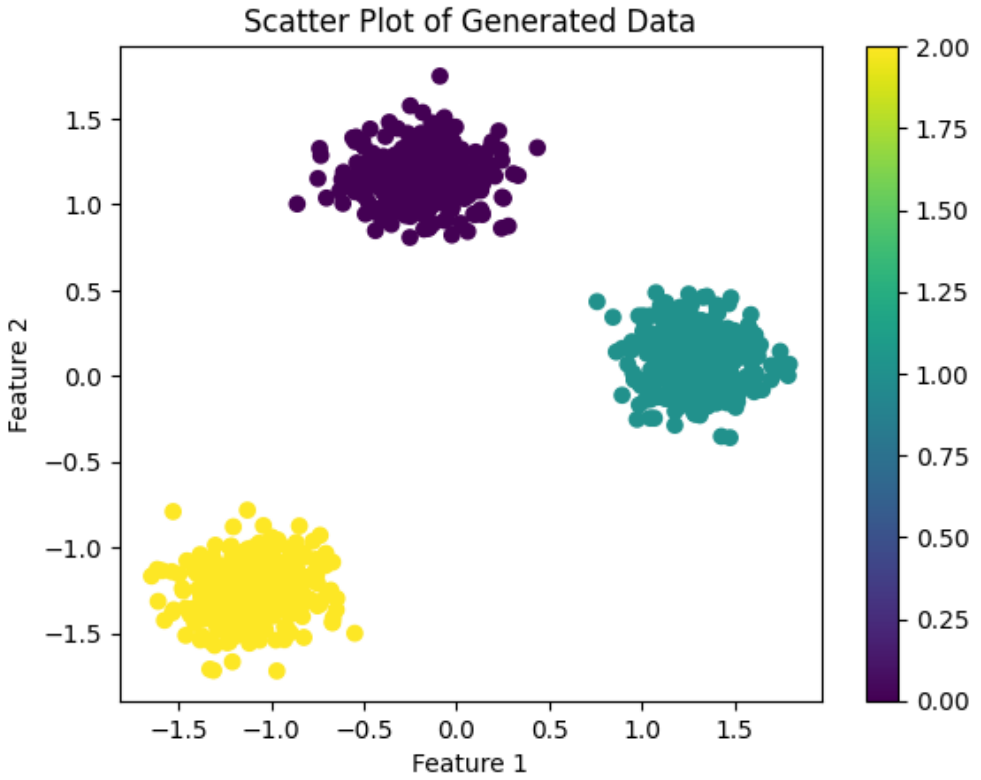
n_sample =1000 : تعداد کل نقاط است که به طور مساوی بین خوشه ها تقسیم می شود.

centers = 3 : تعداد مراکزی که باید تولید شوند.

از standardScaler برای استاندارد سازی داده ها استفاده می شود.

(b

از plt.scatter برای نمایش داده تولید شده استفاده می شود که به صورت زیر میباشد:



(c

از DBSCAN برای آموزش داده استاندارد شده استفاده میشود.

```
: from sklearn.cluster import DBSCAN

dbscan = DBSCAN(eps=0.1, min_samples=5)
dbscan.fit(X_std)
```

```
: DBSCAN
DBSCAN(eps=0.1)
```

eps:0.1 = حداکثر فاصله بین دو نمونه برای یکی در همسایگی دیگری در نظر گرفته شود.

(d

```
: import numpy as np

labels = dbscan.labels_
n_clusters = len(set(labels)) - (1 if -1 in labels else 0)

# noise points
n_noise = np.sum(labels == -1)

print("Number of clusters:", n_clusters)
print("Number of noise points:", n_noise)
```

```
Number of clusters: 3
Number of noise points: 26
```

برای دسترسی به تعداد خوشه ها و تعداد نقاط نویز از دستور بالا استفاده می شود که خروجی آن نمایش داده شده

(e

ارزیابی روی خوشه بندی انجام شده به صورت زیر میباشد:

```
Homogeneity Score: 0.9749633548119571
Completeness Score: 0.8996511259593948
V-measure Score: 0.9357944141824966
Adjusted Rand Index: 0.9605442073591193
Adjusted Mutual Information: 0.9356237779278681
Silhouette Coefficient: 0.7881006205107333
```