تمرین های سری دوم
درس یادگیری ماشین

فرهاد دلیرانی
۹۶۱۳۱۱۲۵

از آنجایی که در شرح تمرین‌ها برای پایتون ورژن خاصی ذکر نشده است تمام کدها را با **پایتون سه‌وشش** نوشته ام و همیـن طـور از numpy و matplotlib ب رای رس م نم ودار اس تفاده‌کرده‌ام. ب رای س ادگی در نص ب آن پکیج‌ه ا از anaconda3 استفاده کرده‌ام.

بخش اول سوال ۱-

KNN از curse of dimensionality رنج می‌برد بخصوص با وجود معیار فاصله‌ی اقلیدسی که با افزایش ابعاد بی‌معنی‌تر می‌شود و ضعیف‌تر عمل می‌کند.

بر اساس قضیه‌ی Vapnik–Chervonenkis theory و Hoefdding با زیاد شدن ابعاد فضای Hypothesis set بزرگ تر می‌شود و برای اینکه پیشبینی که می‌کنیم برای داده‌های که تا حالا ندیده‌ایم یک پیشبینی خوب باشد و E_{out} کم باشد باید میزان داده‌هارا زیاد کنیم.

$$E_{\text{out}}(g) \leq E_{\text{in}}(g) + \sqrt{\frac{8}{N} \ln \left(\frac{4((2N)^{d_{\text{vc}}} + 1)}{\delta} \right)} \,. \qquad (2.14)$$

در تصویر بالا بر اساس قضیه‌هایی که اشاره شد باندی برای خطای داده‌هایی که ندیده ایم تعیین شده است. که d_{VC} همان VC Dimension است و هر چه بیشتر باشد یعنی بزرگی Hypotesis set بیشتر است و برای خنثا کردن اثر آن باید مقدار N خارج از لگاریتم که همان تعداد داده‌ها است را باید بیشتر کنیم.

بخش اول سوال ۲-

در الگوریتم KNN هر چه میزان K کمتر باشد variance بیشتر است و با افزایش k میزان variance کم می‌شود.

در الگوریتم KNN هر چه میزان K زیاد تر باشد bias بیشتر است و هر چه K کم می‌شود میزان بایاس هم کمتر می شود.

هر چه میزان k کمتر باشد ناحیه‌ای که KNN برای پیشبینی کلاس یک نمونه استفاده می‌کند محدودتر می‌شود و همین طور classifier بیشتر تحت فشار قرار می‌گیرد تا توزیع کلی دیتا را نادیده بگیرد. به همین دلیل واریانس بیشتر می شود و بایاس کمتر. همین طور از این جهت می شود دید که هر چه k کوچک تر باشد مرز جداکننده شارپ تر می‌شود در نتیجه واریانس بیشتر می شود.

با زیاد شدن K داده‌های بیشتری برای پیش بینی برچست یک نمونه استفاده می‌شوند و همین طور مرز تصمیم smooth تر می‌شود در نتیجه بایاس زیاد می‌شود و از واریانس کاسته می‌شود.

بخش اول سوال ۳-

الگوریتم KNN الگوریتمی غیر پارامتری است زیرا با زیاد شدن داده‌ها میزان پارامترهایی که باید مد نظر گرفت بیشتر می‌شود الگوریتم از نظر محاسباتی پیچیده تر می‌شود و همین طور کند تر می‌شود. در حالی که یک الگوریتم پارامتری از تعداد محدودی پارامتر استفاده می‌کند و سریع تر است و از نظر محاسباتی پیچیدگی کمتر دارد و با زیاد

شدن داده ها میزان پارامتر ها بیشتر نمی شود.

