گزارش پروژه‌ی شناسایی آماری الگو

عنوان پروژه:

پیاده‌سازی مقاله‌ی

Multi-label classification with Bayesian network-based chain classifiers

فرهاد دلیرانی

dalirani.1373@gmail.com

۹۶۱۳۱۱۲۵

۰ – مقدمه

در Multi-label Classification هدف اختصاص یک یا بیش از یک برچسب به یک نمونه اس ت. روش‌ه ای مختلف ی برای حل این مسئله وجود دارند که معمولا از هزینه‌ی محاسباتی بالا و یا نادیده گرفتن رابطه‌ی بین داده‌ها رنج می‌برند. در این گزارش، در فصل اول به معرفی Multi-label Classification می‌پردازی م و بعض ی از مهم‌تری ن روش‌های موجود را معرفی می‌کنیم و ویژگی‌های خوب و بد آن‌ها را بررسی می‌کنیم. سپس دو روش، شبکه‌های بیزین چند بعدی و دسته‌بندهای زنجیره‌ای را معرفی می‌کنیم که پایه‌ی روش پیشنهادی مقاله هستند. بعد از معرفی آن دو، روش پیشنهادی مقاله‌ی انتخاب شده برای پیاده‌سازی را که قدرت شبکه‌های بیزین چند بعدی و دسته‌بندهای زنجیره‌ای را ترکیب می‌کند را، به صورت کامل توضیح می‌دهیم و بررسی می‌کنیم که چرا روش پیشنهادی در ساده ترین شکل پیاده‌سازی خود، به راحتی بسیاری از روش‌های مورد استفاده را شکست می‌دهد. در فصل دو به پیاده‌سازی مقاله می‌پردازیم و قسمت‌های مختلف و تابع‌های برنامه را توضیح می‌دهیم و در فصل سوم نتیجه‌های حاصل از اجرای کدها را ارائه می‌کنیم و عملکرد روش پیشنهادی را بررسی می‌کنیم، همین طور نتایج به دست‌آمده را با نتایج موجود در مقاله مقایسه می‌کنیم تا درستی پیاده‌سازی انجام شده را نشان دهیم.

۱-خلاصه‌ی مقاله‌ی Multi-label classification with Bayesian network-based chain classifiers

برای پروژه‌ی درس شناسایی آماری الگو مقاله‌ی زیر را انتخاب کرده ام که در بخش اول گزارش، خلاصه‌ای از آن را ارایه می‌کنم:

Multi-label classification with Bayesian network-based chain classifiers

Authors:
L. Enrique Sucar, Concha Bielza , Eduardo F. Morales , Pablo Hernandez-Leal, Julio H. Zaragoza , Pedro Larrañaga

Keywords:
{Multi-label classification, Chain classifier, Bayesian networks}

Pattern Recognition Letters 41 (2014) 14–22

در classification چند- برچسب[1]، هدف اختصاص دادن یک یا بیش از یک برچست به یک نمونه است. این کار معمولا به دو روش انجام می‌شود: ۱- ایجاد مجموعه‌ی توانی [2] برچسب‌ها و استفاده از آن‌ها به عنوان برچسب‌های جدید که به این روش مجموعه‌ی توانی برچسب‌ها [3] می‌گویند. ۲- ایجاد و آموزش یک classifier به ازای هر کلاس که به آن، Binary Relevance می‌گویند. روش اول از پیچیدگی محاسباتی بالا رنج می‌برد و روش دوم وابستگی بین کلاس‌ها را نادیده می‌گیرد. دسته‌بندهای Chain Classifier اخیرا ارایه شده اند که این دو مشکل را برطرف می‌کنند. هر classifier در زنجیره علاوه بر ویژگی‌های نمونه‌ها، خروجی سایر دسته‌بندهای پیشین بر روی زنجیره را می‌گیرد. این مقاله روشی را ارایه می‌دهد که قدرت **دسته‌بندهای زنجیره‌ای و شبکه‌های بیزین** [4] را برای Multi-label Classification ترکیب می‌کند و از یک ساختار زنجیر-درخت مانند استفاده می‌کند که بر اساس شبکه‌ی بیزین ساخته شده است که به آن Bayesian Network Based Chain می‌گویند.

در زمینه‌ی PGM[5] دو استراتژی برای رفع محدودیت‌های روش مجموعه‌ی توانی و روش Binary Relevance پیشنهاد شده است:

1 Multi-label classification
2 Power set
3 Label power-set methods
4 Bayesian network
5 Probabilistic Graphical Model

۱- استفاده از Chain Classifiers

۲- استفاده از شبکه‌های چند بعدی بیزین.

Chain Classifier از d دسته‌بند تشکیل شده اند که هر classifier بر روی زنجیره، خروجی دسته‌بندهای پیش از خود بر روی زنجیره را به عنوان ویژگی اضافه دریافت می‌کند. یکی از مشکل‌های این روش افزایش تعداد ویژگی‌ها است.

شبکه‌های چند بعدی بیزین نوعی شبکه‌ی بیزین هستند که برای حل مسائل دسته‌بندی چند برچسب (و یاچند بعدی) طراحی شده اند. این شبکه‌ها از سه زیر گراف تشکیل شده‌اند:

۱- متغییرهای کلاس[6]

۲- متغییرهای ویژگی[7]

۳- ساختار پل[8]، یال‌هایی که زیرگراف کلاس و ویژگی ها را به‌هم وصل می‌کند.

در ادامه هر کدام را بیشتر توضیح خواهم داد.

دسته‌بندهای زنجیره‌ای بیزین[9] هر دو استراتژی بالا را ترکیب می‌کنند و از قدرت هر کدام به طور هم زمان استفاده می‌کند.

ساخت classifier زنجیره‌ای بیزین **دو فاز اصلی** دارد: ۱- به دست آوردن ساختار وابستگی کلاس‌ها ۲- ساخت دسته‌بند زنجیره‌ای بیزین با توجه به ساختار وابستگی کلاس‌ها. ساختار وابستگی کلاس‌ها در فاز اول با استفاده از شبکه‌ی بیزین به دست می‌آید که به عنوان یک راهنما برای فاز دو عمل می‌کند در فاز یک می‌توان از الگوریتم‌هایی مانند

Approximating Discrete Probability Distributions with Dependence Trees

By C.K. CHOW AND C.N.LIU

IEEE TRANSACTIONS ON INFORMATION THEORY,VOL.IT-14,NO.3, MAY 1968

استفاده کرد. در آخر بعد از ساخت زنجیره بر اساس رابطه‌ی وابستگی بین کلاس‌ها، خروجی هر نمونه بر اساس

6 class variables
7 feature variables
8 bridge structure
9 Bayesian Chain Classifiers (BCC)

خروجی هر دسته‌بند روی زنجیره به دست می‌آید.

یک مدل زنجیره‌ای بیزین ساده که از یک **ساختار درختی در فاز اول** و دسته‌بند کننده‌ی **بیز ساده**[10] به عنوان دسته‌بند پایه استفاده می‌کند، بسیاری از الگوریتم‌های مطرح دسته‌بندی چند- برچسب را به راحتی شکست می‌دهد.

در ادامه ابتدا دسته‌بندی چند بعدی و شبکه‌های چند بعدی بیزین را معرفی کنیم و بعد از آن دسته‌بندهای زنجیره‌ای را معرفی می‌کنیم و سپس از آن دو به دسته‌بندهای زنجیره‌ای بیزین می‌رسیم.

Multidimensional Classification برابر است با جست و جوی تابع h که هر نمونه که با m ویژگی $x=(x_1,x_2,\ldots,x_n)$ نشان داده می‌شود را به وکتوری شامل d کلاس نگاشت می‌کند.

$$h : \Omega_{X_1} \times \cdots \times \Omega_{X_m} \rightarrow \Omega_{C_1} \times \cdots \times \Omega_{C_d}$$

$$(x_1, \ldots, x_m) \mapsto (c_1, \ldots, c_d)$$

تحت zero-one loss function تابع h باید به هر نمونه x محتمل‌ترین مقدار از کلاس‌ها را نسبت بدهد به طوری که:

$$\arg\max_{c_1,\ldots,c_d} P(C_1 = c_1, \ldots, C_d = c_d | \mathbf{x})$$

که البته این مساله از نوع NP-Hard است.

