تمرین سری سوم

درس داده کاوی

جناب آقای دکتر فراهانی

تهیه کننده:

**پگاه احدیان**

**99422014**

تابستان 1400

1)

الگوریتم های SVM از مجموعه ای از توابع ریاضی که به عنوان کرنل تعریف می شوند، استفاده می کنند. وظیفه کرنل این است که داده ها را به عنوان ورودی گرفته و آن ها را به شکل مورد نیاز تبدیل کند. الگوریتم های مختلف SVM ، از انواع مختلف توابع کرنل استفاده می کنند. این توابع می توانند انواع متفاوتی داشته باشند. به عنوان مثال خطی ، غیرخطی ، چند جمله ای ، تابع پایه شعاعی (RBF) و سیگموئید.. اکنون قصد داریم شرح مفصلی از توابع کرنل SVM و کرنل های مختلف و نمونه هایی از جمله خطی ، غیرخطی ، چند جمله ای ، کرنل گاوسی ، تابع پایه شعاعی ( RBF ) ، سیگموئید و غیره را برای شما ارائه دهیم.

## توابع کرنل در SVM

توابع کرنل ، برای داده های ترتیبی ، نمودار ها ، متن ها ، تصاویر و همچنین بردار ها معرفی می شوند. پرکاربردترین نوع تابع کرنل، RBF است. زیرا دارای پاسخ محلی و متناهی در کل بازه محور x است.توابع کرنل ، ضرب داخلی بین دو نقطه در یک فضای ویژگی مناسب را برمی گردانند. بنابراین ، با هزینه محاسباتی کم، حتی در فضاهای با ابعاد بالا، مفهومی از شباهت را تعریف می کنند.

**کرنل های رایج مورد استفاده در SVM ها و کاربرد های آن ها را مشاهده کنیم :**

### ۱- کرنل چند جمله ای

این کرنل در پردازش تصویر پرکاربرد است. معادله آن به صورت زیر است :

$$k(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i} \cdot \mathbf{x_j} + 1)^d$$

که در آن d درجه چند جمله ای است.

### ۲- کرنل گاوسی

این یک کرنل برای اهداف عمومی است. و هنگامی که هیچ دانش پیشینی در مورد داده ها وجود ندارد استفاده می شود. معادله آن به صورت زیر است :

$$k(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right)$$

## 3- تابع پایه شعاعی گاوسی ( RBF )

این کرنلی برای اهداف عمومی کلربرد دارد. و هنگامی که هیچ دانش پیشینی در مورد داده ها وجود نداشته باشد، مورد استفاده قرار می گیرد. معادله آن به صورت زیر است :

فرمول کرنل RBF

$$k(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right)$$

$$\gamma > 0$$

## ۴- کرنل RBF لاپلاس

این هم یک کرنل برای اهداف عمومی است. و هنگامی که هیچ دانش پیشینی در مورد داده ها وجود ندارد استفاده می شود. معادله آن به صورت زیر است :

$$k(x, y) = \exp\left(-\frac{\|x - y\|}{\sigma}\right)$$

## ۵- کرنل تانژانت هیپربولیک ( tanh )

می توانیم از آن در شبکه های عصبی استفاده کنیم. معادله مربوط به آن عبارت است از :

$$k(\mathbf{x_i}, \mathbf{x_j}) = \tanh(\kappa \mathbf{x_i} \cdot \mathbf{x_j} + c)$$

## ۶- کرنل سیگموئید

می توان این کرنل را در شبکه های عصبی مورد استفاده قرار داد. معادله مربوط به آن عبارت است از :

$$k(x, y) = \tanh(\alpha x^T y + c)$$

## ۷- کرنل تابع بِسِل ( Bessel ) از نوع اول

ما می توانیم از آن برای حذف مقطع عرضی در توابع ریاضی استفاده کنیم. معادله آن عبارت است از :

$$k(x,y) = \frac{J_{v+1}(\sigma\|x-y\|)}{\|x-y\|^{-n(v+1)}}$$

که J تابع بسل از نوع اول است.

## ۸- کرنل پایه شعاعی ANOVA

ما می توانیم از آن در مسائل رگرسیون استفاده کنیم. معادله مربوط به آن عبارت است از :

$$k(x,y) = \sum_{k=1}^{n} \exp(-\sigma(x^k - y^k)^2)^d$$

## ۹- کرنل spline خطی بصورت یک بعدی

این کرنل، هنگام کار با بردارهای بزرگ داده پراکنده ، کاربرد زیادی دارد. این کرنل اغلب در دسته بندی متن مورد استفاده قرار می گیرد. کرنل spline همچنین در مسائل رگرسیون عملکرد خوبی دارد. معادله آن عبارت است از :

$$k(x,y) = 1 + xy + xy\ min(x,y) - \frac{x+y}{2}\ min(x,y)^2 + \frac{1}{3}\min(x,y)^3$$

2)طبق کد

3) شش حالت مختلف مدل SVMمتشکل از چهار کرنل مختلف خطی ،سیگموید، چندجمله ای و شعاعی با پارامترهای مختلف را انجام دادیم و مقدار خطای هر حالت را با استفاده از تابع محاسبه خطا Calculate_error بدست آوردیم.

4)طبق کد

5)

الف ) عملیات binningبروی ستون battery powerانجام گرفت و طبق کد.

ب) عملیات one hot encodingبرای ستون های categoricalاستفاده می شوند. مقادیر categoricalنوع داده ای هستند که تنها میتوانند مقادیر محدودی را در خود جای دهند ، برای مثال ممکن است برای ستون رنگی ، شامل رنگهای سبز ، ابی ، قرمز و ... باشد.