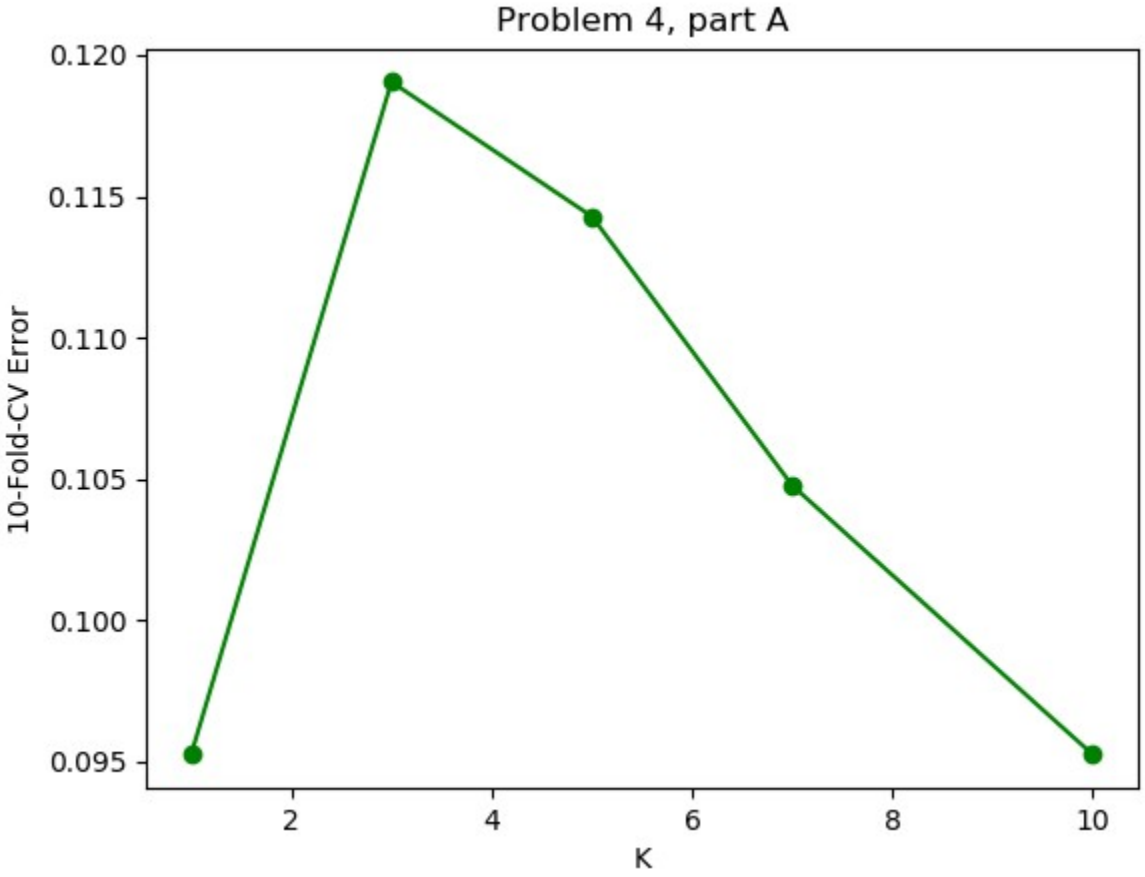
بخش اول سوال ۴-

کدهای این سوال در فایل problem4.py قرار دارد. برای این سوال تابع‌های زیر را نوشته ام:

```
euclidean_distance
manhattan_distance
minkowski_distance
minkowski_distance_p4
minkowski_distance_p_half
cosine_distance
KNN
ten_fold_cross_validation
```

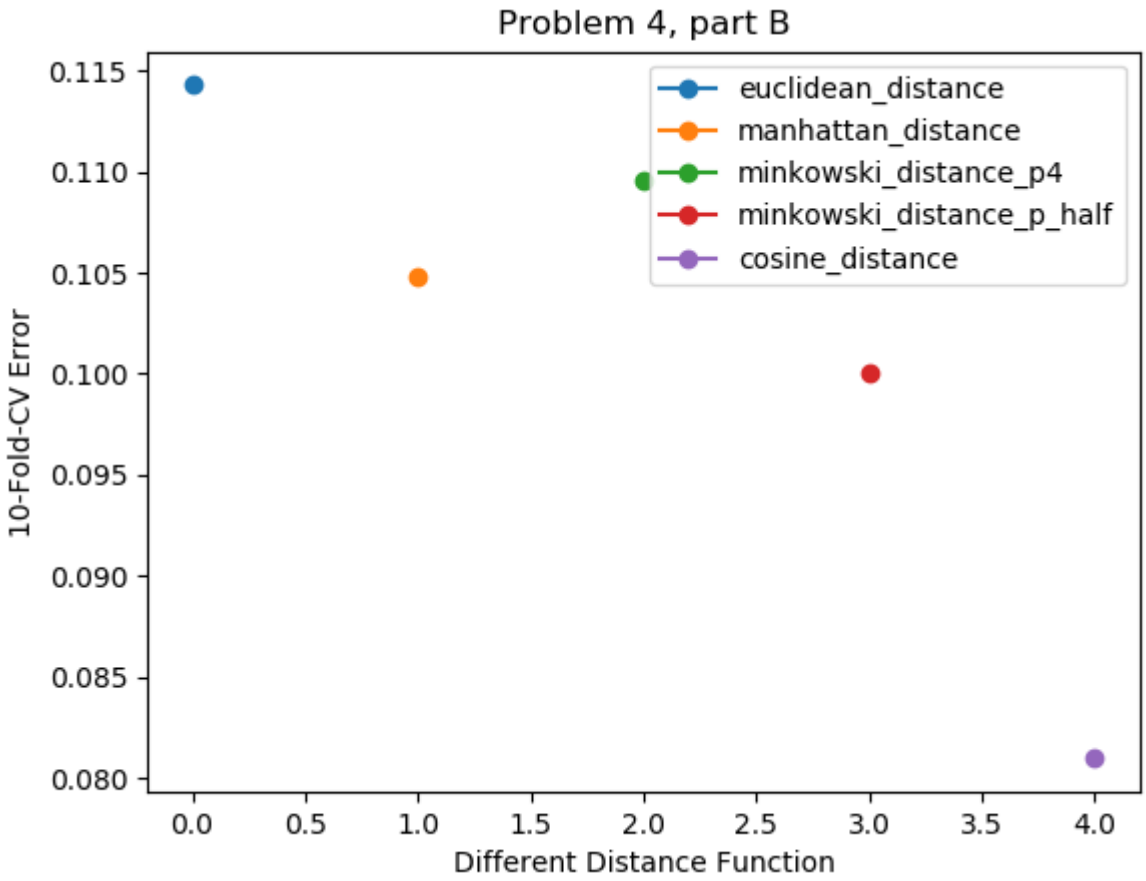
بخش a)

از آنجا که شافل کردن دیتاست به صورت رندوم صورت می‌گیرد در هر با اجرای کد خروجی‌های متفاوتی ایجاد می‌شود. در زیر یک نمونه از خروجی را مشاهده می‌کنید. (**خطا از یک است و به درصد تبدیل نشده است**)



```
=====================================
Part A:
K=1,    Error=0.09523809523809522
K=3,    Error=0.11904761904761903
K=5,    Error=0.11428571428571428
K=7,    Error=0.10476190476190475
K=10,   Error=0.09523809523809522
```

بخش b)




```
Part B:
Distance Function=euclidean_distance,   Error=0.11428571428571428
Distance Function=manhattan_distance,   Error=0.10476190476190475
Distance Function=minkowski_distance_p4,    Error=0.10952380952380951
Distance Function=minkowski_distance_p_half,    Error=0.1
Distance Function=cosine_distance,  Error=0.08095238095238096
====================================
```

همین طور که در تصویر می بینید فاصله‌ی اقلیدوسی بدترین عملکرد را داشته و فاصله‌ی کسینوسی بهـترین عملکـرد را داشته است.

فاصله‌ی Euclidean بدترین عملکرد را داشته است زیر این روش در سنجیدن فاصله در دیتاست‌هایی که بی ن دیتاه ا

correlation وجود دارد و یا اسکیل ابعاد مختلف متفاوت است، زیاد خوب عمل نمی‌کند.