مساله‌ی **Multi-label Classification** نوعی مساله‌ی دسته‌بندی چند بعدی است که هر متغییر کلاس به صورت دودویی است.

$$|\Omega_{C_i}| = 2 \text{ for } i = 1, \ldots, d$$

10 Naive Bayes

یک Classifier چند بعدی بیزین[11] (MBC) روی مجموعه‌ی$V = \{Z_1,Z_2, \ldots, Z_n\}$ و متغییرهای تصادفی گسسته به ازای n های بزرگ‌تر مساوی یک، شبکه‌ی بیزین $B=(g,\theta)$ است به طوری که g یک گراف جهت دار بدون دور (DAG) است و θ مجموعه‌ی پارامترهای

$$\theta_{z|\mathbf{pa}(z)} = P(z|\mathbf{pa}(z))$$

است. pa(z) مجموعه‌ی همه‌ی والدین متغییر z در g است.در نتیجه احتمال توام[12] متغییرها برابر می‌شود با:

$$P_B(z_1, \ldots, z_n) = \prod_{i=1}^{n} P_B(z_i|\mathbf{pa}(z_i))$$

مجموعه‌ی متغییرهای تصادفی V خود شامل دو بخش است:

۱- $V_c = \{C_1,C_2,\ldots,C_d\}$ متغییرهای کلاس

۲- $V_x = \{X_1,\ldots,X_m\}$ متغییرهای ویژگی

مجموعه‌ی یال‌ها A را هم می‌توان به سه بخش تقسیم کرد: ۱- Ac که متغییرهای کلاس را به هم متصل می‌کند ۲- Ax که متغییرهای ویژگی را بهم متصل می‌کند ۳- Acx که متغییرهای کلاس را به متغییرهای ویژگی وصل می کند.
به این ترتیب می‌توان گراف g را به سه زیرگراف تقسیم کرد:

$$\mathcal{G}_C = (\mathcal{V}_C, \mathcal{A}_C), \mathcal{G}_X = (\mathcal{V}_X, \mathcal{A}_X) \text{ and } \mathcal{G}_{CX} = (\mathcal{V}, \mathcal{A}_{CX}),$$

که به آن‌ها زیرگراف‌های کلاس، ویژگی و پل می‌گویند. در شکل زیر یک شبکه‌ی چند بعدی بیزین را مشاهده می‌کنید.

11 multidimensional Bayesian network classifier
12 Joint Probability

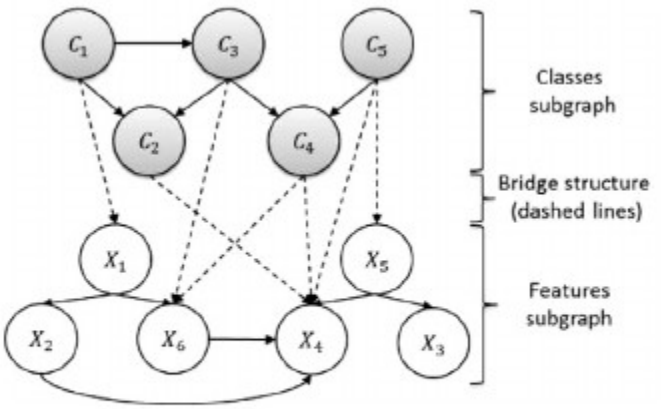

Fig. 1. A multi-dimensional Bayesian network classifier structure, showing the three subgraphs: classes, features and bridge.

دسته‌بندهای زنجیره‌ای، به عنوان یک روش در برخورد با مساله‌ی دسته‌بندی چند- برچسب استفاده می‌شوند. یک دسته‌بند زنجیره‌ای از d دسته‌بند متصل به هم، در یک زنجیره تشکیل شده است. هر classifier در زنجیره فقط و فقط اقدام به برچسب زدن یک نوع کلاس می‌کند. هر دسته‌بند بر روی زنجیره خروجی دسته‌بندهای پیشین روی زنجیره را به عنوان ویژگی اضافه دریافت می‌کند. هر دسته‌بندکننده‌ی Ci بر روی زنجیره ویژگی داده‌ها و همین طور خروجی کلاس‌های C_1 تا C_{i-1} بر روی زنجیره را به عنوان ویژگی اضافه می‌گیرد. در هنگام برچسب‌زنی عملیات از C_1 شروع می‌شود و به تمام زنجیره گسترش پیدا می‌کند. برای هر Ci

$$\arg\max_{c_i} P(c_i|\mathbf{x}, c_1, c_2, \ldots, c_{i-1})$$

محاسبه می‌شود. در پایان از الحاق خروجی تمام دسته‌بندهای روی زنجیره، کلاس‌های نسبت داده شده به یک نمونه به دست می‌آید.

در دسته‌بندی چند-برچسب هدف پیدا کردن کلاس‌هایی است که عبارت زیر را ماکسیمایز کند

$$\arg\max_{c_1,\ldots,c_d} P(C_1 = c_1, \ldots, C_d = c_d|\mathbf{x})$$

اگر Chain Rule را اعمال کنیم عبارت

$$\arg\max_{c_1,\ldots,c_d} P(c_1|c_2,\ldots,c_d,\mathbf{x})P(c_2|c_3,\ldots,c_d,\mathbf{x})\cdots P(c_d|\mathbf{x}) \qquad (4)$$

به دست می‌آید. و اگر وابستگی ها و استقلال متغییرها در شبکه‌ی بیزین را اعمال کنیم والدین هر متغییر ارزش پیدا می‌کنند و می توان عبارت شماره‌ی ۴ را به این صورت نوشت:

$$\arg\max_{c_1,\ldots,c_d} \prod_{i=1}^{d} P(c_i|\mathbf{pa}(C_i),\mathbf{x}) \qquad (5)$$

و برای ساده سازی بیشتر فرض می کنیم می‌توان عبارت شماره‌ی ۵ را به صورت عبارت‌های مستقل از هم به صورت زیر نوشت:

$$\arg\max_{c_1} P(c_1|\mathbf{pa}(C_1),\mathbf{x})$$
$$\arg\max_{c_2} P(c_2|\mathbf{pa}(C_2),\mathbf{x})$$
$$\ldots$$
$$\arg\max_{c_d} P(c_d|\mathbf{pa}(C_d),\mathbf{x})$$

با این ساده سازی از **شبکه‌های بیزین به (Bayesian Chain Classifier(BBC** می‌رسیم:

a BCC makes two basic assumptions:

1. The class dependency structure given the features can be represented by a Bayesian network.
2. The most probable joint combination of class assignment (total abduction) is approximated by the concatenation of the most probable individual classes.

با این فرض ها ساخت یک دسته‌بند زنجیره‌ای را می‌توان با کلاسی که به بقیه وابستگی ندارد شروع کرد و بعد به سراغ فرزندانش در Dependency Structure رفت و به همین ترتیب ادامه داد. به همین دلیل باید بر اساس ساختار وابستگی یک ترکیب در ظاهر شدن دسته‌بندها بر روی زنجیره ایجاد کنیم.

دسته‌بند ساده‌ی زنجیره‌ای بیزین درختی[13]: ساده ترین دسته بند زنجیره‌ای بیزین است که در زنجیره هر متغییر یک والد دارد و در نتیجه یک درخت است و ساختار وابستگی را می‌توان با الگوریتمChow و Liu به دست آورد.یال‌های خروجی این الگوریتم جهت ندارد به همین دلیل می توان تصادفی یک گره را ریشه در نظر گرفت و درخت را پیمایش کرد و در حین پیمایش جهت را مشخص کرد. با d کلاس می‌توانند d درخت وابستگی متفاوت ایجاد کرد.

الگوریتم ساخت دسته‌بند(**TNBCC (Tree Naive Bayes Chain Classifier** را در تصویر زیر مشاهده می‌کنید:

13 Tree Naive Bayesian chain classifier (TNBCC)

1. Build an undirected tree to approximate the dependency structure among class variables.
2. Create an order for the chain classifier by randomly selecting one class as the root of the tree and assigning the rest of the links in order.
3. For each class variable (node) in the chain, build a naïve Bayes classifier for class C_i which has as attributes its parent $\mathbf{Pa}(C_i)$ and all the features $\mathbf{x}$, taking advantage of the conditional independence properties.
4. To classify a new instance concatenate the outputs of the chain.

براس ساخت TNBCC در **مرحله‌ی یک** ساختار وابستگی بین کلاس‌ها استخراج می‌شود که می‌توان از Chow & Liu's Algorithm استفاده کرد. در **مرحله‌ی دو** از روی درخت بدون جهت وابستگی، یک درخت جهت‌دار ساخته می‌شود. در **مرحله‌ی سه** برای هر برچسب(کلاس) یک classifier ساخته می‌شود که برای آموزش آن از ویژگی‌های نمونه‌ها و کلاس والد آن در درخت وابستگی به عنوان ویژگی اضافه استفاده می‌کنیم. در **مرحله‌ی چهارم** برای تعیین برچسب‌های یک نمونه از ریشه‌ی درخت شروع به حرکت می‌کنیم و در راه ریشه به برگ‌ها خروجی هر classifier را محاسبه می‌کنیم، و در انتها برچسب‌های نمونه برابر است با تمام برچسب‌هایی که دسته‌بندها به آن نمونه داده‌اند.