در صورت استفاده از این نوع داده ای بعضی از مدل های یادگیری ماشین امکان انجام پردازش مستقیم و بدون کدبندی بروی این نوع داده را ندارند ، یکی از روش های encodingروش one hot encodingاست.
batteryبروی ستون binningهست که از عملیات binning_bettery_powerما همان ستونی categoricalتنها ستون powerحاصل شده است و مابقی ستون ها از نوع intو floatهستند.

ج) گاهی اوقات داده های خامی که در اختیار داریم برای تحلیل هایی که در نظر داریم اماده نیستند و تخمین های ما را دچار مشکل میکنند ، ممکن است داده ها نرمال نبوده ، خطی نبوده و یا دارای پراکندگی یکسانی نبوده باشند ، در این جا ما از تبدیلاتی چون ریشه دوم، لگاریتم، وارون و مجذور کردن برای رفع مشکل نرمال نبودن ، خطی نبودن و یا توزیع پراکنده استفاده میکنیم. به طابتدا ما با ازمون های نرمال بودن می بایست نرمال بودن توزیع داده هامون رو تست کنیم ، بعد میتونیم از تبدیل های logو نمایی برای نرمال کردن توزیع داده هامون استفاده کنیم و البته با انجام مجدد ازمون های نرمال بودن ، می توانیم ببینیم تبدیلاتی که انجام دادیم مفید بوده است یا خیر ور کلی زمانی که داده ها نرمال نیستند از روش های ریشه دوم، لگاریتم و وارونه کردن برای تبدیل داده ها استفاده می شود.

6)طبق کد

7)

## درخت تصمیم چیست؟

درخت تصمیم(decision tree) یکی از پرکاربردترین الگوریتم‌ها در بین الگوریتم‌های داده‌کاوی است. درخت تصمیم دقیقا مانند یک درخت است با این تفاوت که از ریشه به سمت پایین (برگ) رشد کرده است. در الگوریتم درخت تصمیم نمونه‌ها را دسته‌بندی می‌کنیم که در واقع دسته‌ها در انتهای گره‌های برگ قرار دارد. درخت تصمیم در مسائلی کاربرد دارد که بتوان آنها را به صورتی مطرح نمود که پاسخ واحدی به صورت نام یک دسته یا کلاس ارائه دهند.

فرض کنید گروهی ورزشکار می‌خواهند برای بازی تنیس در هوای آزاد بروند، آن‌ها باید وضعیت هوا را برای رفتن به زمین تنیس درنظر بگیرند. این تصمیم‌گیری با استفاده از الگوریتم درخت تصمیم قابل انجام است.

## روش طراحی درخت تصمیم(decision tree)

روش های ساخت درخت تصمیم معمولا به صورت بالا به پایین عمل می کنند به این معنی که ابتدا فضای ورودی به فضاهای کوچکتر تقسیم می‌شود، سپس فرآیند تقسیم بندی برای هر یک از این قسمت‌ها تکرار می شود. به عبارت دیگر در هنگام ساخت درخت، ابتدا ریشه ساخته می‌شود، سپس هر یک از زیر شاخه ها به شاخه‌های دیگری تقسیم می‌شود و این فرآیند تکرار می شود.

## درخت تصمیم چگونه کار می‌کند؟

عملکرد درخت تصمیم به این صورت است که یک گره ریشه در بالای آن قرار دارد و برگ های آن در پایین می باشند. یک رکورد در گره ریشه وارد می‌شود و در این گره یک تست صورت می گیرد تا معلوم شود که این رکورد به کدام یک از گره های فرزند (شاخه پایین‌تر) خواهد رفت.

درخت تصمیم از تعدادی گره و شاخه تشکیل شده است که در آن نمونه ها را به نحوی طبقه بندی می‌کند که از ریشه به سمت پایین رشد می کند و در نهایت به گره های برگ م ی‌رسد. هر گره داخلی یا غیر برگ با یک ویژگی مشخص می شود. این ویژگی سوالی را در رابطه با مثال ورودی مطرح می کند. در هر گره داخلی به تعداد جواب های ممکن با این سوال شاخه وجود دارد که هریک با مقدار آن جواب مشخص می‌شوند. برگ‌های این درخت با یک کلاس و یا یک طبقه از جواب‌ها مشخص می‌شوند.

## الگوریتمهای برای ساخت درخت تصمیم:

ID3:Iterative Dichotomiser

C4.5: Classifier 4.5

CART: Classification And Regression Tree

ID4

ds CART: DempsterShafer  Classification And Regression Tree

ID5R

EC4.5:Efficient Classifier 4.5

CHAID: Chi square Automatic Interaction Detection

RF: Random Forest

RT: Random Tree

DS: Decision Stump

QUEST: Quick Unbiased  Efficient Statistical Tree

**الگوریتم ID3:**

این الگوریتم یکی از ساده ترین الگوریتم های درخت تصمیم(decision-tree) است. در این الگوریتم درخت تصمیم از بالا به پایین ساخته می‌شود. این الگوریتم با این سوال شروع می‌شود: کدام ویژگی باید در ریشه درخت مورد آزمایش، قرار بگیرد؟ برای یافتن جواب از معیار بهره اطلاعات  استفاده می‌شود.

با انتخاب این ویژگی، برای هر یک از مقادیر ممکن آن یک شاخه ایجاد شده و نمونه های آموزشی بر اساس ویژگی هر شاخه مرتب می‌شوند. سپس عملیات فوق برای نمونه های قرار گرفته در هر شاخه تکرار می شوند تا بهترین ویژگی برای گره بعدی انتخاب شود.

**الگوریتم CART:**

این الگوریتم متغیرهای ورودی را برای یافتن بهترین تجزیه می آزماید تا شاخص ناخالصی حاصل از تجزیه کمترین مقدار باشد.در تجزیه دو زیر گروه ایجاد میشوند و هر کدام در مرحله بعد به دو زیر گروه دیگر تقسیم میشوند و این روند ادامه دارد تا یکی از معیارهای توقف ارضا شود.درخت cart بازگشتی دودویی است.

