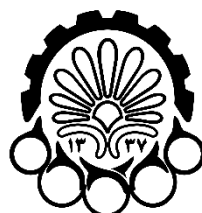


به نام خداوند بخشنده و مهربان



دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

درس هوش مصنوعی

گزارش پروژه

مقایسه روش‌های مختلف طبقه‌بندی روی یک مسأله

استاد درس:

دکتر قطعی

نام دانشجو:

امیررضا رادجو

۹۷۱۳۰۱۸

بهار ۱۴۰۰

دیتاست مورد استفاده:

دیتای مورد استفاده ما در این گزارش مربوط به داده‌های موجود در کتابخانه sklearn که مربوط به سرطان سینه است می باشد. این داده دارای ۳۱ ویژگی مختلف برای 569 داده مختلف میباشد و در اینجا ما قصد داریم که دیتا ها را بر اساس خوش خیم و یا بدخیم بودن نوع سرطان دسته بندی کنیم. دیتایی که در اختیار داریم شامل ویژگی‌های مفید و مختلفی مانند میانگین شعاع تقارن تعقر و... می‌باشد.

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	radius error	texture error	perimeter error	area error	smoothness error	compactness error
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	1.0950	0.9053	8.589	153.40	0.006399	0.04904
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	0.5435	0.7339	3.398	74.08	0.005225	0.01308
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	0.7456	0.7869	4.585	94.03	0.006150	0.04006
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	0.4956	1.1560	3.445	27.23	0.009110	0.07458
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	0.7572	0.7813	5.438	94.44	0.011490	0.02461

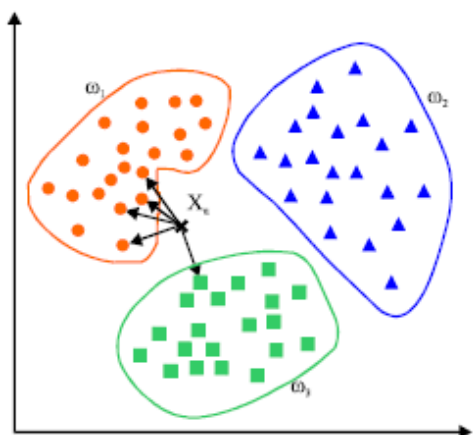
در این گزارش ما از سه روش مختلف دسته بندی خود را بر اساس خوش خیم و یا بدخیم بودن سرطان انجام میدهیم.

روش KNN یا k همسایه نزدیک:

یکی از شهودی‌ترین الگوریتم‌های دسته‌بندی روش KNN و یا K همسایگی نزدیک، است. ایده پشت این الگوریتم به این صورت است که برای این که برچسب یک نقطه جدید را تعیین کنیم بررسی می‌کنیم که K نقطه‌ای که ویژگی‌های آن‌ها به این نقطه نزدیکتر است (در فضای ویژگی‌ها به این نقطه نزدیکتر هستند) کدامند و در نهایت برچسب نقطه جدید را برابر با بیشترین برچسبی که در بین این K نقطه وجود دارد قرار می‌دهیم. توجه کنید که در این مدل پارامتری وجود ندارد که یادگیری به آن وابسته باشد. به مدل‌هایی که خروجی آن‌ها وابسته به پارامتر خاصی نیست مدل‌های غیرپارامتری (non parametric) گفته می‌شود. از این رو مدل K همسایگی نزدیک نیز مدلی ناپارامتری است. در مثالی که در مورد تومورها بررسی می‌کردیم نیز بطور مشابه برای تصمیم‌گیری در مورد یک نمونه جدید

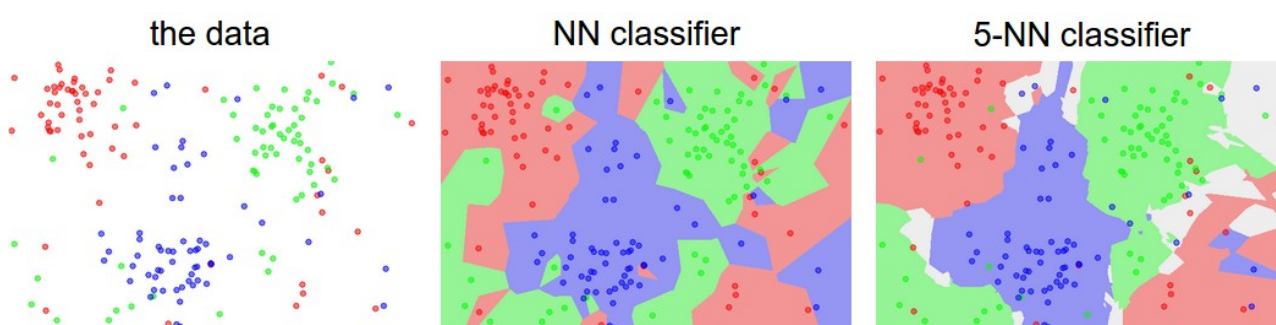
(اندازه‌گیری‌های مربوط به یک بیمار جدید) K همسایگی نزدیک به این نمونه را در نظر بگیریم (تومورهایی که بر اساس ویژگی‌های اندازه‌گیری شده به این تومور شبیه بوده‌اند) و برچسب اکثریت را در این همسایگی به نقطه جدید نیز نسبت دهیم زیرا معقول به نظر می‌رسد که تومورهای با ویژگی‌های مشابه در یک