در تصویر زیر یک TNBCC را مشاهده می‌کنیم که classifier ها بر اساس ساختار وابستگی به هم وصل شده‌اند و هر دسته‌بند یک پدر دارد و هر دسته‌بند علاوه بر ویژگی‌های داده‌ها خروجی دسته‌بند پدر را به عنوان ویژگی اضافه دریافت می‌کند.

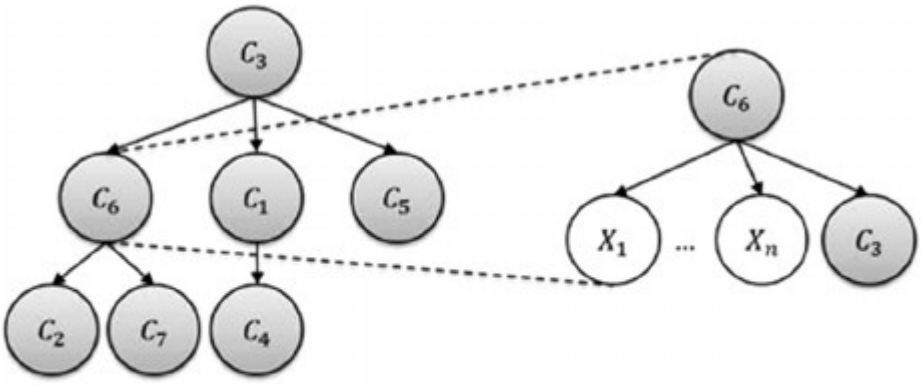

Fig. 3. An example of a Tree Naïve Bayesian Chain Classifier where each node (C_6, for instance) in the chain yields a naïve Bayesian classifier which has as attributes its parent class (C_3) and all the features ($X_1, \ldots, X_n$).

عملکرد TNBCC را می‌توان با تغییر در موردهای زیر بهبود داد که در مقاله برای هر کدام به موردهایی اشاره شده است:

۱- نحوه‌ی آموزش: در هنگام آموزش وقتی از کلاس پدر به عنوان ویژگی اضافه برای یک classifier استفاده می‌کنیم می‌توان خروجی دسته‌بند پدر برای آن نمونه را محاسبه کرد و سپس استفاده کرد و یا از مقدار واقعی آن کلاس برای آن نمونه که در دیتاست آمده است استفاده کرد که بر اساس مقاله شیوه‌ی دوم بهتر است.

۲- ترتیبی که کلاس‌ها برای قرارگیری در زنجیره انتخاب می‌شوند: می‌توان در هنگام ساخت درخت جهت دار وابستگی از روی درخت Chow و Liu، ریشه‌ی درخت را تصادفی انتخاب کرد و یا گره‌ای را به عنوان ریشه انتخاب کرد که بیشترین یال را دارد.

۳- پیچیدگی زنجیر: می‌توان فقط از پدر یک دسته‌بند به عنوان ویژگی اضافه استفاده کرد و یا از روش‌های دیگر مانند در نظر گرفتن تمام دسته‌بندهای در مسیر ریشه تا آن دسته‌بند به عنوان ویژگی اضافه استفاده کرد.

۴- دسته‌بند پایه: از دسته‌بندهای مختلفی می‌توان به عنوان دسته‌بند پایه استفاده کرد مانند SVM و یا Naive Bayes

۵- استفاده از یک زنجیر یا چندین زنجیر: می‌توان از همان یک ساختار درختی زنجیره‌ای استفاده کرد یا با استفاده از تکنیک‌های ensemble مجموعه‌ای از زنجیره‌های درختی را ساخت و از نتایج آن‌ها برای تعیین نتیجه‌ی نهایی

استفاده کرد.

برای ارزیابی مقاله چند معیار زیر استفاده شده‌اند:

1. *Mean accuracy* over the d class variables (accuracy per label):

$$\mathrm{M-Acc} = \frac{1}{d}\sum_{j=1}^{d} Acc_j = \frac{1}{d}\sum_{j=1}^{d}\frac{1}{N}\sum_{i=1}^{N}\delta(c'_{ij}, c_{ij}) \qquad (6)$$

2. *Global accuracy* over the d-dimensional class variable (accuracy per example, also called subset zero-one loss):

$$\mathrm{G-Acc} = \frac{1}{N}\sum_{i=1}^{N}\delta(\mathbf{c}'_i, \mathbf{c}_i)$$

3. *Multi-label accuracy*, also called Jaccard measure, as defined in Tsoumakas and Katakis (2007):

$$\mathrm{ML-Acc} = \frac{1}{N}\sum_{i=1}^{N}\frac{|\mathbf{c}_i \wedge \mathbf{c}'_i|}{|\mathbf{c}_i \vee \mathbf{c}'_i|} \qquad (8)$$

4. *F-measure* is the harmonic mean between precision and recall:

$$\mathrm{F-measure} = \frac{1}{d}\sum_{j=1}^{d}\frac{2p_j r_j}{(p_j + r_j)} \qquad (9)$$

در ادامه مقاله، نویسندگان به ارائه و مقایسه نتایجشان با آخرین روش‌ها برای دسته‌بندی چند برچسب پرداخته‌اند.

داده های آموزش و تست پروژه:

در مقاله به داده‌های استفاده شده اشاره شده است که آن‌ها از لینک‌های زیر قابل دسترس هستند:

http://mulan.sourceforge.net/datasets.html
http://mlkd.csd.auth.gr/multilabel.html
http://www.cs.waikato.ac.nz/ml/weka/index.html

۲-پیاده‌سازی

برای پیاده سازی مقاله‌ی دسته‌بندی چند-برچسب به وسیله‌ی زنجیر بر اساس شبکه‌های بیزین، علاوه بر روش اصلی مقاله که Tree Naive Bayes Chain Classifier است موردهای دیگری را هم پیاده‌سازی کرده‌ام:

- **Bayesian Network Based-Chain: Tree Naive Bayes Chain Classifier**
- **Binary Relevance: Naive Bayes As Base Classifier**
- **Bayesian Network Based-Chain: Tree SVM Chain Classifier**
- **Binary Relevance: SVM As Base Classifier**

کدها با زبان برنامه نویسی Python 2.7.11 نوشته شده اند و از پکیج‌های زیر استفاده شده است:

- Numpy (1.11.3)
- Networkx (باید از ورژن ۱.۱۱ استفاده شود، ورژن‌های بالاتر حاوی یک باگ اصلاح نشده هستند.)
- Sklearn

سیستم عاملی که در آن پیاده‌سازی را انجام دادم لینوکس/گنوم (توزیع فدورا ۲۳) است، کدها را هم در ویندوز ۱۰ تست کردم و در صورتی که از ورژن‌های اشاره شده پایتون و پکیج‌ها استفاده شود بدون مشکل اجرا می‌شوند.

فایل MWST.py

در این فایل تابع‌های مربوط به ساخت درخت بدون جهت وابستگی بین کلاس‌ها توسط الگوریتم Chow and Liu قرار دارند. الگوریتم Chow and Liu در مقاله‌ی زیر شرح داده شده است:

Approximating Discrete Probability Distributions with Dependence Trees

در این فایل 4 تابع وجود دارد که آن‌ها را در ادامه توضیح می‌دهم.

```
def marginal_probabilities(dataset, u):
    """
    :param data-set: Data-set
    :param u: Index of u'th features
    :return: The marginal probabilities for the u'th features of the data-set.
             return is a default dictionary object.
    """
```

این تابع دیتاستی که با حذف ویژگی‌های نمونه‌ها از روی دیتاست اصلی و نگه داشتن کلاس‌های هر نمونه ساخته شده است را می‌گیرد و احتمال مقدارهای مختلف ویژگی u را محاسبه می‌کند و آن احتمال‌ها را برمی گرداند.

```
def marginal_pair_probabilities(dataset, u, v):
    """
    :param X: Data-set
    :param u: Index u'th features
    :param v: Index v'th features
    :return: The marginal probabilities for the u'th and v'th features of the data-set.
             return is a default dictionary object.
    """
```

این تابع دیتاست جدیدی که در بالا توضیح داده شد را می‌گیرد و احتمال تمام جفت مقدارهای مختلف دو ویژگی u و v را محاسبه می‌کند.

```
def mutual_information(dataset, u, v):
    """
    :param dataset: Data-set
    :param u: Index of u'th features
    :param v: Index of v'th features
    :return: return mutual information of u'th and v'th features
    """
```

این تابع بر اساس تابع زیر میزان اطلاعات مشترک دو ویژگی در دیتاست جدید (کلاس در دیتاست اولیه) را محاسبه می‌کند:

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log \left(\frac{p(x,y)}{p(x)\,p(y)} \right),$$

.

```
def chow_liu_tree(dataset, n):
    """
    This function makes a chow liu's tree. First it calculates
    mutual information among all features, then it multiply all
    mutual information by -1 and at last it builds chow & liu's
    tree by constructing minimum spanning tree of a complete graph
    which its nodes are features and its edges are mutual information
    which were multiplied by -1.

    :param dataset: Data-set
    :param n: Number of features
    :return: An undirected tree which is a networkx object.
    """
```

این تابع دیتاست و تعداد فیچرها را می‌گیرد و ابتدا با استفاده از سه تابع بالا یک گراف تشکیل می‌دهد که گره‌های آن فیچرهای مختلف هستند و یال‌های بین آن‌ها mutual information بین هر دو گره ممکن، ضرب در منفی یک است. سپس الگوریتم minimum spanning tree بر آن گراف اعمال می‌شود. با این کار درخت بدون جهت Chow و Liu ساخته می‌شود. گره‌های این درخت ویژگی‌های دیتاست جدید است که از دیتاست اصلی با نگه داشتن کلاس‌های هر نمونه و حذف ویژگی‌های نمونه‌ها ساخته شده است. در نتیجه گره‌های درخت کلاس‌ها هستند و یال‌ها منفی mutual information بین دو کلاس دو سر یال هستند.

فایل dependencyTreeOfClasses.py

در این فایل سه تابع برای استخراج اطلاعات مورد نیاز از روی درخت Chow و Liu قرار دارد.

```
def create_directed_tree_from_chow_liu_tree(chowLiuTree):
    """
    Create a directed tree from a Chow and Liu's tree(MWST) which is bi-directed.
    :param chowLiuTree: is an undirected graph. a networkx object.
    :return: return a pair (root, directedTree),
             root: is the root of directed tree
             directedTree: a directed tree which is built from a Chow and Liu's tree(MWST).
             the tree returned as a networkx.digraph object.
    """
```

این تابع یک Undirectional Tree را که توسط تابع chow_liu_tree ساخته می‌شود را می‌گیرد و به صورت تصادفی یک گره از آن را به عنوان ریشه انتخاب می‌کند. سپس از آن گره به صورت preorder شروع به پیمایش درخت می‌کند و یک درخت جهت دار می‌سازد.

```
def topological_sort_of_classifiers(rootDirectedTree):
    """
    This function converts directed tree which is built from Chow & Liu's
    tree to a topological order of classes.
    :param rootDirectedTree:  a pair (root, directedTree),
             root: is the root of directed tree
             directedTree: a directed tree which is built from a Chow and Liu's tree(MWST).
             the tree returned as a networkx.digraph object.
    :return: a pair (parents, topologicalSort):
                parents: a dictionary that contains parent of each node,
                topologicalSort: a topological sort of classifiers
    """
```

این تابع درخت جهت داری را که با استفاده از تابع قبل می‌سازیم را می‌گیرد. ابتدا با استفاده از آن درخت topological sort درخت را محاسبه می‌کند. برای محاسبه‌ی برچسب‌های یک کلاس ابتدا از دسته‌بندی شروع می‌کنیم که به سایر کلاس‌ها وابسته نیست و ریشه‌ی درخت وابستگی است و سپس به سراغ دسته‌بندهایی می‌رویم که در درخت وابستگی فرزند آن دسته‌بند هستند و این کار را باید تا زمان رسیدن به دسته‌بندهای موجود در برگ‌های درخت وابستگی ادامه دهیم، سپس با کنار هم قرار دادن کلاس‌های پیشبینی شده توسط همه‌ی دسته‌بندها، برچسب‌های یک ورودی را به دست آوریم. به همین دلیل به topological sort درخت نیاز داریم تا هنگام محاسبه‌ی پیشبینی کلاس یک نمونه توسط یک دسته‌بند در درخت دسته‌بندها، کلاسی که آن نمونه به عنوان ویژگی اضافه برای پیشبینی دریافت می‌کند قبلا آماده شده باشد. علاوه بر توپولوژی سورت درخت وابستگی این تابع والد هر گره را نیز محاسبه می‌کند. در پایان آن دو مقدار محاسبه شده را به عنوان خروجی باز می‌گرداند.

```
def create_dataset_for_chow_liu_tree(dataset):
    """
    This function creates a dataset for chow and liu's MWST algorithm,
    it just contain classes of each observation in this form:

    [[class0 sample0, class1 sample0,...,classN sample0],
     [class0 sample1, class1 sample1,...,classN sample1],
     ...,
     [class0 sampleN, class1 sampleN,...,classN sampleN]]

    :param dataset:
    :return: appropriate dataset for creating chow & liu tree
    """
```

این تابع دیتاستی را که از فایل خوانده‌ایم را می‌گیرد و تمام ویژگی‌های نمونه‌ها را حذف می‌کند و فقط کلاس‌های هر observation را نگه می‌دارد. با اینکار یک دیتاست جدید از کلاس‌ها می‌سازد و آن را باز می‌گرداند. در قسمت‌های دیگر کد، دیتاست خروجی این تابع به تابع ساخت درخت Chow and Liu داده می‌شود و آن درخت بدون جهت وابستگی بین کلاس‌ها را می‌سازد.

فایل read_inputs.py

در این فایل 4 تابع برای خواندن دیتاست‌ها از فایل‌ها و آماده کردن دیتاست برای classifierها قرار دارد.

```
def read_input_discrete(filePath):
    """
    This function reads discrete datasets from file.
    :param filePath: path of data-set
    :return: a list in this form:
            [[x0 feature0, x0 feature1,..., x0 featureN, x0 class0, ..., x0 classM],
             [x1 feature1, x1 feature1,..., x1 featureN, x1 class0, ..., x1 classM],
             ...
             [xn feature0, xn feature1,..., xn featureN, xn class0, ..., xn classM]]
    """
```

این تابع نام فایل و فولدر دیتاستی که داده‌های آن گسسته است و فولدر آن در مسیر پروژه قرار دارد را به عنوان ورودی می‌گیرد و سپس دیتاست را از فایل درون فولدر هم نام فایل می‌خواند و لیستی از لیست‌ها را همانطور که در تصویر بالا شرح داده شده است را برمی‌گرداند.

```
def read_input_continues(filePath):
    """
        This function reads continues datasets from file.
        :param filePath: path of data-set
        :return: a list in this form:
                [[x0 feature0, x0 feature1,..., x0 featureN, x0 class0, ..., x0 classM],
                 [x1 feature1, x1 feature1,..., x1 featureN, x1 class0, ..., x1 classM],
                 ...
                 [xn feature0, xn feature1,..., xn featureN, xn class0, ..., xn classM]]
        """
```

این تابع نام فولدر دیتاستی که داده‌های آن پیوسته است و فولدر آن در مسیر پروژه قراد دارد را به عنوان ورودی می‌گیرد و سپس دیتاست را از فایل می‌خواند و لیستی از لیست‌ها را همان طور که در تصویر بالا شرح داده شده است را برمی‌گرداند.

از آنجایی که classifier هایی که برای پروژه نوشته‌ام از نوع Binary Relevance و Bayesian Based Chain هستند برای آماده کردن دیتاست برای هر کدام از آن دو نوع کلاسیفایر به یک تابع جدا نیاز داریم:

```
def create_training_set_for_binary_relevance_classifier(train, classOfClassifier):
    """
    This function gets a multi-label dataset and prepares a new train dataset
    for each classifier.
    Training set for each classifiers in binary relevance just
    consists of features and the class of classifier.

    :param train: training data
    :param classOfClassifier: determines a class that classifiers wanted to
                                predict data belongs to it or not
    :return:return a pair (new training set, actual class of training set)
            new training set: a new training set for the classifier in this form
            [[feature0,..., featureN, class parent],
             ...,
             [feature0,..., featureN, class parent]]

             actual class of training set: which is a list in this form:
            [class of observation 0 {0,1}, ..., class of observation N {0,1}]
    """
```

در روش binary relevance برای Multi-label Classification به ازای هر کلاس یک Classifier در نظر می‌گیریم که ویژگی‌های دیتاست و مقدار آن کلاس در دیتاست را برای آموزش می‌گیرد و نیازی به خروجی سایر کلاسیفایرها برای آموزش ندارد و هر کلاسیفایر از سایر کلاسیفایرها مستقل است. این تابع دیتاست چند-برچسب را می‌گیرد و از هر observation در دیتاست فقط فیچرها و کلاس متناظر با آن کلاسیفایر را نگه می‌دارد و در آخر فیچرها و کلاس متناظر با کلاسیفایر را به صورت دو لیست مجزا باز می‌گرداند که به عنوان ورودی برای آموزش آن کلاسیفایر در سایر نقاط برنامه مورد استفاده قرار می‌گیرد.