8)طبق کد

## 9)

به طور کلی می توان این طور گفت که در آنالیز تصمیم، یک درخت تصمیم به عنوان ابزاری برای به تصویر کشیدن و آنالیز تصمیم، به ویژه در آن مواردی که مقادیر مورد انتظار از رقابت ها متناوبا محاسبه می شود، مورد استفاده قرار می گیرد. هر درخت تصمیم دارای سه نوع گره است:

گره تصمیم: به طور معمول با مربع نشان داده می شود.

گره تصادفی: معمولا با دایره مشخص می شود.

گره پایانی: معمولا با مثلث مشخص می شود.

هر درخت تصمیم می تواند به نحوی بسیار فشرده و در قالب یک دیاگرام، توجه ها را بر روی مسئله و رابطه بین رویداد های مختلف جلب کند. در هر درخت تصمیم مربع نشان دهنده تصمیم گیری، بیضی نشان دهنده فعالیت و لوزی نشان دهنده نتیجه است.

به عبارت دیگر، عملکرد درخت تصمیم به این صورت است که یک گره ریشه در بالای آن قرار دارد و برگ های آن در پایین می باشند. یک رکورد در گره ریشه وارد می شود و در این گره یک تست صورت می گیرد تا معلوم شود که این رکورد به کدام یک از گره های فرزند (شاخه پایین تر) خواهد رفت.

هر درخت تصمیم از تعدادی گره و شاخه تشکیل شده است که در آن نمونه ها را به نحوی طبقه بندی می کند که از ریشه به سمت پایین رشد می کند و در نهایت به گره های برگ می رسد. هر گره داخلی یا غیر برگ با یک ویژگی مشخص می شود. این ویژگی سوالی را در رابطه با مثال ورودی مطرح می کند. در هر گره داخلی به تعداد جواب های ممکن با این سوال شاخه وجود دارد که هر یک با مقدار آن جواب مشخص می شوند. برگ های این درخت با یک کلاس و یا یک طبقه از جواب ها مشخص می شوند.

سه معیار اصلی برای انتخاب ویژگی ها (مشخصه ها) برای انشعاب درخت تصمیم عبارتند از:

Info Gain

Gain ratio

Gini index

## 10)

**هرس کردن:**

به حذف کردن تعدادی از زیرگره های درخت تصمیم، هرس کردن می گویند. این عمل مشابه هرس کردن درخت ها و دقیقا بر خلاف عمل تقسیم کردن می باشد که بر تعداد زیرگره ها می افزود.

**پیش هرس (Pre pruning):**

که در آن یک درخت تصمیم به وسیله توقف های مکرر در مراحل اولیه ساخت درخت، هرس می شود. در واقع به محض ایجاد یک توقف گره به برگ تبدیل می شود.

**هرس پسین (Post pruning):**

که نسبت به رویکرد اول رایج تر است و در آن فرآیند به این صورت است که زیرگره های یک درخت کامل را حذف می کند. یک زیر گره در یک درخت با حذف کردن شاخه ها و جایگزینی آنها با یک برگ، هرس می شود.

12) نتیجه مقایسه رندوم فارست و درخت تصمیم نشان میدهد نتیجه رندوم فارست بهتر بوده است

13)

## مزایای درخت تصمیم:

مهم ترین فواید استفاده از این الگوریتم عبارتند از:

الف- فهم ساده: هر انسان با اندکی مطالعه و آموزش می تواند، طریقه کار با درخت تصمیم را بیاموزد.

ب- کار کردن با داده های بزرگ و پیچیده: درخت تصمیم در عین سادگی می تواند با داده های پیچیده به راحتی کار کند و از روی آنها تصمیم بسازد.

ج-استفاده مجدد آسان: در صورتی که درخت تصمیم برای یک مسئله ساخته شد، نمونه های مختلف از آن مسئله را می توان با آن درخت تصمیم محاسبه کرد.

د- قابلیت ترکیب با روش های دیگر: نتیجه درخت تصمیم را می توان با تکنیک های تصمیم سازی دیگر ترکیب کرده و نتایج بهتری بدست آورد.

14)

**C4.5**

با استفاده از پسوند الگوریتم ID3 کوینلان ، درخت تصمیم را از یک مجموعه تنظیم می کند. درخت تصمیم شامل شاخه هایی است که نتایج آزمایشات انجام شده بر روی ویژگی های انتخاب شده و گره های برگ را نشان می دهد ، که به برچسب های کلاس اختصاص داده شده است. C4.5 پس از ساختن درخت تصمیم ، سعی می کند با استفاده از یک مرحله هرس ، پیچیدگی آن را کاهش دهد و هدف آن حذف شاخه ها با حداقل کمک به

دقت کلی است. هنگامی که درخت هرس شد ، می توان دانش را استخراج و به صورت قوانین (در صورت وجود) ارائه داد.

Grow & prune algorithms

**Ripper**

(هرس افزایشی مکرر برای تولید خطا) یکی از کارآمدترین و مورد استفاده ترین الگوریتم های یادگیری قاعده است. این یک استراتژی تقسیم و تسخیر را برای حاکمیت استقرا اجرا می کند. Ripper برای تدوین یک مجموعه اولیه از قوانین برای هر کلاس ، اصطلاحاً به آن افزوده می شود (IREP). سپس ، مرحله بهینه سازی اضافی هر قانون را در مجموعه فعلی به نوبه خود در نظر می گیرد و دو قانون جایگزین از آنها ایجاد می کند: یک قانون جایگزینی و یک قانون تجدید نظر. پس از آن ، در مورد اینکه آیا مدل باید قاعده اصلی ، جایگزینی یا قانون تجدید نظر را بر اساس معیار حداقل طول توصیف حفظ کند ، تصمیم گیری می شود