دسته (خوش‌خیم یا بدخیم) قرار بگیرند.



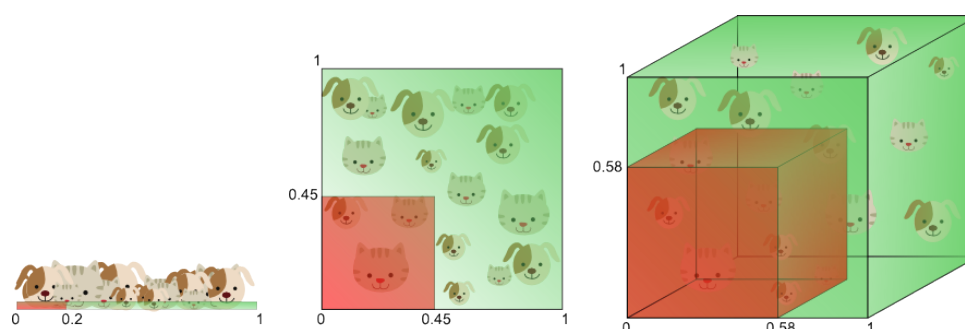
در این مدل دو ابر پارامتر (*Hyperparameter*) وجود دارد. (ابر پارامتر به هر پارامتری از مدل گفته می‌شود که در فرآیند یادگیری وارد نمی‌شود و ثابت هستند و قبل از یادگیری باید تعیین شوند برخلاف پارامترهای معمولی که در فرآیند یادگیری تعیین می‌شوند). در این مدل یکی خود عدد K ، یعنی اندازه همسایگی حول یک نقطه است. و دیگری انتخاب یک معیار برای اندازه‌گیری فاصله است که بر اساس آن همسایگی تعیین می‌شود. در ریاضیات می‌توان انواع زیادی تابع به عنوان معیار فاصله در نظر گرفت.

نمودار زیر نیز مرز تصمیم‌گیری را برای این الگوریتم نمایش می‌دهد. منظور از مرز تصمیم‌گیری این است که الگوریتم برای هر ناحیه (یا در برخی مواقع نقطه) از فضا چه تصمیمی می‌گیرد.



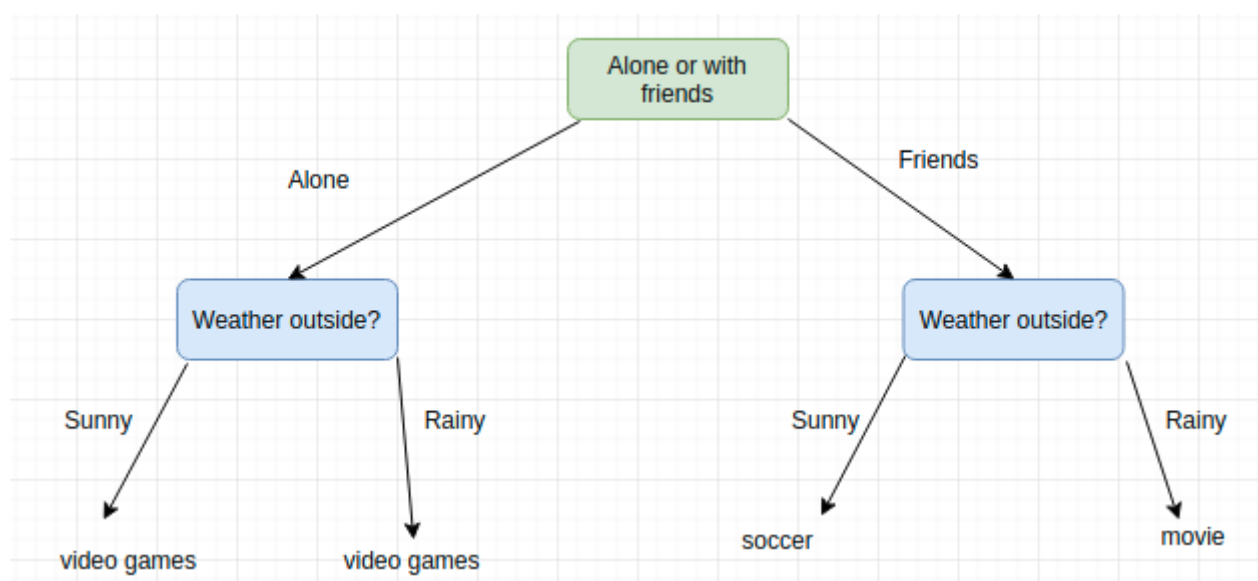
نمایش این نمودار در فهمیدن این که روش ما روی داده چطور تصمیم‌گیری می‌کند بسیار مهم است و گاهی می‌توان با دیدن این نمودار به برطرف کردن مشکلات مدل پرداخت.