فاصله‌ی منهتن بر اساس مقاله‌ی زیر

On the Surprising Behavior of Distance Metrics in High Dimensional Space Charu C. Aggarwal1, Alexander Hinneburg2, and Daniel A. Keim2

در داده‌ها با ابعاد بالا بهتر از فاصله‌ی Euclidean عمل می‌کند .یکی از دلایل آن این است که در فاصله‌ی اقلیدسی توان دو برای محاسبه‌ی فاصله استفاده می‌شود و در فاصله‌ی منهتن چنین چیزی نیست.

Cosine Distance از باقی روش‌ها بهتر عمل می‌کند. در فضای وکتوری (Vector Space) و با ابعاد زیاد فاصله‌ی کسینوسی فقط از ضرب داخلی و طول هر بردار استفاده می‌کند در نتیجه فاصله فقط تحت تاثیر چیزهایی قرار می‌گیرد که بین دو وکتور مشترک است. در این روش اندازه‌گیری، فاصله‌ی دو بردار در یک راستا نسبت به هم کم است و هر چه راستای دو وکتور از هم فاصله‌ی می‌گیرد فاصله‌ی کسینوسی بیشتر می شود.وکتورهای اعضای یک کلاس نسبت به هم زاویه‌ی کمتر دارند و بیشتر در راستای هم هستند.

بخش اول سوال ۵-

بخش a)

KNN ساده و قابل درک است و همین طور scalable ، و برای هر نوع دیتاساختاری مناسب است. در این الگوریتم لازم نیست مدلی از نمونه‌ها بسازیم به همین دلیل هم نگهداری و دسترسی به آن آسان است. همین طور این الگوریتم جنبه‌های بدی هم دارد مانند هزینه‌ی محاسبانی زیاد و سرعت کم. هر بار که نمونه‌ای جدید وارد می‌شود نیاز است که با تمام داده‌ها چک شود. همین طور دقت آن پایدار نیست و یکی از دلایل این موضوع استفاده از فاصله‌ی اقلیدیسی است که در دیتاست‌ها با ابعاد زیاد و correlation نمی‌تواند به خوبی فاصله را نشان دهد و معیار خوبی نیست.

بخش b)

برای رفع مشکل‌های KNN روش‌های مختلفی سال‌های اخیر پیشنهاد شده است که می‌توان به موارد زیر اشاره کرد: استفاده از فاصله‌ی mahalanobis به جای فاصله‌ی اقلیدسی، فاصله‌ی mahalanobis ابعاد مختلف داده‌ها را scale می‌کند و correlation را از بین می‌برد که به آن LMNN می‌گویند.استفاده از نگاشت غیر خطی ویژگی‌ها که بر اساس شبکه‌های عصبی عمیق است که به آن Dnet-kNN می‌گویند. یا استفاده از روش WDkNN که بر اساس تابع وزن‌داده شده‌ی شباهت است.

بخش c)

در KNN معمولی کلاس یک نمونه‌ی ورودی بر حسب k همسایه‌ی آن به صورت زیر تعیین می‌شود:

$$y_t' = \arg\max_{c\in\{c_1,c_2\}} \sum_{x_i\in\phi(x_t)} I(y_i = c)$$

$$= \max\left\{ \sum_{x_i\in(x_t)} I(y_i = c_1), \sum_{x_i\in(x_t)} I(y_i = c_2) \right\}$$

که برابر است با بیشترین کلاسی که در k همسایه‌ی نمونه‌ی ورودی قرار دارد. ولی در WdkNN به همسایه‌های مختلف وزن‌های مختلفی نسبت داده می‌شود، کلاس یک نمونه‌ی ورودی بر حسب k همسایه‌ی آن به صورت زیر تعیین می‌شود:

$$y_t' = \arg\max_{c\in\{c_1,c_2\}} \sum_{x_i\in\phi(x_t)} I(y_i = c)\frac{1}{d(x_t,x_i)}$$

که همان روش KNN معمولی است با این تفاوت که به هر همسایه وزنی برابر با معکوس فاصله‌ی آن همسایه تا

نقطه‌ی ورودی مورد نظر داده شده است. در این روش دقت در دیتاست‌های غیربالاس بیشتر است. پیاده سازی این الگوریتم در فایل problem5.py موجود است.

بخش d)