**PART**

(نظریه تشدید تطبیقی تصوری) [4] یک الگوریتم درخت تصمیم جزئی است. به طور خاص ، PART با توجه به استراتژی تقسیم و تسخیر ، مجموعه ای از قوانین را ایجاد می کند ، همه موارد را از مجموعه آموزش که تحت این قانون هستند حذف می کند و به صورت بازگشتی ادامه می دهد تا زمانی که هیچ موردی باقی بماند. برای ایجاد یک قانون واحد ، PART یک درخت تصمیم C4.5 جزئی برای مجموعه موارد فعلی ایجاد می کند و برگ با بیشترین پوشش را به عنوان قانون جدید انتخاب می کند. پس از آن ، درخت تصمیم گیری جزئی به همراه موارد تحت پوشش قانون جدید از داده های آموزش حذف می شود تا از تعمیم اولیه جلوگیری شود. این روند تکرار می شود تا زمانی که همه موارد با قوانین استخراج شده پوشش داده شوند.

**CAMUR**

(طبقه بندی با مدل های مبتنی بر قانون جایگزین و MUltiple) [5،6] بر اساس الگوریتم RIPPER ساخته شده است. مبانی متعدد و معادل قاعده را از طریق محاسبه تکراری مدل طبقه بندی مبتنی بر قانون استخراج می کند. CAMUR شامل یک مخزن دانش موردی (پایگاه داده) و یک ابزار پرس و جو است.

**BIGBIOCL**

نسخه بهبود یافته CAMUR است که برای کنترل صدها هزار ویژگی طراحی شده است. طبق استراتژی CAMUR ، این برنامه برای یادگیری مدل های طبقه بندی چند گزینه ای و معادل از طریق حذف تکراری ویژگی های انتخاب شده طراحی شده است

Based on fuzzification

**FURIA**

نسخه بهبود یافته الگوریتم RIPPER است. FURIA از الگوریتم اصلاح شده RIPPER به عنوان مبنا استفاده می کند و قوانین فازی و مجموعه قوانین نامرتب را یاد می گیرد. نقطه قوت اصلی این الگوریتم روش کشش قانون است ، که با حل فشار مشکل رکوردهای جدیدی که طبقه بندی می شود می تواند خارج از فضای تحت پوشش قوانین قبلی باشد. نمایش قوانین فازی نیز پیشرفته است ، اساساً ، یک قانون فازی از طریق جایگزینی فواصل با فواصل فازی ، یعنی مجموعه های فازی با عملکرد عضویت ذوزنقه ای ، بدست می آید.

Based on probability estimation

**MLRules**

یک الگوریتم القایی برای حل مسائل طبقه بندی از طریق برآورد احتمال است. ایده اصلی این است که از قوانین واحد به عنوان طبقه بندی کننده منفرد استفاده کرده و سپس سیستم طبقه بندی گروه را بر روی آنها پیاده سازی کنید. متفاوت از رویه های کلاسیک پوشاندن متوالی (که به آن روشهای تقسیم و تسخیر) نیز گفته می شود ، قوانین جدید بدون تعدیل قوانینی که قبلاً اضافه شده اند ، اضافه می شوند. مزیت اصلی الگوریتم MLRules این واقعیت است که از یک روش آماری ساده و قدرتمند برای القای قوانین استفاده می شود ، که به نوبه خود منجر به گروه های نهایی با دقت پیش بینی بسیار بالا می شود.

Rank based

**TSP**

بر اساس مقادیر نسبی بین جفت ویژگی ها تکنیک القایی است. TSP برای داده های ریزآرایه و قوانین ایجاد شده در یک فضای ویژگی که با مقایسه دو به دو سطح بیان ژن تشکیل شده است ، توسعه نیافته است. مزیت اصلی رویکرد TSP این است که مبتنی بر مقادیر نسبی ، مشکل ادغام داده ها از منابع مختلف است که به طور بالقوه در مقیاس های مختلف نشان داده می شود و می تواند از اثرات دسته ای رنج ببرد. علاوه بر این ، طبقه بندی TSP قوانین تصمیم گیری را ارائه می دهد که تفسیر آن آسان است از آنجا که آنها مقادیر نسبی بین جفت ویژگی ها را شامل می شوند (ژن ها در مورد آن) .TSP با زبان R پیاده سازی می شود و از بسته Bioconductor tspair در دسترس است.

**k-TSP**

الگوریتم TSP است که دقیقاً از k جفت ژن برای طبقه بندی داده های بیان ژن استفاده می کند. به جای استفاده از یک التزال مقایسه ای واحد ، K-TSP از گروه های مقایسه k استفاده می کند و رای گیری اکثریت را برای تصمیم گیری در مورد صحت مجموعه اعمال می کند تحت اللفظی. وقتی k = 1 ، الگوریتم معادل الگوریتم TSP است.

(16

از طریق لینک به دیتاست رسیده و روی تاریخ ذکر شده فیلتر شده و خروجی گرفته شد.

از طریق کد زیر تاریخ را به فرمت ذکر شده تبدیل شد:

```
df['Date']= df['Date'].apply(lambda x: datetime.strptime(x, '%b %d, %Y').strftime('%Y-%m-%d'))
```

خروجی:

| | Date | Price | Open | High | Low | Vol. | Change % |
|---|---|---|---|---|---|---|---|
| **0** | 2021-05-01 | 57,807.1 | 57,719.1 | 58,449.4 | 57,029.5 | 63.41K | 0.15% |
| **1** | 2021-04-30 | 57,720.3 | 53,562.3 | 57,925.6 | 53,088.7 | 103.74K | 7.77% |
| **2** | 2021-04-29 | 53,560.8 | 54,838.6 | 55,173.7 | 52,400.0 | 83.90K | -2.34% |
| **3** | 2021-04-28 | 54,841.4 | 55,036.0 | 56,419.9 | 53,876.4 | 86.96K | -0.35% |
| **4** | 2021-04-27 | 55,036.5 | 54,011.1 | 55,427.8 | 53,345.0 | 84.08K | 1.88% |