با این که این منطق پشت این مدل امید بخش است اما دست کم یک مشکل جدی دارد. با بالا رفتن بُعد، مشکل نفرین بُعد یا **curse of dimension** پیش می‌آید. نفرین بُعد، در یادگیری ماشین به سنگین شدن هزینه‌های محاسباتی و سخت شدن مسأله در بُعد بالا گفته می‌شود. مسأله این است که شهودی که در ابعاد پایین (کمتر مساوی ۳) برای ما سازگار است در فضاهای با بُعد بالاتر دارد صادق نیستند. در روش K همسایگی نزدیک این مشکل به این معنا است که با افزایش بُعد، نقاطی که در یک همسایگی هستند کمتر شبیه به هم خواهند بود. از این رو برای مسئله‌های دسته‌بندی که بُعد داده‌ی ورودی آن بیشتر از ۳ باشد از الگوریتم KNN استفاده نمی‌شود.



درخت تصمیم گیری:

یک راه ساده استفاده از مدل‌های درخت تصمیم‌گیری (Decision Tree) است. مدل‌های درختی خانوادگی ساده اما قدرتمند در یادگیری ماشین هستند. از آن‌ها هم در مسئله‌های دسته‌بندی و هم رگرسیونی استفاده می‌شود. به‌طور شهودی ایده‌ی اصلی در مدل‌های درختی، سوال‌های متوالی و سلسله‌رابتی است که تصمیم‌گیری در مورد این سوالات، در نهایت، ما را به جواب می‌رساند. به عنوان مثال فرض کنید که می‌خواهید در مورد اینکه بعد از ظهر جمعه پاییزی را به چه ترتیب بگذرانید تصمیم‌گیری کنید و به دلایلی برایتان واضح نیست که چه برنامه‌ی بچینید. در این‌صورت می‌توانید با بررسی چندین پرسش به جواب برسید. مثلاً اولین سوال می‌تواند در مورد هوا باشد. مثلاً اگر هوا ابری بود تصمیم می‌گیرید با گروه دوستانتان وقت بگذرانید و اگر صاف بود ترجیح می‌دهید به تنهایی پیاده‌روی کنید. و مثلاً در گذراندن وقت با دوستانتان اگر مجموع پول‌هایتان از حدی، مثلاً ۱۰۰ تومن، بیشتر شد از بیرون غذا سفارش می‌دهید و فیلم می‌بینید و در غیر این‌صورت زمین فوتبال اجاره می‌کنید. پس می‌توانیم تصمیم‌گیری در مورد برنامه‌ی بعد از ظهر جمعه را به‌صورت زیر نمایش دهیم:



مدل‌های درختی، روی داده‌های واقعی نیز به‌طور مشابهی عمل می‌کنند. یعنی اگر داده‌ی با n ویژگی داشته باشیم و بخواهیم l را پیش‌بینی کنیم، در هر گام، یکی از متغیرهای ویژگی را بر اساس آستانه‌ی به دو بخش تقسیم می‌کنیم و بعد در هر گروه به سراغ متغیر دیگری می‌رویم.