خروجی های تابع بالا برای کلاسیفایر I ام:

[[X0 feature 0, ..., X0 feature n], ..., [Xm feature 0, ..., Xm feature n]]

[X0 value of class i, X1 value of class I, ..., Xm value of class I]

```
def create_training_set_for_bayesian_based_chain_classifier(train, classOfClassifier,
                                                            parentClassOfClassifier):
    """
    This function gets a mulilabel dataset and prepares a new train dataset
    for each classifier.
    Training set for each classifiers in a bayesian based chain manner
    consists of features, output of parent classifier as feature and
    the class of classifier.

    :param train: training data
    :param classOfClassifier: determines a class that classifiers wanted to
                              predict data belongs to it or not
    :param parentClassOfClassifier: parent of classifier
    :return:return a pair (new training set, actual class of training set)
            new training set: a new training set for the classifier in this form
            [[feature0,..., featureN, class parent],
             ...,
             [feature0,..., featureN, class parent]]

             actual class of training set: which is a list in this form:
            [class of observation 0 {0,1}, ..., class of observation N {0,1}]
    """
```

در روش Bayesian-based chain برای Multi-label Classification به ازای هر کلاس یک Classifier در نظر می‌گیریم که ویژگی‌های دیتاست و مقدار آن کلاس در دیتاست را برای آموزش می‌گیرد و علاوه بر فیچرها، خروجی کلاسیفایر پدر در درخت وابستگی را هم به عنوان فیچر اضافه می‌گیرد. این تابع دیتاست چند-برچسب را می‌گیرد و از

هر observation در دیتاست فقط فیچرها و کلاس متناظر با آن کلاسیفایر و کلاس کلاسیفایر پدر را نگ ه می‌دارد و در آخر فیچرها را به همراه کلاس کلاسیفایر پدر و کلاس متناظر با کلاسیفایر را به صورت دو لیست مجزا باز می‌گرداند که به عنوان ورودی برای آموزش آن کلاسیفایر در سایر نقاط برنامه مورد استفاده قرار می‌گیرد.

[[X0 feature 0, ..., X0 feature n, X0 value of class parent], ...,

[Xm feature 0, ..., Xm feature n,, X0 value of class parent]]

[X0 value of class i, X1 value of class I, ..., Xm value of class i]

فایل BRM_NB_DISCRETE.py

در این فایل سه تابع موجود است، که برای ایجاد یک Binary Relevance است که کلاس یفایرهای پ ایه‌ی آن Naive Bayes است. در این روش برای حل مسائل Multi-label Classification ب ه ازای ه ر کلاس ی ک کلاس یفایر س اخته می‌شود که از ویژگی‌های داده‌ها برای آموزش استفاده می‌کنند و هر کلاسیفایر از سایر کلاسیفایرها مستقل است. این دسته‌بند برای دیتاست‌های گسسته است. همین طور که در مقاله توضیح داده شده است اگر دیتاست پیوسته باشد با توجه به مقاله‌های خاصی که در مقاله به آن‌ها اشاره شده است ابتدا به گسسته تبدیل می‌شوند.

```
def create_classifiers(train):
    """
    This function creates classifiers for each class.
    This is an binary relevance method, so classifiers are
    independent.
    :param train: training data-set
    :return: a dictionary of trained classifiers
    """
```

این تابع دیتاست را می‌گیرد و به تعداد کلاس‌ها در دیتاست، classifier از نوع Naive-Bayes ایجاد می‌کند و با استفاده از تابع create_training_set_for_binary_relevance_classifier از دیتاست اصلی برای همه‌ی کلاسیفایرها دیتاست‌های مناسب را ایجاد می‌کند و سپس آن‌ها را آموزش می‌دهد و در پایان کلاسیفایرهای آموزش داده شده را برای استفاده باز می‌گرداند.

```
def predict(classifiers, testSample):
    """
    Use classifiers to predict labels of input sample.
    :param classifiers: a dictionary of trained classifiers
    :param testSamples: A test sample which user wants
           to predict its labels
    :return: list of labels
    """
```

این تابع کلاسیفایرهای آموزش دیده شده به ازای هر کلاس را به همراه یک ورودی را می‌گیرد و با استفاده از هر کلاسیفایر به طور مستقل محاسبه می‌کند که آیا آن نمونه به کلاس I متعلق است یا خیر. در آخر مجموعه‌ای از برچسب‌ها را برمی‌گرداند که برچسب‌هایی که نمونه به آن‌ها تعلق داشته است مقدارشان ۱ است و سایرین صفر هستند.

```
def BRM_NB_DISCRETE(path):
    """
    This function uses other functions to read
    dataset, train classifiers and evalutes model
    with ten-fold cross validation.
    :param path: path of data-set
    :return: 10-fold accuracy of model
    """
```

این تابع، تابع‌های معرفی شده را فرا می‌خواند (تابع ten-fold-cross validation را در ادامه معرفی می‌کنم)، ابتدا دیتابیس خوانده می‌شود. سپس دیتاست ، تابع ایجاد و آموزش کلاسیفایر و تابع پیشبینی‌کننده‌ی برچسب‌های یک نمونه، به عنوان ورودی به تابع ten_fold_cross_validation_without_scaling داده می‌شوند و در آن تابع، با استفاده از آرگمان‌های ورودی، کلاسیفایرها ایجاد می‌شوند، آموزش داده می‌شوند و سپس تست می‌شوند و این کار برای هر ده fold دوباره انجام می‌شود.

در آخر دقت الگوریتم از تابع ten_fold_cross_validation_without_scaling گرفته می‌شود و به عنوان خروجی بازگردانده می‌شود.

فایل BRM_SVM.py

در این فایل سه تابع موجود است، که برای ایجاد یک Binary Relevance است که کلاسیفایرهای پایه‌ی آن SVM است. در این روش برای حل مسائل Multi-label Classification به ازای هر کلاس یک کلاسیفایر ساخته می‌شود که از ویژگی‌های داده‌ها برای آموزش استفاده می‌کنند و هر کلاسیفایر از سایر کلاسیفایرها مستقل است. این دسته‌بند برای دیتاست‌های پیوسته است.

```
def create_classifiers(train):
    """
    This function creates classifiers for each class.
    This is an binary relevance method, so classifiers are
    independent. SVM is the base classier.

    :param train: training dataset
    :param parents: a dictionary that contains parent of each node
    :param topologicalSort: a topological sort of classifiers
    :return: a dictionary trained classifiers
    """
```

این تابع دیتاست را می‌گیرد و با توجه به تعداد کلاس‌ها در دیتاست، classifier از نوع SVM ایجاد می‌کند و با استفاده از تابع create_training_set_for_binary_relevance_classifier از دیتاست اصلی برای همه‌ی کلاسیفایرها دیتاست مناسب را ایجاد می‌کند و سپس آن‌ها را آموزش می‌دهد و در پایان کلاسیفایرهای آموزش داده شده را برای استفاده باز می‌گرداند.

```
def predict(classifiers, testSample):
    """
    Use classifiers to predict labels of input sample.
    :param classifiers: a dictionary of trained classifiers
    :param testSamples: A test sample which this function
            wants to predict its labels
    :return: list of labels
    """
```

این تابع کلاسیفایرهای آموزش دیده شده به ازای هر کلاس را به همراه یک ورودی را می‌گیرد و با استفاده از هر کلاسیفایر به طور مستقل محاسبه می‌کند که آیا آن نمونه به کلاس I متعلق است یا خیر. در آخر مجموعه‌ای از برچسب‌ها را برمی‌گرداند که برچسب‌هایی که نمونه به آن‌ها تعلق داشته است مقدارشان ۱ است و سایرین صفر هستند.

```
def BRM_SVM(path):
    """
        This function uses other functions to read
        dataset, train classifiers and evalutes model
        with ten-fold cross validation.
        :param path: path of data-set
        :return: 10-fold accuracy of model
    """
```

این تابع، تابع‌های معرفی شده را فرا می‌خواند (تابع ten-fold-cross validation را در ادامه معرفی می‌کنم)، ابتدا دیتابیس خوانده می‌شود. سپس دیتاست، تابع ایجاد و آموزش کلاسیفایر و تابع پیشبینی‌کننده‌ی برچسب‌های یک نمونه به عنوان ورودی به ten_fold_cross_validation_with_scaling داده می‌شوند و در آن تابع، با استفاده از آرگمان‌های ورودی، کلاسیفایرها ایجاد می‌شوند، آموزش داده می‌شوند و سپس تست می‌شوند و این کار برای هر ده fold دوباره انجام می‌شود.

در آخر دقت الگوریتم از تابع ten_fold_cross_validation_with_scaling گرفته می‌شود و به عنوان خروجی بازگردانده می‌شود.