با توجه به اینکه دیتاها به صورت object هستن برای ادامه نیاز داریم به float یا integer تبدیل شوند و حرف k از کنار فیچر Vol. حذف شد:

```
data = data.apply(lambda x: x.str.replace(',', ''))

data['Vol.'] = data['Vol.'].str.replace(r'\D','')
data
```

| | Date | Price | Open | High | Low | Vol. | Change % |
|---|---|---|---|---|---|---|---|
| **0** | 2021-05-01 | 57807.1 | 57719.1 | 58449.4 | 57029.5 | 6341 | 0.15% |
| **1** | 2021-04-30 | 57720.3 | 53562.3 | 57925.6 | 53088.7 | 10374 | 7.77% |
| **2** | 2021-04-29 | 53560.8 | 54838.6 | 55173.7 | 52400.0 | 8390 | -2.34% |
| **3** | 2021-04-28 | 54841.4 | 55036.0 | 56419.9 | 53876.4 | 8696 | -0.35% |
| **4** | 2021-04-27 | 55036.5 | 54011.1 | 55427.8 | 53345.0 | 8408 | 1.88% |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **3936** | 2010-07-22 | 0.1 | 0.1 | 0.1 | 0.1 | 216 | 0.00% |
| **3937** | 2010-07-21 | 0.1 | 0.1 | 0.1 | 0.1 | 058 | 0.00% |
| **3938** | 2010-07-20 | 0.1 | 0.1 | 0.1 | 0.1 | 026 | 0.00% |
| **3939** | 2010-07-19 | 0.1 | 0.1 | 0.1 | 0.1 | 057 | 0.00% |
| **3940** | 2010-07-18 | 0.1 | 0.0 | 0.1 | 0.1 | 008 | 0.00% |

3941 rows × 7 columns

حال دیتاهای NON را با 0 جایگزین میکنیم:

```
data = data.replace(np.nan, 0)
```

حال از روش های یادگیری ماشین بهره میبریم:

```
X = data.drop(['Price', 'Date', 'Change %'], axis = 1)

y = data['Price']

X_test, X_train, y_test, y_train = train_test_split(X,y, test_size = 0.9699 ,random_state = 42, shuffle = False)

X_train
```

(17

در اینجا از روش هایی که استفاده شده است لیستی مشاهده میکنید:

- رگرسیون خطی

- رگرسیون لاسو
- رگرسیون ریج
- روش SVR با کرنل RB
- روش SVR
- شبکه عصبی LSTM

```
lin_regression = LinearRegression().fit(X_train, y_train)

lin_pred = lin_regression.predict(X_test)
lin_pred_score = math.sqrt(mean_squared_error(y_test, lin_pred))
print('test score : %.2f RMSE' % (lin_pred_score))
test score : 961.10 RMSE


ridge = Ridge().fit(X_train, y_train)

ridge_pred = ridge.predict(X_test)
ridge_pred_score = math.sqrt(mean_squared_error(y_test, ridge_pred))
print('test score : %.2f RMSE' % (ridge_pred_score))
test score : 961.10 RMSE


lasso = Lasso().fit(X_train, y_train)

lasso_pred = lasso.predict(X_test)
lasso_pred_score = math.sqrt(mean_squared_error(y_test, lasso_pred))
print('test score : %.2f RMSE' % (lasso_pred_score))
test score : 994.15 RMSE
svr_poly = SVR(kernel='poly').fit(X_train,y_train)

svr_poly_pred = svr_poly.predict(X_test)
svr_poly_pred_score = math.sqrt(mean_squared_error(y_test, svr_poly_pred))
print('test score : %.2f RMSE' % (svr_poly_pred_score))
test score : 19534.72 RMSE
svr_rbf = SVR(kernel='rbf', gamma=0.1).fit(X_train, y_train)

svr_rbf_pred = svr_rbf.predict(X_test)
svr_rbf_pred_score = math.sqrt(mean_squared_error(y_test, svr_rbf_pred))
print('test score : %.2f RMSE' % (svr_rbf_pred_score))
test score : 49181.85 RMSE
LSTM

dataset = data.values
```

```
dataset = np.delete(dataset,0,1)
dataset = np.delete(dataset,5,1)
dataset.astype('float32')


train_size = int(len(dataset) * 0.9699)

test_size = len(dataset) - train_size
test, train = dataset[0:test_size,0:1], dataset[test_size:len(dataset),1:2]
print(len(train), len(test))
print(train.shape[1], test.shape[1])
```

output

```
119 3822
1 1
```

آرایه ی جدا شده ی trainدارای 3822سطر و یک ستون و آرایه ی جدا شده ی testنیز دارای 119سطر و یک ستون است.

با استفاده از تابع create datasetاین آرایه ها را به ماتریس تبدیل می کنیم.

```
:def create_dataset(dataset, look_back=1)

dataX, dataY = [], []
for i in range(len(dataset)-look_back):
a = dataset[i:(i+look_back), 0]
dataX.append(a)
dataY.append(dataset[i + look_back, 0])
return np.array(dataX), np.array(dataY)


look_back = 1

trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
trainY.shape


trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))

testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))


tensor_trainX = tf.convert_to_tensor(trainX, np.float32)

tensor_testX = tf.convert_to_tensor(testX, np.float32)
```

تابع خطای شبکه عصبی را هم به ترتیب mean square errorو بهینه ساز adamدر نظر می گیریم.

```
()model = Sequential

model.add(LSTM(4, input_shape=(1, look_back)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(tensor_trainX, trainY, epochs=100, batch_size=1, verbose=2)
```

RMSE :

```
trainPredict = model.predict(tensor_trainX)

testPredict = model.predict(tensor_testX)


trainPredict = model.predict(tensor_trainX)

testPredict = model.predict(tensor_testX)

trainScore = math.sqrt(mean_squared_error(trainY, trainPredict[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY, testPredict[:,0]))
print('Test Score: %.2f RMSE' % (testScore))
trainY.shape
```

```
Train Score: 5345.90 RMSE
Test Score: 49464.87 RMSE
```

## (18

تکنیک های Bagging , Boostingو votingاست و سه مثال برای ارائه داریم :

**Voting** :

این روش بر اساس voteهایی که از مدل های تکین می گیرد خروجی نهایی را می سازد. برای پیاده سازی این روش از کتابخانه رگرسیون بر اساس Voting Regressorاستفاده می کنیم.