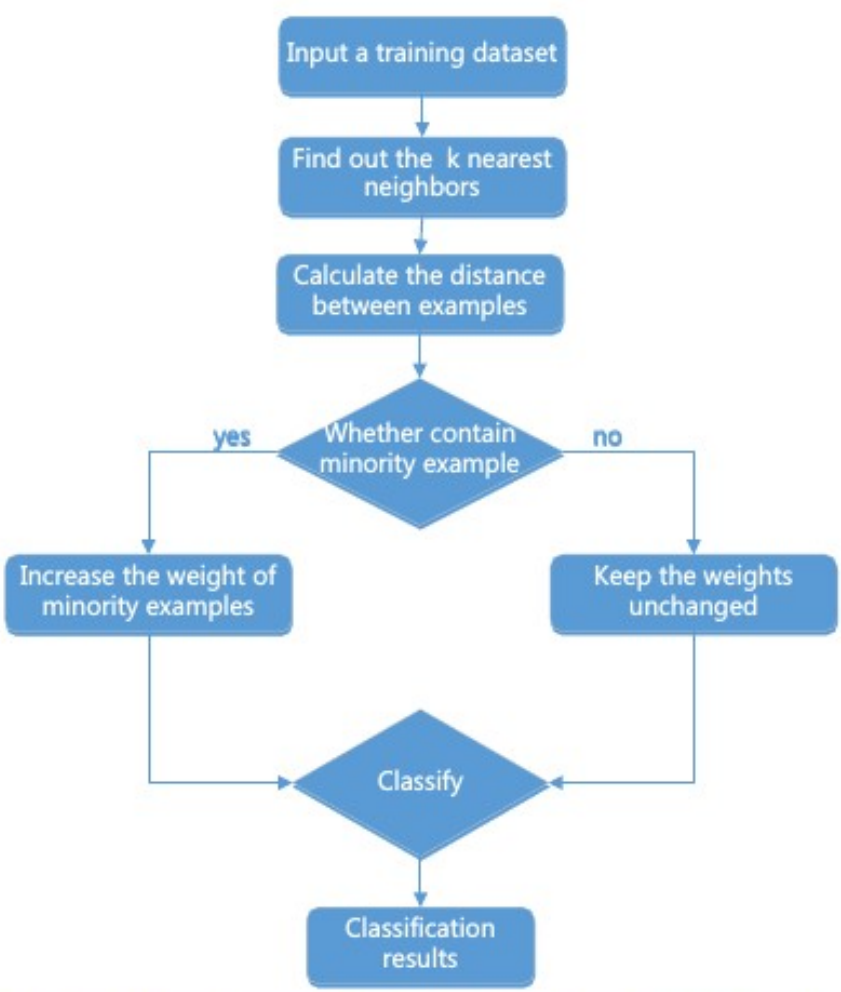


Fig.3. The flow chart of KNN algorithm based on the minority class

هسته‌ی اصلی روش پیشنهادی مقاله، افزایش وزن کلاس اقلیت است. این الگوریتم ابتدا K همسایه‌ی نزدیک نمونه‌ی ورودی را حساب می‌کند. فاصله‌ی آن‌ها را برای وزن دادن تا نمونه‌ی ورودی حساب می‌کند. در صورت وجود نداشتن نمونه‌هایی از کلاس اقلیت کلاس نمونه‌ی ورودی را مشخص می‌کند ولی اگر از کلاس اقلیت نمونه‌هایی وجود داشته باشند k همسایه‌ی نزدیک به آن را پیدا می‌کند و براساس تعداد نمونه‌های کلاس اکثریت در همسایگی آن به آن وزن مشخصی اختصاص می‌دهد و در پایان بر اساس وزن نمونه‌های اقلیت و اکثریت کلاس نمونه‌ی ورودی را مشخص می‌کند.

وزن نمونه‌های کلاس اقلیت اینگونه تعیین می شود:

$$L_{\min} = \frac{\left(N_{maj}\right)^{\alpha}}{K}$$

ولی از آنجایی که مقدار آن ممکن است صفر شود از این تابع برای مجاسبه‌ی وزن استفاده می‌شود:

$$w(L) = (L_{\min} + 1)\lambda$$

و در آخر کلا س نمونه‌ی ورودی براساس همسایگانش اینگونه محاسبه می‌شود:

$$y_t' = \arg\max_{c \in \{c_1, c_2\}} \sum_{x_i \in \phi(x_t)} I(y_i = c) \frac{1}{d(x_t, x_i)} w(L)$$

کد تابع نوشته شده‌ی این روش پیشنهادی مقاله در فایل problem5 موجود است.

بخش e)

در فایل problem5.py تابع‌های زیر را برای پیاده سازی مقاله نوشته ام:

```
euclidean_distance
WDKNN
major_neighbour_of_a_minor
proposed_KNN
five_fold_cross_validation
```

نتایج اجرای WDKNN و proposed_KNN را با k=3 بر روی ۷ دیتاست اول مقاله را در زیر مشاهده می‌کنید:

```
/home/bat/anaconda3/bin/python /home/bat/Dropbox/codes/pythonCode/mach
It takes few seconds please wait, ...

====================================
WDKNN  :
F-measure - G-mean
('yeast3', {'F-measure': 0.7252, 'G-mean': 0.8057})
('ecoli3', {'F-measure': 0.5824, 'G-mean': 0.7345})
('yeast-2_vs_4', {'F-measure': 0.7756, 'G-mean': 0.831})
('yeast-0-3-5-9_vs_7-8', {'F-measure': 0.3571, 'G-mean': 0.5106})
('yeast-0-2-5-6_vs_3-7-8-9', {'F-measure': 0.6146, 'G-mean': 0.7193})
('yeast-0-2-5-7-9_vs_3-6-8', {'F-measure': 0.8135, 'G-mean': 0.8753})
('ecoli-0-2-6-7_vs_3-5', {'F-measure': 0.6819, 'G-mean': 0.7526})
====================================
ImprovedKNN  :
F-measure - G-mean
('yeast3', {'F-measure': 0.7545, 'G-mean': 0.8465})
('ecoli3', {'F-measure': 0.5807, 'G-mean': 0.7589})
('yeast-2_vs_4', {'F-measure': 0.7794, 'G-mean': 0.8386})
('yeast-0-3-5-9_vs_7-8', {'F-measure': 0.4556, 'G-mean': 0.6419})
('yeast-0-2-5-6_vs_3-7-8-9', {'F-measure': 0.624, 'G-mean': 0.7583})
('yeast-0-2-5-7-9_vs_3-6-8', {'F-measure': 0.8091, 'G-mean': 0.8835})
('ecoli-0-2-6-7_vs_3-5', {'F-measure': 0.7037, 'G-mean': 0.7719})
====================================
```

بخش دوم سوال ۱:

درخت تصمیم تمام داده‌های آموزش را در ریشه قرار می‌دهد و سپس یک ویژگی را انتخاب می‌کند و بر اساس آن یک سری فرزند برای ریشه ایجاد می‌شود و داده‌های آموزش را بر اساس آن ویژگی به فرزند متناظر منتقل می‌کند و این کار برای هر فرزند به همین ترتیب صورت می‌گیرد تا زمانی که در یک مسیر از ریشه تا برگ تمام فیچرها استفاده شده باشند و یا تمام داده‌های آموزش در برگ دارای یک کلاس باشند. به همین دلیل درخت تصمیم در حالت عادی تا زمانی که داده‌های یک برگ تماما از یک کلاس نباشند مرتب اقدام افزایش عمق درخت و تقسیم هر گره از درخت بر اساس فیچر می‌کند. به همین دلیل تا زمانی که تمام داده های آموزش را درست دسته بندی نکند و خطای آموزش را تا حد ممکن کاهش ندهد از کار نمی ایستند. و مرز تصمیم آن دارای شکستگی ها و خط‌های بسیار زیادی می شود و به اصطلاح smooth نیست. به همین دلایل درخت تصمیم دچار بیش برازش می شود.