فایل tnbcc_DISCRETE.py

در این فایل سه تابع موجود است، که برای ایجاد یک Bayesian Networks Based-Chain هستند که به کلاسیفایرهای پایه‌ی آن Naive Bayes است. در این روش برای حل مسائل Multi-label Classification به ازای هر کلاس یک کلاسیفایر ساخته می‌شود که از ویژگی‌های داده‌ها و کلاس والد در ساختار وابستگی برای آموزش استفاده می‌کنند و هر کلاسیفایر به کلاسیفایر پدرش وابسته است و از سایرین مستقل است. این دسته‌بند برای دیتاست‌های گسسته است.

```
def create_classifiers(train, parents, topologicalSort):
    """
    This function creates classifiers for each class and link them according to
    dependency directed tree which is built from Chow & Liu's algorithm(MWST)
    :param train: training dataset
    :param parents: a dictionary that contains parent of each node
    :param topologicalSort: a topological sort of classifiers
    :return: a dictionary of trained classifiers
    """
```

این تابع دیتاست، توپولوژی سورت کلاس‌ها در ساختار درخت وابستگی و یک دیکشنری که پدر هر کلاس در آن مشخص شده است را می‌گیرد و با توجه به تعداد کلاس‌ها در دیتاست classifier از نوع Naive Bayes ایجاد می‌کند و با استفاده از تابع create_training_set_for_bayesian_based_chain_classifier از دیتاست اصلی برای همه‌ی کلاسیفایرها دیتاست مناسب را ایجاد می‌کند و سپس آن‌ها را آموزش می‌دهد و در پایان کلاسیفایرهای آموزش داده شده را برای استفاده باز می‌گرداند.

```
def predict(classifiers, parents, topologicalSort, testSample):
    """
    Use classifiers according to their order in topological sort
    to predict labels of test sample.
    :param classifiers: a dictionary of trained classifiers
    :param parents: a dictionary that contains parent of each node
    :param topologicalSort: a topological sort of classifiers
    :param testSamples: A test sample which this function
            wants to predict its labels
    :return: list of labels
    """
```

این تابع کلاسیفایرهای آموزش دیده شده به ازای هر کلاس، دیکشنری که پدر هر گره در آن مشخص شده است، توپولوژی سورت درخت جهت دار وابستگی و یک ورودی را می‌گیرد .سپس از کلاسیفایر ریشه شروع می‌کند بعد از تعیین کلاس توسط کلاسیفایر ریشه به کلاسیفایرهای فرزند ریشه می‌رود و با استفاده از ویژگی‌ها و خروجی

کلاسیفایر پدر، کلاس‌های نمونه‌ی ورودی برای هر کدام از کلاسیفایرهای فرزند ریشه مشخص می‌شود و این کار طبق توپولوژی سورت کلاسیفایرها ادامه پیدا می‌کند تا زمانی که تمام کلاسیفایرها را ببینیم. در آخر مجموعه‌ای از برچسب‌ها را برمی‌گرداند که برچسب‌هایی که نمونه به آن‌ها تعلق داشته است مقدارشان ۱ است و سایرین صفر هستند.

```
def tnbcc_DISCRETE(path):
    """
        This function uses other functions to read
        dataset, train classifiers and evaluate model
        with ten-fold cross validation.
        :param path: path of data-set
        :return: 10-fold accuracy of model
    """
```

این تابع، تابع‌های معرفی شده را فرا می‌خواند (تابع ten-fold-cross validation را در ادامه معرفی می‌کنم)، ابتدا دیتابیس خوانده می‌شود. سپس درخت بدون جهت Chow و Liu ساخته می‌شود و با استفاده از آن درخت جهت دار وابستگی ساخته می‌شود و بعد با استفاده از درخت جهت دار وابستگی توپولوژی سورت درخت جهت دار وابستگی و یک دیکشنری به دست می‌آید که در آن پدر هر کلاس مشخص شده است. سپس دیتاست ، تابع ایجاد و آموزش کلاسیفایر، تابع پیشبینی‌کننده‌ی برچسب‌های یک نمونه، توپولوژی سورت و دیکشنری پدر گره‌ها به عنوان ورودی به ten_fold_cross_validation_without_scaling داده می‌شوند و در آن تابع، با استفاده از آرگمان‌های ورودی کلاسیفایرها ایجاد می‌شوند، آموزش داده می‌شوند و سپس تست می‌شوند و این کار برای هر ده fold دوباره انجام می‌شود.

در آخر دقت الگوریتم از تابع ten_fold_cross_validation_without_scaling گرفته می‌شود و به عنوان خروجی بازگردانده می‌شود.

فایل tsvmcc.py

```
def create_classifiers(train, parents, topologicalSort):
    """
    This function creates classifiers for each class and link them according to
    dependency directed tree which is built from Chow & Liu's algorithm(MWST)
    Base Classifier is SVM
    :param train: training dataset
    :param parents: a dictionary that contains parent of each node
    :param topologicalSort: a topological sort of classifiers
    :return: a dictionary trained classifiers
    """
```

در این فایل سه تابع موجود است، که برای ایجاد یک Bayesian Networks Based-Chain هستند که به کلاسیفایرهای پایه‌ی آن SVM است. در این روش برای حل مسائل Multi-label Classification به ازای هر کلاس یک کلاسیفایر ساخته می‌شود که از ویژگی‌های داده‌ها و کلاس والد در ساختار وابستگی برای آموزش استفاده می‌کنند و هر کلاسیفایر به کلاسیفایر پدرش وابسته است و از سایرین مستقل است. این دسته‌بند برای دیتاست‌های پیوسته است.

این تابع دیتاست، توپولوژی سورت کلاس‌ها در ساختار درخت وابستیگی و یک دیکشنری که پدر هر کلاس در آن مشخص شده است را می‌گیرد و با توجه به تعداد کلاس‌ها در دیتاست، classifier از نوع SVM ایجاد می‌کند و با استفاده از تابع create_training_set_for_bayesian_based_chain_classifier از دیتاست اصلی برای همه‌ی کلاسیفایرها دیتاست مناسب را ایجاد می‌کند و سپس آن‌ها را آموزش می‌دهد و در پایان کلاسیفایرهای آموزش داده شده را برای استفاده باز می‌گرداند.

```
def predict(classifiers, parents, topologicalSort, testSample):
    """
    Use classifiers according to their order in topological sort
    to predict labels of test sample.
    :param classifiers: a dictionary of trained classifiers
    :param parents: a dictionary that contains parent of each node
    :param topologicalSort: a topological sort of classifiers
    :param testSamples: A test sample which this function
           wants to predict its labels
    :return: list of labels
    """
```

این تابع کلاسیفایرهای آموزش دیده شده به ازای هر کلاس، دیکشنری که پدر هر گره در آن مشخص شده است، توپولوژی سورت درخت جهت دار وابستگی و یک ورودی را می‌گیرد. سپس از کلاسیفایر ریشه شروع می‌کند بعد از تعیین کلاس توسط کلاسیفایر ریشه به کلاسیفایرهای فرزند ریشه می‌رود و با استفاده از ویژگی‌ها و خروجی کلاسیفایر پدر، کلاس نمونه برای هر کدام از کلاسیفایرهای فرزند ریشه مشخص می‌شود و این کار طبق توپولوژی سورت کلاسیفایرها ادامه پیدا می‌کند تا زمانی که تمام کلاسیفایرها را ببینیم. در آخر مجموعه‌ای از برچسب‌ها را برمی‌گرداند که برچسب‌هایی که نمونه به آن‌ها تعلق داشته است مقدارشان ۱ است و سایرین صفر هستند.

```
def tsvmcc(path):
    """
        This function uses other functions to read
        dataset, train classifiers and evaluate model
        with ten-fold cross validation.
        :param path: path of data-set
        :return: 10-fold accuracy of model
    """
```

این تابع، تابع‌های معرفی شده را فرا می‌خواند (تابع ten-fold-cross validation را در ادامه معرفی می‌کنم)، ابتدا دیتابیس خوانده می‌شود. سپس درخت بدون جهت Chow و Liu ساخته می‌شود و با استفاده از آن درخت جهت دارد وابستگی ساخته می‌شود و بعد با استفاده از درخت جهت دار وابستگی توپولوژی سورت درخت جهت دار وابستگی و یک دیکشنری به دست می‌آید که در آن پدر هر کلاس مشخص شده است. سپس دیتاست ، تابع ایجاد و آموزش کلاسیفایر، تابع پیشبینی‌کننده‌ی برچسب‌های یک نمونه، توپولوژی سورت و دیکشنری پدر گره‌ها به عنوان ورودی به

ten_fold_cross_validation_with_scaling داده می‌شوند و در آن تابع، با استفاده از آرگمان‌های ورودی کلاسیفایرها ایجاد می‌شوند، آموزش داده می‌شوند و سپس تست می‌شوند و این کار برای هر ده fold دوباره انجام می‌شود.