داریم :

```
()model_1 = LinearRegression

()model_2 = Lasso
```

```
()model_3 = Ridge
)final_model = VotingRegressor
,'estimators=[('lr', model_1), ('ridge', model_2), ('lasso
([(model_3
()lab_enc = preprocessing.LabelEncoder
encoded_y_train = lab_enc.fit_transform(y_train)
final_model.fit(X_train, encoded_y_train)
pred_final = final_model.predict(X_test)
```

**:Bagging**

این روش بخش هایی از دیتاست را با طول های مشخص و مساوی انتخاب می کند و یک مدل یادگیری ماشین را روی این بخش های بدست آمده پیاده می کند.

```
,final_bag_model = BaggingRegressor(base_estimator=SVR(kernel='poly')
(n_estimators=10, random_state=0
()lab_enc = preprocessing.LabelEncoder
encoded_y_train = lab_enc.fit_transform(y_train)
final_bag_model.fit(X_train, encoded_y_train)
pred_final_bag = final_bag_model.predict(X_test)
```

**: Boosting**

مدل چندین و چند بار به صورت متوالی ( سری ) مورد استفاده قرار می گیرد؛ به طوری که خروجی یک مرحله به عنوان ورودی مرحله بعدی به همان مدل داده می شود.

پیاده سازی :

```
,()final_Boost_model = AdaBoostRegressor(base_estimator=SVR
(n_estimators=10, random_state=0
()lab_enc = preprocessing.LabelEncoder
encoded_y_train = lab_enc.fit_transform(y_train)
final_Boost_model.fit(X_train, encoded_y_train)
pred_final_boost = final_Boost_model.predict(X_test)
```

(19

در این سوال از روش AdaBoostاستفاده شده است. و خروجی های متعددی با پارامترهای مختلف از این روش گرفته شد:

**حالت اول: استفاده از مدل SVRبا کرنل rbfو گامای یک صدم**

```
final_Boost_model1=
,AdaBoostRegressor(base_estimator=SVR(kernel='rbf', gamma= 0.01)
(n_estimators=10, random_state=0
()lab_enc = preprocessing.LabelEncoder
encoded_y_train = lab_enc.fit_transform(y_train)
final_Boost_model1.fit(X_train, encoded_y_train)
pred_final_boost1 = final_Boost_model1.predict(X_test)
,pred_final_boost1_score = math.sqrt(mean_squared_error(y_test
((pred_final_boost1
print('test score : %.2f RMSE' % (pred_final_boost1_score))
```

test score : 48611.42 RMSE

**حالت دوم : مدل کرنل چند جمله ای درجه2 :**

```
final_Boost_model2=
,AdaBoostRegressor(base_estimator=SVR(kernel='poly', degree = 2)
(n_estimators=10, random_state=0
()lab_enc = preprocessing.LabelEncoder
encoded_y_train = lab_enc.fit_transform(y_train)
final_Boost_model2.fit(X_train, encoded_y_train)
pred_final_boost2 = final_Boost_model2.predict(X_test)
,pred_final_boost2_score = math.sqrt(mean_squared_error(y_test
((pred_final_boost2
```

```
print('test score : %.2f RMSE' % (pred_final_boost2_score))
```

test score : 39445.92 RMSE

**حالت سوم : دنباله ای به طول 20:**

```
final_Boost_model3=
,AdaBoostRegressor(base_estimator=SVR(kernel='rbf', gamma= 0.01)
(n_estimators=20, random_state=0
()lab_enc = preprocessing.LabelEncoder
encoded_y_train = lab_enc.fit_transform(y_train)
final_Boost_model3.fit(X_train, encoded_y_train)
pred_final_boost3 = final_Boost_model3.predict(X_test)
,pred_final_boost3_score = math.sqrt(mean_squared_error(y_test
((pred_final_boost3
print('test score : %.2f RMSE' % (pred_final_boost1_score))
```

test score : 48611.42 RMSE

**حالت چهارم :**

```
final_Boost_model4=
,AdaBoostRegressor(base_estimator=SVR(kernel='poly', degree = 2)
(n_estimators=20, random_state=0
()lab_enc = preprocessing.LabelEncoder
encoded_y_train = lab_enc.fit_transform(y_train)
final_Boost_model4.fit(X_train, encoded_y_train)
pred_final_boost4 = final_Boost_model2.predict(X_test)
,pred_final_boost4_score = math.sqrt(mean_squared_error(y_test
((pred_final_boost4
print('test score : %.2f RMSE' % (pred_final_boost4_score))
```

test score : 39445.92 RMSE

(20

الگوریتم Random Forestبه عنوان یکی دیگر از روش های یادگیری ترکیبی است. این روش کاملا شبیه روش Baggingاست؛ تنها تفاوت موجود در این روش ، استفاده از درخت های تصمیم به عنوان مدل

های یادگیرنده ضعیف است. درخت های تصمیم در مسائل رگرسیون، پس ازجداسازی داده ی آموزش، برای پیش بینی روی داده های تست نقاط داده ی تست هرکدام پس از دیگری در درخت تصمیم قرار می گیرند وپس از گذشتن از شروط نود های درخت به یکی از نود های برگ می رسند.سپس مقدار yتخمینی با میانگین گرفتن از مقادیر yنقاط موجود در آن برگبه دست می آید.