در شکل زیر میتوانید یک درخت تصمیم گیری پیچیده تر را برای سرطان سینه مشاهده کنید

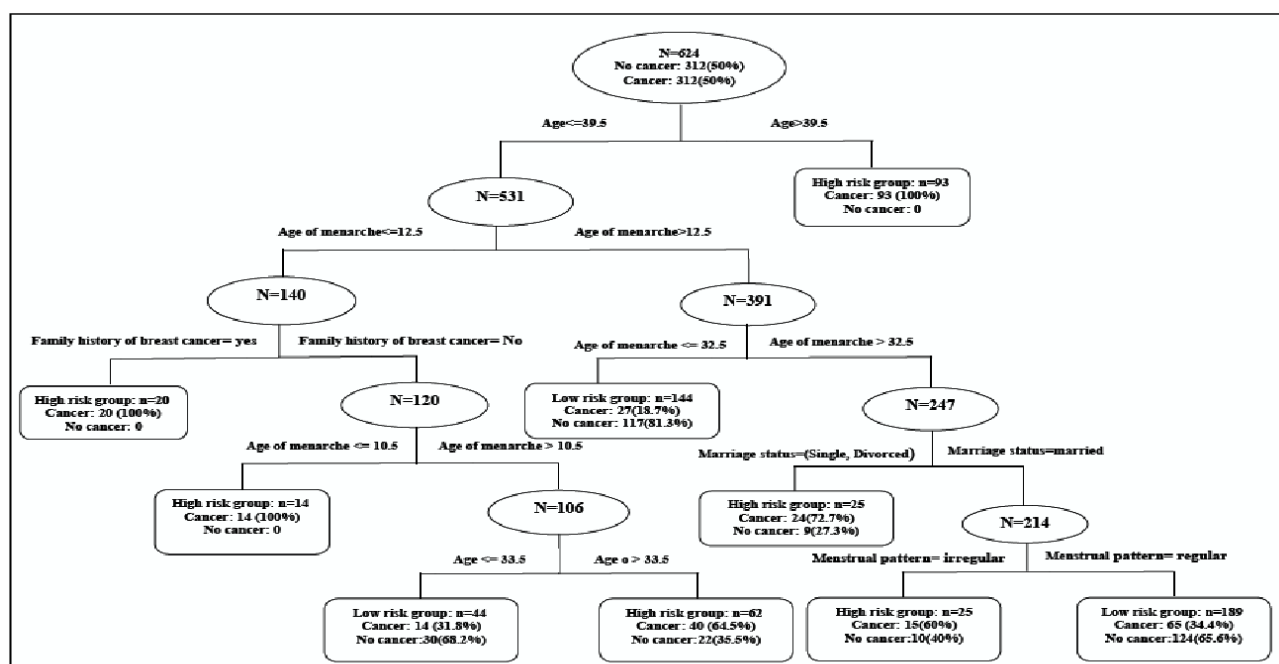
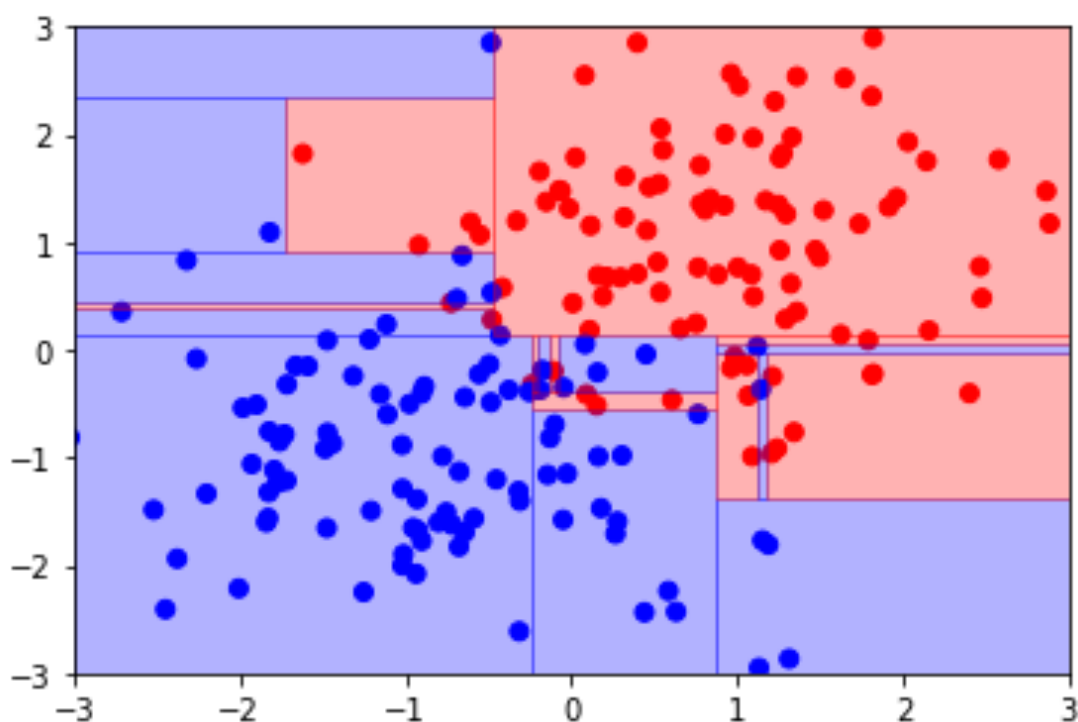