در آخر دقت الگوریتم از تابع ten_fold_cross_validation_with_scaling گرفته می‌شود و به عنوان خروجی بازگردانده می‌شود.

فایل tnbcc.py و BRM_GNB.py

در مقاله مدل‌هایی کلاسیفایر پایه‌ی آن‌ها Naive Bayes است برای دیتاست‌های گسسته به کار رفته‌اند که در قسمت‌های قبلی آن‌ها را معرفی کردم و به پیاده سازی آن‌ها پرداختم. برای مشاهده‌ی شخصی، دو کلاسیفایر در فایل‌های BRM_GNB.py ، tnbcc.py را برای دیتاست‌های پیوسته نوشتم که از Gaussian NB استفاده می‌کنند ولی از آن‌جایی که بخشی از مقاله نیستند به آن‌ها اشاره نمی‌کنم.

فایل accuracy_of_model.py

در این فایل سه تابع موجود است که برای سنجیدن دقت مدل‌ها هستند.

```
def mean_accuracy(predictions, dataset):
    """
    This functions Calculates mean accuracy of model
    which can be a binary relevance model or bayesian
    based chain.
    :param predictions for test samples:
    :param dataset: the dataset which its samples were used
    :return: accuracy
    """
```

این تابع یک دیتاست و کلاس‌هایی که به ازای هر نمونه از دیتاسب برای هر نمونه پیشبینی شده است را می‌گیرد و با استفاده از تابع زیر دقت میانگین Multi-label Model را محاسبه می‌کند:

1. *Mean accuracy* over the d class variables (accuracy per label):

$$\mathrm{M-Acc} = \frac{1}{d}\sum_{j=1}^{d} Acc_j = \frac{1}{d}\sum_{j=1}^{d}\frac{1}{N}\sum_{i=1}^{N}\delta(c'_{ij}, c_{ij}) \qquad (6)$$

where $\delta(c'_{ij}, c_{ij}) = 1$ if $c'_{ij} = c_{ij}$ and 0 otherwise, and c'_{ij} denotes the C_j class value outputted by the model for instance i and c_{ij} is its true value.

```
def ten_fold_cross_validation_without_scaling(learningFunction, predictFuncion, dataSet, additionalArg):
    """
    This function uses 10-fold-cv to evaluate learningFunction
    which can be SVM,NaiveBayes, and any other learning algorithm.
    :param learningFunction: Is a function that learns
           from data to predict label of new inputs.
           It can be any Machine Learning algorithms
           like TNBCC, TSVMCC and ...
    :param predictFuncion: Is a function that predicts labels of data
           according to the function which learned
           from data.
    :param argumentOFLearningFunction: is a list
        that contains necessary argument for
        learningFunction and predictFuncion.
    :param dataSet: training data
    :return: return average 10-fold cv error
    """
```

این تابع به عنوان ورودی آرگمان‌های زیر را می‌گیرد:

- تابع یادگیری که می‌تواند TNBCC, TSVMCC, Binary Relevance TN و Binary Relevance SVM باشد.
- تابع پیشبینی برچسب‌های یک نمونه ورودی
- دیتاست
- آرگمان‌های ضروری برای تابع یادگیری و پیشبینی. اگر تابع یادگیری از نوع Binary Relevance باشد برابر با None است ولی اگر از نوع Bayesian Based Chain باشد شامل توپولوژی سورت و لیست والد هر کلاس در ساختار جهت دار وابستگی است.

این تابع ابتدا دیتاست را به ده قسمت تبدیل می‌کند و هر بار یک قسمت را تست در نظر می‌گیرد و سایر قسمت‌ها را برای آموزش استفاده می‌کند در هر بار از ده بار، با استفاده از تابع‌ها و داده‌هایی که به عنوان ورودی گرفته است ابتدا کلاسفایرها ساخته و آموزش داده می‌شوند و سپس برچسب‌های نمونه‌های تست پیشبینی می‌شوند و بعد دقت

الگوریتم محاسبه می‌شود. در آخر دقت میانگین مدل برای ده fold باز گردانده می‌شود.

```
def ten_fold_cross_validation_with_scaling(learningFunction, predictFuncion, dataSet,additionalArg):
    """
    This function uses 10-fold-cv to evaluate learningFunction
    which can be SVM,NaiveBayes, and any other learning algorithm.
    :param learningFunction: Is a function that learns
           from data to predict label of new inputs.
           It can be any Machine Learning algorithms
           like TNBCC, TSVMCC, KNN, decisionTree, SVM and ...
    :param predictFuncion: Is a function that predict labels of data
           according to the function which learned
           from data.
    :param argumentOFLearningFunction: is a list
       that contains necessary argument of
       learningFunction.
    :param dataSet: training data
    :return: return average 10-fold cv accuracy
    """
```

این تابع مانند تابع قبل است با این تفاوت که داده‌ها scale می‌شوند.

فایل runCode.py

در این فایل دو الگوریتم TNBCC-Discrete و Binary-Relevance-Naive-Bayes را برای دیتاست‌های گسسته‌ی medical و enron فراخوانده می‌شوند.

همین طور دو الگوریتم TSVMCC و Binary-Relevance-SVM برای دیتاست‌های پیوسته‌ی scene، emotions و yeast فراخوانده می‌شوند.

بعد از محاسبه‌ی دقت مدل‌ها برای دیتاست‌ها، در پایان دقت هر کدام از الگوریتم‌ها برای دیتاست‌های مربوط گزارش می‌شود. که در بخش بعد به نتیجه‌های حاصل از اجرای برنامه می‌پردازیم.

۳-نتیجه‌گیری

در بخش اول خلاصه‌ای از مقاله توضیح داده‌شد و در بخش دوم به پیاده‌سازی پرداختیم. در این بخش نتایج حاصل از اجرای مدل‌های مختلف، که آن‌ها را پیاده‌سازی کرده‌ام را بررسی می‌کنیم.

در مقاله الگوریتم اصلی‌ای که کامل توضیح داده شده است، الگوریتم Tree Naive Bayes Chain Classifier است که جزییات آن به خوبی شرح داده شده است و نتایج به دست‌آمده توسط پیاده سازی‌های من با نتایج مقاله در این مورد کمتر از 1 درصد اختلاف دارد که آن نیز به دلیل استفاده از ten-fold-cv و انتخاب تصادفی ریشه در درخت وابستگی است و می‌توان گفت **نتایج پیاده سازی انجام شده با مقاله برای مدل‌های Naive-Base کاملا یکسان است**.

در مقاله مشخص نشده است Binary Relevance مدلی که استفاده شده است کلاسیفایر پایه‌ی آن SVM است یا Naive-Bayes به همین دلیل هر دو را پیاده‌سازی کردم که از Binary-Relevance-Naive-Bayes برای دیتاست‌های گسسته و از Binary-Relevance-SVM برای دیتاست‌های پیوسته استفاده کرده‌ام.

الگوریتم Tree Naive Bayes Chain Classifier و Binary Relevance-Naive Bayes در مقاله برای دیتاست‌های گسسته مورد استفاده قرار گرفته است و در صورتی که دیتاست پیوسته بوده است با استفاده از الگوریتم:

static, global, supervised and top-down discretization algorithm (Cheng-Jung , 2008)

دیتاست گسسته‌سازی شده است. از آنجایی که این الگوریتم برای زبان پایتون در دسترس نبود و پارامترهای استفاده شده برای آن در مقاله ذکر نشده است و در دیتاست‌های مورد استفاده **در مقاله دیتاست‌های گسسته موجود است**، برای ارزیابی آن دو مدل از دیتاست‌های گسسته استفاده کردم و دیتاست‌های پیوسته را به گسسته تبدیل نکردم زیرا برای درستی‌سنجی پیاده‌سازی، دیتاست‌های گسسته‌ی استفاده شده در مقاله کافی است و نیازی به دیتاست‌های بیشتر نیست.

الگوریتم Tree SVM Chain Classifier و Binary Tree Relevance-SVM در مقاله برای دیتاست‌های پیوسته و گسسته مورد استفاده قرار گرفته اند. در مقاله به این الگوریتم‌ها که کلاسیفایرهای پایه‌ی آن‌ها از نوع SVM است کم

تر پرداخته شده است و جزییات دقیقی از نحوه‌ی پیاده سازی آن‌ها بیان نشده است به همین دلیل نتایج به دست آمده مقدار بسیار اندکی (معمولا یک تا دو درصد) با نتایج مقاله متفاوت است که با توجه به شافل کردن دیتاست و استفاده از ten-fold-cross-validation و انتخاب تصادفی ریشه‌ی درخت وابستگی کاملا طبیعی است. ولی در روش‌هایی که کلاسیفایر پایه‌ی آن‌ها Naive-Bayes است به دلیل اینکه جزییات کافی ارایه شده است دقت نتایج به دست آمده با مقاله یکسان است و آن اندک اختلاف را هم ندارند.