کد:

```
random_forest_model=
RandomForestRegressor(max_depth=2,random_state=0)
encode our target#
()lab_enc = preprocessing.LabelEncoder
encoded_y_train = lab_enc.fit_transform(y_train)
random_forest_model.fit(X_train,encoded_y_train)
random_forest_model_pred = random_forest_model.predict(X_test)
,random_forest_model_pred_score = math.sqrt(mean_squared_error(y_test
((random_forest_model_pred
print('test score : %.2f RMSE' % (random_forest_model_pred_score))
test score : 47243.62 RMSE
= random_forest_model4
RandomForestRegressor(max_depth=4,random_state=0)
encode our target#
()lab_enc = preprocessing.LabelEncoder
encoded_y_train = lab_enc.fit_transform(y_train)
random_forest_model4.fit(X_train,encoded_y_train)
random_forest_model4_pred = random_forest_model4.predict(X_test)
,random_forest_model4_pred_score = math.sqrt(mean_squared_error(y_test
((random_forest_model4_pred
print('test score : %.2f RMSE' % (random_forest_model4_pred_score))
```

test score : 46942.08 RMSE

## (21

فرمول خطای درصدی RMSEرا با استفاده از توابع کتابخانه numpyحساب می کنیم. سپس شرط گفته شده را بررسی می کنیم که اگر این مقدار حساب شده که در واقع امتیاز ( ) scoreمقدار تخمینی است، بیش از 0.95بود، یعنی خطای ما کمتر از 5صدم است و درست میگیریم.

```
def ignore_5_percent_error (y_true, y_predict):
,rmspe = np.sqrt(np.mean(np.square(((y_true - y_predict) / y_true))
((axis = 0
: if rmspe > 0.95
print('RMSPE value and Prediction status are ' + str(rmspe) + ' and
('True
: else
print('RMSPE value and Prediction status are ' + str(rmspe) + ' and
('False
return
ignore_5_percent_error(y_test, random_forest_model4_pred)
RMSPE value and Prediction status are 0.9406361017366941 and False
```

## (22

یک ستون به دیتاست طوری اضافه می کنیم که اگر مقدار قیمت اپن از مقدار قبلیش بیشتر بود مقدار ستون جدید برابر با یک و اگر کمتر باشد مقدار منفی یک می گیرد.سپس شبکه عصبی LSTMرا این بار با مقادیر تست و آموزش جدید پیاده سازی می کنیم.

```
x22 = data22['Open'].values
y22 = data22['Open-diff'].values
= xtest, xtrain, ytest, ytrain = train_test_split(x22, y22, test_size
(random_state = 42, shuffle=False ,0.9699
ytrain
array([-1.0, -1.0, -1.0, ..., 0.0, -1.0, 0], dtype=object)
```

```
xtrain = np.reshape(xtrain, (xtrain.shape[0], 1))
xtest = np.reshape(xtest, (xtest.shape[0], 1))
ytrain = np.reshape(ytrain, (y train.shape[0], 1))
ytrain.shape
(1 ,3823)
xtrain = np.reshape(xtrain, (xtrain.shape[0], 1, xtrain.shape[1]))
xtest = np.reshape(xtest, (xtest.shape[0], 1, xtest.shape[1]))
ytrain = np.reshape(ytrain, (ytrain.shape[0], 1, ytrain.shape[1]))
Ytrain.shape
TensorShape([3823, 1, 1])
()model = Sequential
model.add(LSTM(4, input_shape=(1, look_back)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
,model.fit(tensor_xtrain, tensor_ytrain, epochs=100, batch_size=1
(verbose=2
Epoch 1/100
6s - loss: 0.8916 - 3823/3823
Epoch 2/100
4s - loss: 0.8904 - 3823/3823
Epoch 3/100
4s - loss: 0.8904 - 3823/3823
Epoch 4/100
4s - loss: 0.8904 - 3823/3823
Epoch 5/100
4s - loss: 0.8905 - 3823/3823
Epoch 6/100
4s - loss: 0.8905 - 3823/3823
Epoch 7/100
4s - loss: 0.8901 - 3823/3823
```