Figure 1. Decision Tree of Women Referred in Imam Khomeini Hospital in Tehran

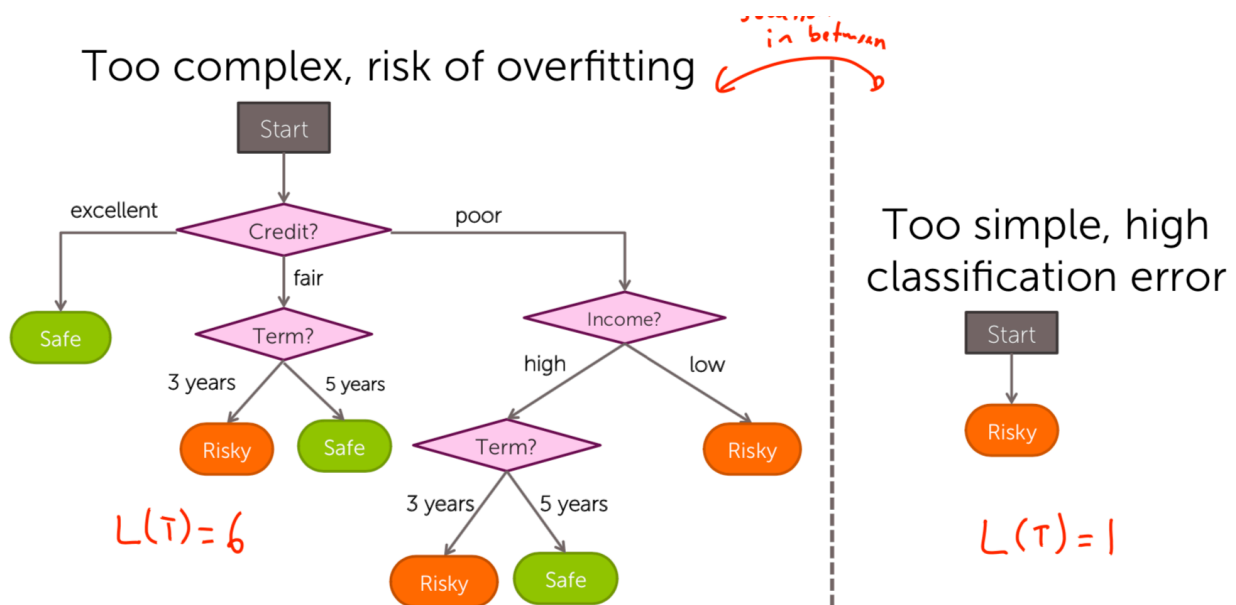
نمودار زیر نیز مرز تصمیم‌گیری را برای این الگوریتم نمایش می‌دهد. منظور از مرز تصمیم‌گیری این است که الگوریتم برای هر ناحیه (یا در برخی مواقع نقطه) از فضا چه تصمیمی می‌گیرد.



دو نکته در مورد مدل درخت تصمیم‌گیری را باید در ذهن داشت.

اول اینکه این مدل‌ها **ترجمه‌پذیر** هستند. یعنی می‌توان خروجی مدل را به خوبی تحلیل کرد. زیرا در حالت ساده ما با چندین مرحله جدا کردن داده به هر جواب می‌رسیم، پس می‌دانیم دقیقاً هر جواب به چه صورت تولید شده است. مشکلی که در اغلب مدل‌هایی که امروزه تب داغی برای آن‌ها وجود دارد، مثل شبکه‌های عصبی، ترجمه‌پذیری خروجی مدل است. بنابراین هرگاه که مدل ساده‌ای با عملکرد نسبتاً مشابه داریم که روند تصمیم‌گیری در آن روشن است به سراغ مدل ساده‌تر اما قابل ترجمه می‌رویم. چراکه فراموش نکنید که در نهایت ما می‌خواهیم بر اساس خروجی مدل، در مورد یک مسئله که ما به ازای خارجی دارد، تصمیم‌گیری کنیم و باید این تصمیم را در کنترل‌شدترین حالت ممکن اتخاذ کنیم تا صدمات ناشی از آن را در کمترین مقدار ممکن نگه داریم.