در زیر هفت اجرای مختلف برنامه و خروجی‌های تولید شده به ازای هر بار اجرا را مشاهده می‌کنید که بعد از آن‌ها، نتایج را در جدولی همراه با نتایج مقاله، در ادامه ارائه داده‌ام. هر بار اجرای برنامه و گرفتن نتیجه‌های الگوریتم‌های مختلف به ازای دیتاست‌های مختلف، زمانی در حدود یک ساعت و نیم تا دو ساعت نیاز داشت.

اجرای شماره‌ی یک:

```
==========================================
Dataset Name: medical
{'tnbbcc_DISCRETE': 0.9744855724649536, 'BRM_NB_DISCRETE': 0.9744663721158565}
Dataset Name: enron
{'tnbbcc_DISCRETE': 0.7880729422089148, 'BRM_NB_DISCRETE': 0.7819993289110287}
==========================================
Dataset Name: scene
{'BRM_SVM': 0.9259069950517318, 'tsvmbcc': 0.925974471434998}
Dataset Name: emotions
{'BRM_SVM': 0.8220065609622743, 'tsvmbcc': 0.8202569710224166}
Dataset Name: yeast
{'BRM_SVM': 0.8138763934833737, 'tsvmbcc': 0.8126638478306594}

Process finished with exit code 0
```

اجرای شماره‌ی دو:

```
=========================================
Dataset Name: medical

{'tnbbcc_DISCRETE': 0.9749522718594882, 'BRM_NB_DISCRETE': 0.9745139366170292}

Dataset Name: enron

{'tnbbcc_DISCRETE': 0.7870073303564513, 'BRM_NB_DISCRETE': 0.7846564540691219}

=========================================
Dataset Name: scene

{'BRM_SVM': 0.925295771479982, 'tsvmbcc': 0.9271353463787675}

Dataset Name: emotions

{'BRM_SVM': 0.8237834882449426, 'tsvmbcc': 0.822316384180791}

Dataset Name: yeast

{'BRM_SVM': 0.8137516253322371, 'tsvmbcc': 0.8134510583781097}

Process finished with exit code 0
```

اجرای شماره‌ی سه:

```
=========================================
Dataset Name: medical

{'tnbbcc_DISCRETE': 0.974388916162112, 'BRM_NB_DISCRETE': 0.9746972126765939}

Dataset Name: enron

{'tnbbcc_DISCRETE': 0.7834216503626459, 'BRM_NB_DISCRETE': 0.7839083189221275}

=========================================
Dataset Name: scene

{'BRM_SVM': 0.9246747076023393, 'tsvmbcc': 0.9268713450292397}

Dataset Name: emotions

{'BRM_SVM': 0.8232823036267541, 'tsvmbcc': 0.8197330052852194}

Dataset Name: yeast

{'BRM_SVM': 0.8139607673480315, 'tsvmbcc': 0.8130411400271527}

Process finished with exit code 0
```

اجرای شماره‌ی چهار:

```
==========================================
Dataset Name: medical

{'tnbbcc_DISCRETE': 0.9744945180821467, 'BRM_NB_DISCRETE': 0.9747146675394097}

Dataset Name: enron

{'tnbbcc_DISCRETE': 0.787339519397052, 'BRM_NB_DISCRETE': 0.7837679064605216}

==========================================
Dataset Name: scene

{'BRM_SVM': 0.9255774853801169, 'tsvmbcc': 0.9256508659469185}

Dataset Name: emotions

{'BRM_SVM': 0.8198423546564607, 'tsvmbcc': 0.8243757973391652}

Dataset Name: yeast

{'BRM_SVM': 0.8143532372793849, 'tsvmbcc': 0.8135383004761266}
```

اجرای شماره‌ی پنج:

```
==========================================
Dataset Name: medical

{'tnbbcc_DISCRETE': 0.9744342988054328, 'BRM_NB_DISCRETE': 0.9747181585119726}

Dataset Name: enron

{'tnbbcc_DISCRETE': 0.7873515215651856, 'BRM_NB_DISCRETE': 0.7833053712928785}

==========================================
Dataset Name: scene

{'BRM_SVM': 0.9249943769680611, 'tsvmbcc': 0.9253317588843905}

Dataset Name: emotions

{'BRM_SVM': 0.8196692181519957, 'tsvmbcc': 0.8199881538181154}

Dataset Name: yeast

{'BRM_SVM': 0.8145940493718568, 'tsvmbcc': 0.8124759785456146}
```

اجرای شماره‌ی شش:

```
========================================
Dataset Name: medical

{'tnbbcc_DISCRETE': 0.9745187367043036, 'BRM_NB_DISCRETE': 0.9746459390170727}

Dataset Name: enron

{'tnbbcc_DISCRETE': 0.7894881656041088, 'BRM_NB_DISCRETE': 0.7840418914384533}

========================================
Dataset Name: scene

{'BRM_SVM': 0.9251824673864147, 'tsvmbcc': 0.9263990103463786}

Dataset Name: emotions

{'BRM_SVM': 0.8201658465463824, 'tsvmbcc': 0.8183342445780937}

Dataset Name: yeast

{'BRM_SVM': 0.8133454117827027, 'tsvmbcc': 0.8135366273400002}
```

اجرای شماره‌ی هفت:

```
========================================
Dataset Name: medical

{'tnbbcc_DISCRETE': 0.9746723394970814, 'BRM_NB_DISCRETE': 0.9749313260241091}

Dataset Name: enron

{'tnbbcc_DISCRETE': 0.7880418398162247, 'BRM_NB_DISCRETE': 0.783367188911545}

========================================
Dataset Name: scene

{'BRM_SVM': 0.9251352339181287, 'tsvmbcc': 0.9266649797570852}

Dataset Name: emotions

{'BRM_SVM': 0.8221295790049208, 'tsvmbcc': 0.8216147257153272}

Dataset Name: yeast

{'BRM_SVM': 0.8141395538941048, 'tsvmbcc': 0.8128833871923818}
```

در زیر میانگین هفت اجرای مختلف را در جدول‌های زیر مشاهده می‌کنید. داده‌هایی که با * مشخص شده اند، نتایج مقاله هستند و آن‌هایی که ستاره ندارند نتایج من هستند. نتیجه‌ی بهتر برای هر دیتاست به صورت Bold نشان داده شده است.

Discrete Data-sets	Binary-Relevance Naive Bayes (BRM-TN)	Tree Naive Base Chain Classifiers (TNBCC)
Data-set: Enron	0.783577571 0.7829*	**0.787245571** 0.7984*
Data-set: Medical	**0.974669659** 0.9746 *	0.974563808 0.9756 *

در مقاله به این نکته اشاره شده است که در دیتاست Medical معیار correlation بین کلاس‌هاتقریبا وجود ندارد به همین دلیل استفاده از Chain Classifier ممکن است نتایج بهتری ایجاد نکند. که در نتایج به دست آمده آن را مشاهده می‌کنید.

Continuous Data-sets	Binary-Relevance SVM (BRM-SVM)	Tree SVM Chain Classifiers (TSVMCC)
Data-set: Scene	0.925251857 0.9392*	**0.933184429** 0.929*
Data-set: Emotions	0.821571686 0.8423*	**0.821808** 0.843*
Data-set: Yeast	0.814016857 0.8646*	**0.814157143** 0.856*

همین طـور کـه در دو جـدول بـالا مشـاهده می‌شـود، روش‌هـای Binary-Relevance رابطه‌ی بین‌کلاس‌ها را در نظر نمی‌گیرند و هر کلاس را به صورت مستقل از سایر کلاس‌ها دسته‌بندی می‌کنند عملکردشان ضعیف‌تر است. در حـالی که روش‌های Tree Naive-Base Chain Classifier و Tree SVM Chain Classifier رابطه‌ی بین کلاس‌ها را در نظر می‌گیرند. به همین دلیل این الگوریتم‌ها نتایج بهتری می‌گیرند.

دو روش TSVMCC و TNBCC دو مورد از ابتدایی‌ترین روش‌های ممکن برای پیاده‌سازی یک دسته‌بند زنجیره‌ای بر اساس شبکه‌های بیزین هستند که به راحتی بسیاری از روش‌های مورد استفاده را شکست می‌دهند. در کارهای آتـی احتمالی به ترکیب‌های پیچیده‌تری از این دسته‌بندها خواهم پرداخت.