Epoch 8/100

4s - loss: 0.8904 - 3823/3823

Epoch 9/100

4s - loss: 0.8902 - 3823/3823

Epoch 10/100

4s - loss: 0.8904 - 3823/3823

Epoch 11/100

4s - loss: 0.8900 - 3823/3823

Epoch 12/100

4s - loss: 0.8896 - 3823/3823

Epoch 13/100

4s - loss: 0.8904 - 3823/3823

Epoch 14/100

4s - loss: 0.8900 - 3823/3823

Epoch 15/100

4s - loss: 0.8904 - 3823/3823

Epoch 16/100

4s - loss: 0.8903 - 3823/3823

Epoch 17/100

4s - loss: 0.8901 - 3823/3823

Epoch 18/100

4s - loss: 0.8898 - 3823/3823

Epoch 19/100

4s - loss: 0.8901 - 3823/3823

Epoch 20/100

4s - loss: 0.8903 - 3823/3823

Epoch 21/100

4s - loss: 0.8897 - 3823/3823

Epoch 22/100

4s - loss: 0.8897 - 3823/3823

Epoch 23/100

4s - loss: 0.8903 - 3823/3823

Epoch 24/100

4s - loss: 0.8902 - 3823/3823

Epoch 25/100

4s - loss: 0.8899 - 3823/3823

Epoch 26/100

4s - loss: 0.8903 - 3823/3823

Epoch 27/100

4s - loss: 0.8902 - 3823/3823

Epoch 28/100

4s - loss: 0.8901 - 3823/3823

Epoch 29/100

4s - loss: 0.8902 - 3823/3823

Epoch 30/100

4s - loss: 0.8903 - 3823/3823

Epoch 31/100

4s - loss: 0.8903 - 3823/3823

Epoch 32/100

4s - loss: 0.8903 - 3823/3823

Epoch 33/100

4s - loss: 0.8900 - 3823/3823

Epoch 34/100

4s - loss: 0.8900 - 3823/3823

Epoch 35/100

4s - loss: 0.8902 - 3823/3823

Epoch 36/100

4s - loss: 0.8901 - 3823/3823

Epoch 37/100

4s - loss: 0.8902 - 3823/3823

Epoch 38/100

4s - loss: 0.8903 - 3823/3823

Epoch 39/100

4s - loss: 0.8901 - 3823/3823

Epoch 40/100

4s - loss: 0.8902 - 3823/3823

Epoch 41/100

4s - loss: 0.8900 - 3823/3823

Epoch 42/100

4s - loss: 0.8904 - 3823/3823

Epoch 43/100

4s - loss: 0.8902 - 3823/3823

Epoch 44/100

4s - loss: 0.8902 - 3823/3823

Epoch 45/100

4s - loss: 0.8896 - 3823/3823

Epoch 46/100

4s - loss: 0.8903 - 3823/3823

Epoch 47/100

4s - loss: 0.8901 - 3823/3823

Epoch 48/100

4s - loss: 0.8899 - 3823/3823

Epoch 49/100

4s - loss: 0.8898 - 3823/3823

Epoch 50/100

4s - loss: 0.8903 - 3823/3823

Epoch 51/100

4s - loss: 0.8899 - 3823/3823

Epoch 52/100

4s - loss: 0.8904 - 3823/3823

Epoch 53/100

4s - loss: 0.8903 - 3823/3823

Epoch 54/100

4s - loss: 0.8900 - 3823/3823

Epoch 55/100

4s - loss: 0.8903 - 3823/3823

Epoch 56/100

4s - loss: 0.8902 - 3823/3823

Epoch 57/100

4s - loss: 0.8898 - 3823/3823

Epoch 58/100

4s - loss: 0.8900 - 3823/3823

Epoch 59/100

4s - loss: 0.8902 - 3823/3823

Epoch 60/100

4s - loss: 0.8902 - 3823/3823

Epoch 61/100

4s - loss: 0.8902 - 3823/3823

Epoch 62/100

4s - loss: 0.8901 - 3823/3823

Epoch 63/100

4s - loss: 0.8889 - 3823/3823

Epoch 64/100

4s - loss: 0.8906 - 3823/3823

Epoch 65/100

4s - loss: 0.8899 - 3823/3823

Epoch 66/100

4s - loss: 0.8899 - 3823/3823

Epoch 67/100

4s - loss: 0.8900 - 3823/3823

Epoch 68/100

4s - loss: 0.8902 - 3823/3823

Epoch 69/100

4s - loss: 0.8902 - 3823/3823

Epoch 70/100

4s - loss: 0.8902 - 3823/3823

Epoch 71/100

4s - loss: 0.8900 - 3823/3823

Epoch 72/100

4s - loss: 0.8900 - 3823/3823

Epoch 73/100

4s - loss: 0.8903 - 3823/3823

Epoch 74/100

4s - loss: 0.8901 - 3823/3823

Epoch 75/100

4s - loss: 0.8904 - 3823/3823

Epoch 76/100

4s - loss: 0.8898 - 3823/3823

Epoch 77/100

4s - loss: 0.8896 - 3823/3823

Epoch 78/100

4s - loss: 0.8902 - 3823/3823

Epoch 79/100

4s - loss: 0.8902 - 3823/3823

Epoch 80/100

4s - loss: 0.8897 - 3823/3823

Epoch 81/100

4s - loss: 0.8902 - 3823/3823

Epoch 82/100

4s - loss: 0.8902 - 3823/3823

Epoch 83/100

4s - loss: 0.8900 - 3823/3823

Epoch 84/100

4s - loss: 0.8902 - 3823/3823

Epoch 85/100

4s - loss: 0.8903 - 3823/3823

Epoch 86/100

4s - loss: 0.8902 - 3823/3823

Epoch 87/100

4s - loss: 0.8898 - 3823/3823

Epoch 88/100

4s - loss: 0.8902 - 3823/3823

Epoch 89/100

4s - loss: 0.8903 - 3823/3823

Epoch 90/100

4s - loss: 0.8901 - 3823/3823

Epoch 91/100

4s - loss: 0.8902 - 3823/3823

Epoch 92/100

4s - loss: 0.8902 - 3823/3823

Epoch 93/100

4s - loss: 0.8897 - 3823/3823

Epoch 94/100

4s - loss: 0.8903 - 3823/3823

Epoch 95/100

4s - loss: 0.8898 - 3823/3823

Epoch 96/100

4s - loss: 0.8903 - 3823/3823

Epoch 97/100

4s - loss: 0.8902 - 3823/3823

Epoch 98/100

4s - loss: 0.8901 - 3823/3823

Epoch 99/100

4s - loss: 0.8899 - 3823/3823

Epoch 100/100

4s - loss: 0.8899 - 3823/3823

<tensorflow.python.keras.callbacks.History at 0x7f8fa3baec10>

خروجی:

```
trainPredict22 = model.predict(tensor_xtrain)
testPredict22 = model.predict(tensor_xtest)
ytrain = np.reshape(ytrain, (ytrain.shape[0], 1))
,trainScore22 = math.sqrt(mean_squared_error(ytrain
((trainPredict22[:,0]
print('Train Score: %.2f RMSE' % (trainScore22))
testScore22 = math.sqrt(mean_squared_error(ytest, testPredict22[:,0]))
print('Test Score: %.2f RMSE' % (testScore22))
```

Train Score: 0.94 RMSE

Test Score: 1.00 RMSE