نکته دوم اما این است که این مدل‌ها قابلیت بیش‌برازش به داده را دارند که پیش‌تر گفتیم که در مدل‌های یادگیری ماشین این موضوع اصلاً قابل قبول نیست. در مدل‌های درخت تصمیم‌گیری، با افزایش عمق درخت، امکان بیش‌برازش به داده وجود دارد. و دلیل آن نیز روشن است، زیرا با افزایش عمق، حالت‌بندی داده را به قدری ظریف کردیم که ممکن است در برگ‌های انتهایی تنها یک داده قرار بگیرد و به اصطلاح گفته می‌شود که مدل داده را حفظ (از بر) کرده است. اما چنین مدلی در پیش‌بینی برای نقاط جدید ضعیف عمل خواهد کرد.



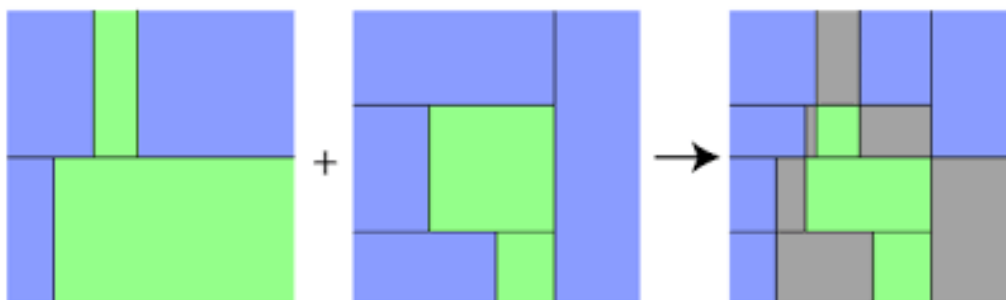
جنگل تصادفی:

گفتیم که مدل‌های درخت تصمیم‌گیری علی‌رغم تمام ویژگی‌های مثبتی که دارند مثل ترجمه‌پذیری و سادگی، اما با یک نقطه‌ی ضعف جدی روبه‌رو هستند و آن، قابلیت بیش‌برازش آن‌ها به داده است. اما می‌توان برای حل این مشکل چاره‌ای اندیشید.

قبل‌تر گفتیم که، خطای مدل را می‌توان در یکی از دو منشا اریبی و واریانس داده جستجو کرد. اگر بخواهیم بدون اینکه مدل را اریب کنیم، خطا را کاهش دهیم، باید راهی پیدا کنیم که واریانس مدل کاهش پیدا کند. مشکل بیش‌برازش به داده در مدل‌های درخت تصمیم‌گیری، با زیاد شدن عمق، نیز ناشی از واریانس زیاد پیش‌بینی آن‌هاست. یعنی پیش‌بینی که مدل برای هر نمونه ارائه می‌دهد، در صورت تکرار فرآیند، پراکندگی زیادی دارد. در آمار مساله‌ی کاهش واریانس، مسئله‌ی جدی است و روش‌های متعددی برای این منظور توسعه یافته‌اند. یکی از معمول‌ترین این روش‌ها میانگین‌گیری است. یعنی برای اینکه تخمینی با واریانس کمتر ارائه دهیم، فرآیند تخمین را چندین بار تکرار می‌کنیم و نتیجه‌ی نهایی را میانگین نتیجه‌های گزارش شده اعلام می‌کنیم. ثابت می‌شود که با این کار، واریانس نتیجه کمتر خواهد شد.