بخش دوم سوال ۲:

برای جلوگیری از بیش برازش و واریانس بالای درخت تصمیم به آن اجازه می‌دهند روی داده‌ها آموزش ببینند و دچار بیش برازش شود و بعد با هرس کردن گره‌های آن به طوری که خطای روی مجموعه‌ی تست بدتر نشود واریانس و بیش برازش آن را از بین می‌برند در این روش یک گره انتخاب می‌شود و حذف می‌شود و در صورتی که حذف کردند باعث بهبود خطای تست شود گره حذف می‌شود در غیر این صورت گره بازگردانده می‌شود. نحوه‌های دیگر هرس کردن هم وجود دارد مانند هرس کردن در حین آموزش درخت که با استفاده از تابع‌های ابتکاری گره‌هایی از درخت انتخاب می‌شوند که گسترش پیدا نکنند.

بخش دوم سوال ۳:

ترکیب تعدادی مدل یادگیری دقت و پایداری Classification را زیاد می‌کند. همچنین با استفاده از تکنیک Bagging یا گرفتن میانگین از مدل‌های نویز دار و unbiased می‌توان به مدلی با واریانس کم رسید. که این دو پایه‌های Random Forest هستند. Random Forest مجموعه‌ای از درخت‌های تصمیم است. این الگوریتم تعداد زیادی درخت تصمیم می‌سازد و از آن‌ها برای Classification استفاده می‌کند. به همین خاطر به آن جنگل می‌گویند زیرا از درخت‌های متفاوتی تشکیل شده است.

$$S = \begin{bmatrix} f_{A1} & f_{B1} & f_{C1} & C_1 \\ \vdots & & \vdots & \\ f_{AN} & f_{BN} & f_{CN} & C_N \end{bmatrix}$$

به طور مثال S در تصویر بالا یک Observation Set است که هر سطر آن یک Sample است و هر ستون مقادیر یک feature برای سمپل‌های مختلف و ستون آخر کلاس هر سمپل مجموعه‌ی آموزش را نشان می‌دهند.

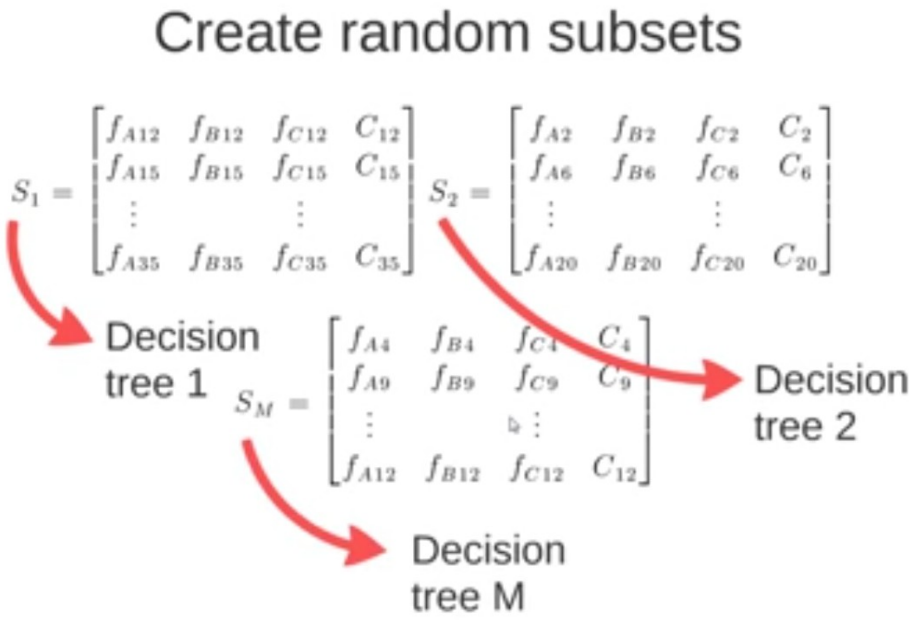
در این روش به تعداد M تا زیر مجموعه از مجموعه‌ی S استخراج می‌کنیم که در هر زیر مجموعه بعضی از سمپل‌های مجموعه‌ی آموزش به صورت رندوم وجود ندارند:

Create random subsets

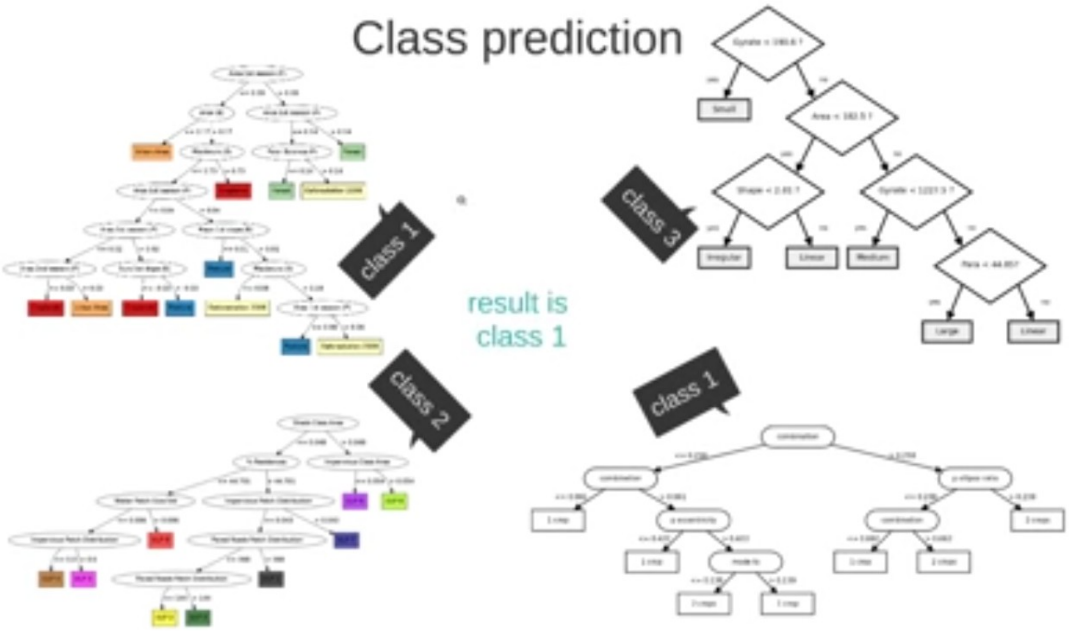
$$S_1 = \begin{bmatrix} f_{A12} & f_{B12} & f_{C12} & C_{12} \\ f_{A15} & f_{B15} & f_{C15} & C_{15} \\ \vdots & & \vdots & \\ f_{A35} & f_{B35} & f_{C35} & C_{35} \end{bmatrix} \quad S_2 = \begin{bmatrix} f_{A2} & f_{B2} & f_{C2} & C_2 \\ f_{A6} & f_{B6} & f_{C6} & C_6 \\ \vdots & & \vdots & \\ f_{A20} & f_{B20} & f_{C20} & C_{20} \end{bmatrix}$$