ایده‌ی روش جنگل تصادفی نیز بر مبنای این حقیقت بنا شده است. در این روش چندین مدل درخت تصمیم‌گیری با عمق کم را روی داده آموزش می‌دهیم و نتیجه‌ی نهایی را برابر با نوعی میانگین بین نتیجه‌ی هر درخت در نظر می‌گیریم. جنگل تصادفی در کلاس مدل‌های **Ensemble** قرار می‌گیرد. در این مدل‌ها، نتیجه‌ی نهایی بر اساس ترکیبی از نتیجه‌های چندین مدل به دست می‌آید. این ترکیب کردن باعث می‌شود که پیش‌بینی پایدارتری داشته باشیم که واریانس و اریبی کمتری دارد. در روش جنگل‌های تصادفی، چندین مدل درختی ساده با عمق کم را می‌گیریم اما هر مدل را روی تمام داده آموزش نمی‌دهیم. در واقع عبارت تصادفی در اسم این مدل به این ویژگی برمی‌گردد که هر درخت تصمیم‌گیری را روی زیرمجموعه‌ی تصادفی از داده با تعداد تصادفی از ویژگی‌های داده آموزش می‌دهیم. در نهایت در مسئله‌های دسته‌بندی، کلاسی که مدل جنگل تصادفی برای هر داده پیش‌بینی می‌کند، رای اکثریت بین نتیجه‌ی درخت‌ها است (برای مسئله‌های رگرسیون نیز میانگین نتیجه‌ی درخت‌ها را گزارش می‌دهد).

نمودار زیر نیز مرز تصمیم‌گیری را برای این الگوریتم نمایش می‌دهد. منظور از مرز تصمیم‌گیری این است که الگوریتم برای هر ناحیه (یا در برخی مواقع نقطه) از فضا چه تصمیمی می‌گیرد.



سنجش مدل:

بعد از هر مسئله‌ی دسته‌بندی، بر اساس نتیجه‌ی مدل ماتریسی تشکیل داده می‌شود، به نام ماتریس درهم‌ریختگی (*Confusion Matrix*). فرض کنید مسئله‌ی دسته‌بندی با دو کلاس (برچسب)، ۰ و ۱ داریم. بعد از اینکه مدل را روی داده آموزش دهیم و خروجی مدل را روی داده‌ی آزمایشی بگیریم می‌توان نتیجه را به شکل ماتریس زیر نمایش داد:

Confusion Matrix

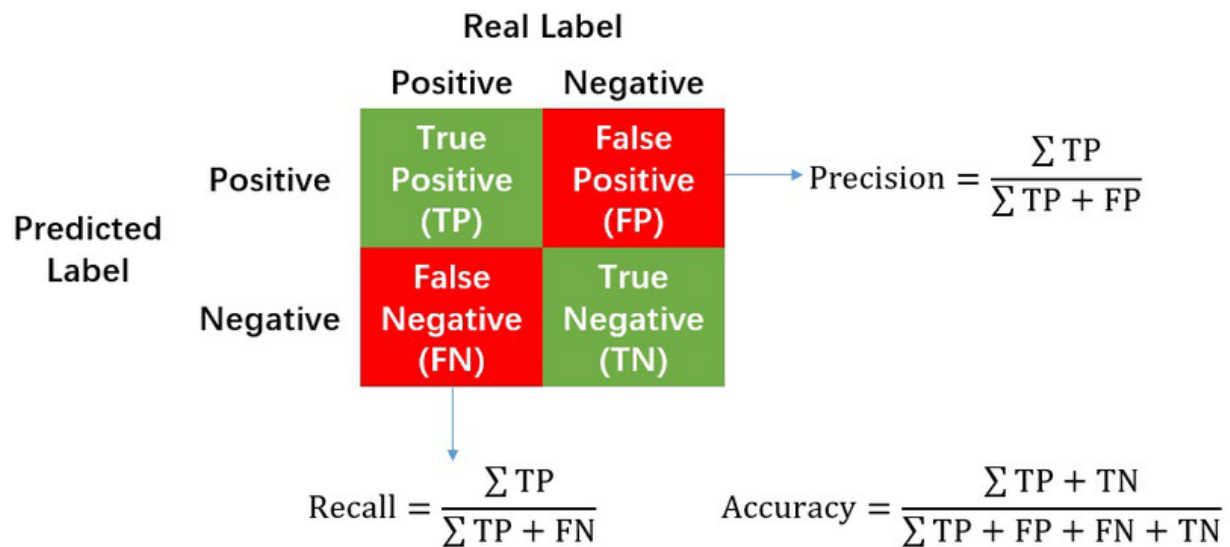
	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