$$S_M = \begin{bmatrix} f_{A4} & f_{B4} & f_{C4} & C_4 \\ f_{A9} & f_{B9} & f_{C9} & C_9 \\ \vdots & & \vdots & \\ f_{A12} & f_{B12} & f_{C12} & C_{12} \end{bmatrix}$$

و بعد برای هر subset یک درخت تصمیم می‌سازیم.



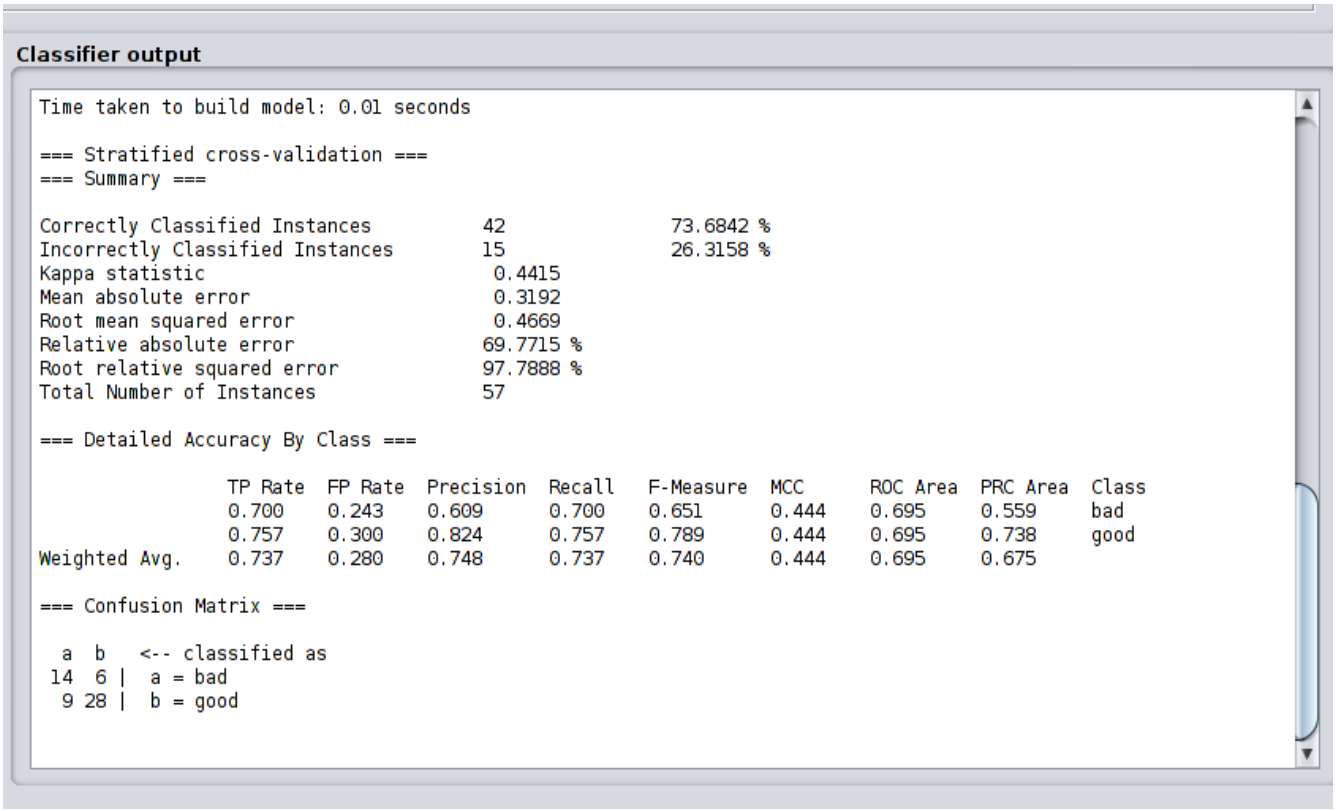
سپس بعد از ساخت جنگل(مجموعه‌ای از درخت ها) از آن برای تعیین کلاس نمونه‌هایی که تاحالا ندیده ایم استفاده می‌کنیم. برای تعیین کلاس یک نمونه، نمونه را به تمام درخت‌های تصمیم می‌دهیم و سپس پیشبینی هر درخت تصمیم را جمع‌آوری می‌کنیم و درنهایت از بین مجموعه‌ی پیشبینی درخت‌ها کلاسی را انتخاب می‌کنیم که بیشتر از سایر کلاس‌ها در مجموعه تکرار شده است:



بخش دوم سوال ۴-

بخش a)

درانتهای تصویر زیر ماتریس پریشانی train داده‌های آموزش labor با J48 را مشاهده می‌کنید:



با فرض اینکه کلاس bad کلاس P است:

TP = 14 FP = 6

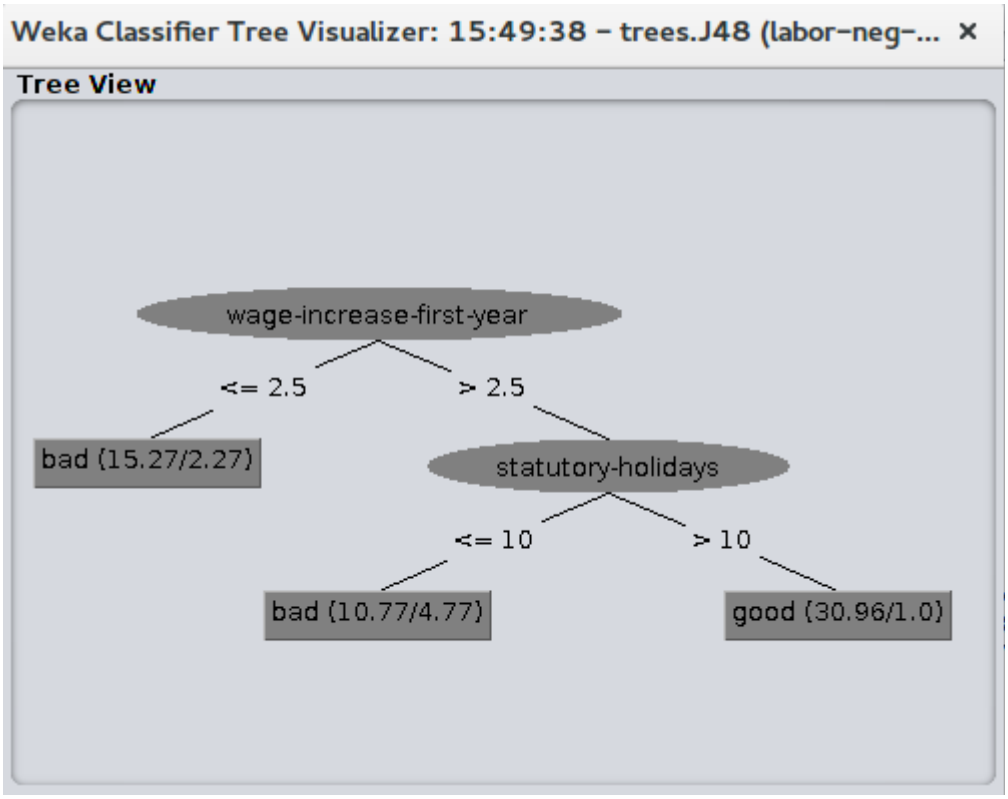
FN = 9 TN = 28

Precision = TP / (TP + FP) = 14/(14+6) = 14/20 = 0.7

Recall = TP / (TP + FN) = 14 / (14+9)=14/23 = 0.609

F-measure = 2*0.7*0.609 / (0.7+0.609) = 0.651336898

تصویر درخت تصمیم را در زیر مشاهده می‌کنید:



برای تعیین کلاس داده‌ی داده شده از ریشه شروع می‌کنیم و با توجه به مقدار ویژگی گره‌ای که در آن هستیم به سمت چپ یا راست می‌رویم تا به برگ برسیم. مقدار ویژگی wage-increase-first-year را اول مورد بررسی قرار می‌دهیم چون در ریشه است که برای داده‌ی زیر برابر با ۳ است در نتیجه به سمت بچه‌ی سمت راست حرکت می‌کنیم. در این قسمت ویژگی statutory-holidays را مورد بررسی قرار می‌دهیم که برای داده‌ی مورد نظر برابر با ۱۲ است پس بار دیگر به سمت راست حرکت می‌کنیم و به برگ می‌رسیم که مقدار برگ نشان می‌دهد کلاس داده‌ی مورد نظر good است.

feature	value	feature	value	Class
duration	1	shift-differential	20	**good**
wage-increase-first-year	3	education-allowance	yes	
wage-increase-second-year	6	statutory-holidays	12	
wage-increase-third-year	4	vacation	generous	
cost-of-living-adjustment	tcf	longterm-disability-assistance	yes	
working-hours	35	contribution-to-dental-plan	full	

pension	ret_allw	bereavement-assistance	no
standby-pay	11	contribution-to-health-plan	half

بخش b)

پارامتر unpruned هرس کردن یا هرس نکردن درخت را مشخص می‌کند. به True کردن آن درخت هرس نمی‌شود.

مقدار خطاها و F-measure و ماتریس confusion آن را در تصویر زیر مشاهده می‌کنید:

```
Time taken to build model: 0 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances          45               78.9474 %
Incorrectly Classified Instances        12               21.0526 %
Kappa statistic                          0.5378
Mean absolute error                      0.2677
Root mean squared error                  0.432
Relative absolute error                 58.5226 %
Root relative squared error             90.4708 %
Total Number of Instances               57

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.700    0.162    0.700      0.700    0.700      0.538    0.768     0.673     bad
                 0.838    0.300    0.838      0.838    0.838      0.538    0.769     0.807     good
Weighted Avg.    0.789    0.252    0.789      0.789    0.789      0.538    0.768     0.760

=== Confusion Matrix ===

  a  b   <-- classified as
 14  6 |  a = bad
  6 31 |  b = good
```

با فرض اینکه کلاس bad کلاس P است:

TP = 14 FP = 6

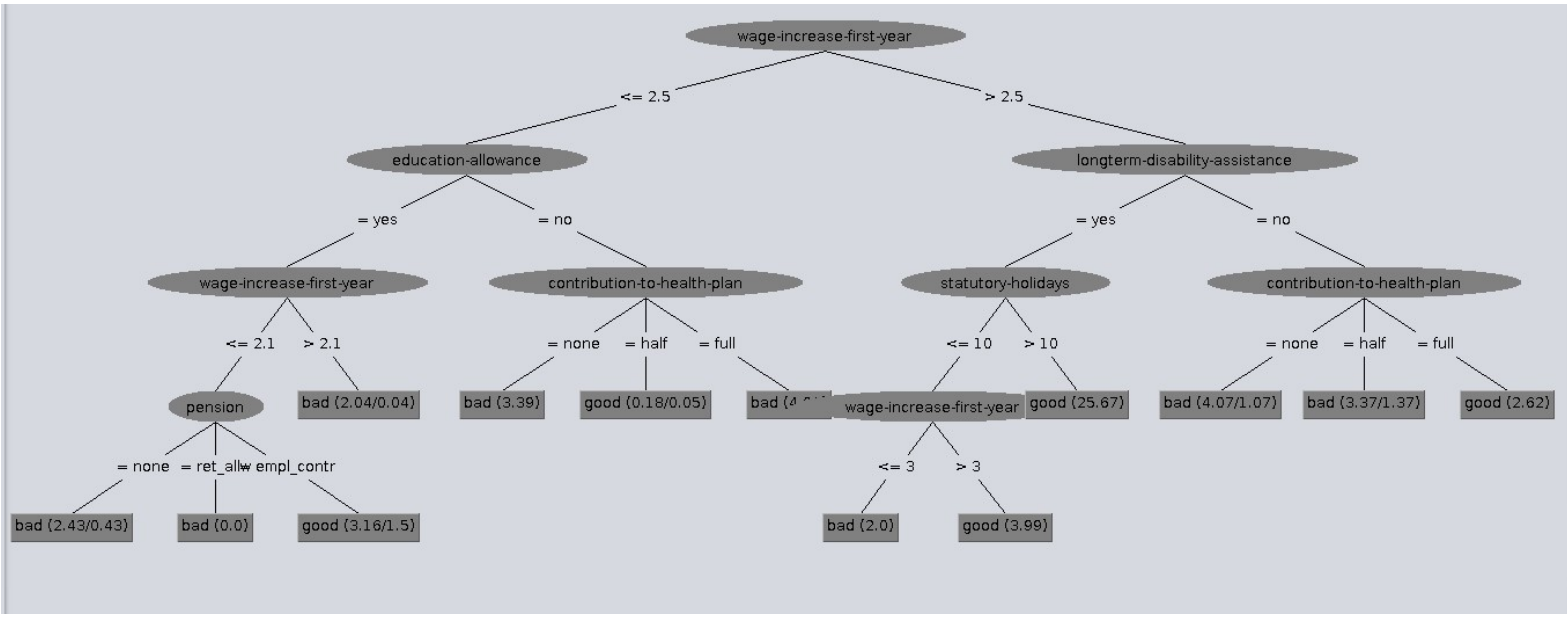
FN = 6 TN = 31

Precision = TP / (TP + FP) = 14/(14+6) = 14/20 = 0.7

Recall = TP / (TP + FN) = 14 / (14+6)=14/20 = 0.7

F-measure = 2*0.7*0.7 / (0.7+0.7) = 0.7

درخت حاصل را در تصویر زیر مشاهده می‌کنید:



داده‌ی زیر را همچون قسمت قبل از ریشه تا برگ درخت به پایین می‌آییم تا کلاس داده را مشخص شود.

Wage-increase-first-year بزرگ تر از 2.5 است پس به سمت راست می‌رویم.longterm-disability-assistance برابر با yes است پس به سمت چپ می‌رویم.statutory-holidays بزرگ تر از 10 است پس به سمت راست می‌رویم و به برگ می‌رسیم که مشخص می‌کند کلاس نمونه برابر با good است.

feature	value	feature	value	Class
duration	1	shift-differential	20	**good**
wage-increase-first-year	3	education-allowance	yes	
wage-increase-second-year	6	statutory-holidays	12	
wage-increase-third-year	4	vacation	generous	
cost-of-living-adjustment	tcf	longterm-disability-assistance	yes	
working-hours	35	contribution-to-dental-plan	full	
pension	ret_allw	bereavement-assistance	no	
standby-pay	11	contribution-to-health-plan	half	

بخش c)

میزان خطاهای مختلف و confusion ماتریس را برای Random-Forest در شکل زیر مشاهده می‌کنید:

Classifier output

```
Time taken to build model: 0.09 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances          51               89.4737 %
Incorrectly Classified Instances         6               10.5263 %
Kappa statistic                          0.7635
Mean absolute error                      0.2294
Root mean squared error                  0.3161
Relative absolute error                 50.1588 %
Root relative squared error             66.2057 %
Total Number of Instances               57

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure  MCC      ROC Area  PRC Area  Class
               0.800     0.054     0.889      0.800     0.842      0.766     0.943     0.899     bad
               0.946     0.200     0.897      0.946     0.921      0.766     0.943     0.971     good
Weighted Avg.  0.895     0.149     0.894      0.895     0.893      0.766     0.943     0.946

=== Confusion Matrix ===

  a  b   <-- classified as
 16  4 |  a = bad
  2 35 |  b = good
```

با فرض اینکه کلاس bad کلاس P است:

TP = 16 FP = 4

FN = 2 TN = 35

Precision = TP / (TP + FP) = 16/(16+4) = 16/20 = 0.8

Recall = TP / (TP + FN) = 16 / (16+2)=16/18 = 0.88

F-measure = 2*0.8*0.88 / (0.8+0.88) = 0.8380

تعداد درخت های ساخته شده برابر با 100 است:

بخش d)

جنگل تصادفی 89.47 درصد نمونه‌ها را به درستی دسته‌بندی کرده است که بیش از دو مورد قبل است. همین طور f-

measure میانگین آن برابر با 0.893 است باز بیش از دو مورد قبلی است. عملکرد جنگل تصادفی از دو مورد درخت تصمیم با هرس و بدون هرس بهتر بوده است.

دلیلش این است که تعدادی زیادی درخت تصمیم ساخته می‌شود و براساس نظر مجموع آن‌ها کلاس نمونه‌ها را مشخص می‌کند و همین استفاده از چندین کلاسیفایر (Bagging) باعث واریانس را کم می‌کند و باعث بهبود پیشبینی کلاس داده‌هایی می‌شود که تاکنون ندیده‌ایم.