در این ماتریس *True Positives* به پیش‌بینی‌هایی می‌گوییم که برای کلاس مثبت به درستی انجام شده‌اند. به عنوان مثال، مساله‌ی سرطان سینه را در نظر بگیرید. *True Positive* در این مثال به تعداد نمونه‌های از داده می‌گوییم که کلاسی که مدل برای آن‌ها پیش‌بینی می‌کند، بدخیم است (*Positive*) و در واقعیت نیز آن‌ها در گروه بدخیم سرطان بودند (*True*). به همین ترتیب نیز *True Negatives* به نمونه‌هایی که گفته می‌شود در گروه از خوش خیم سانحه پیش‌بینی شده‌اند (*Negative*) و در حقیقت نیز بدخیم بودند (*True*). اما گاهی کلاس پیش‌بینی شده با کلاس حقیقت متفاوت است، در این مواقع اگر در حقیقت نمونه‌ای به کلاس منفی (خوش خیم) تعلق داشته باشد و در کلاس مثبت (بدخیم) پیش‌بینی شود، از عبارت *False Positive* استفاده می‌کنیم در حالت عکس از عبارت *False Negative* استفاده می‌کنیم. حال که با درایه‌های این ماتریس را آشنا شدیم، به معرفی معیارهای دقت (*Accuracy*)، صحت (*Precision*)، یادآوری (*Recall*) و امتیاز *F-1 (F1-Score)* می‌پردازیم.

دقت شهودی‌ترین اندازه‌ی است که به عملکرد یک مدل می‌توان نسبت داد، و آن نسبت پیش‌بینی‌های درست به کل مشاهدات است.

صحت نسبت پیش‌بینی‌های درست از کلاس مثبت به کل پیش‌بینی‌های مثبت است.

recall نیز نسبت پیش‌بینی‌های درست از کلاس مثبت، به تمام نمونه‌های با کلاس مثبت در داده است.



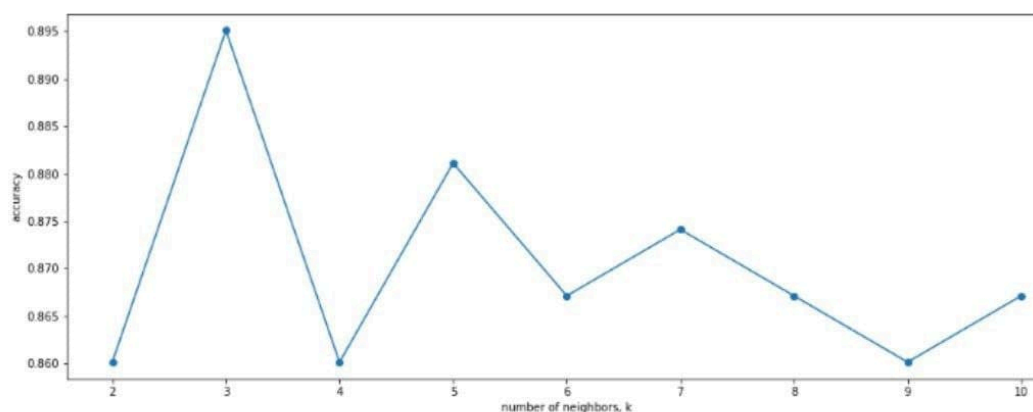
امتیاز F1 نیز جمع وزنداری از دو معیار صحت و یادآوری است. بنابراین این معیار هر دو مفهوم *False Positive* و *False Negative* را مد نظر قرار می‌دهد. به‌طور شهودی، تعبیر این معیار به سادگی معیاری مثل دقت نیست، اما بیشتر از آن استفاده می‌شود به خصوص زمانی که توزیع کلاس‌های متفاوت یکی نباشد. دقت در حالتی که ما برای *False Positive* و *False Negative* هزینه‌ی یکسانی پرداخت کنیم، معیار خوبی است. منظور از هزینه داشتن، همان عواقب تصمیم‌گیری بر اساس مدل است. فرض کنید مسئله‌ی ما شناسایی ژن موثر در بیماری و در نهایت از بین بردن این ژن است. واضح است که در این مساله *False Positive* هزینه‌ی زیادی برای ما دارد چراکه اگر ژن واقعاً در بیماری موثر نباشد، ما در واقع داریم یک ژن سالم را از بین می‌بریم، بنابراین هزینه‌ی که *False Positive* دارد در این مسئله با هزینه‌ی *False Negative* متفاوت است. زمانی که هزینه‌ی این دو متفاوت باشد، بهتر است در سنجش عملکرد مدل هر دو را دخیل کنیم، و در این مسئله‌ها معیار امتیاز *F1**، بیشتر استفاده می‌شود.

$$\text{F1 Score} = \frac{2 \times (\text{Precision} \times \text{Recall})}{\text{Precision} + \text{Recall}}$$

بررسی کدها:

مدل knn:

```
[6]: X = dataframe[['mean radius','mean concave points']].values
[7]: Y = dataframe['target'].map({'benign':0,'malignant':1})
[10]: X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = 0.25,stratify = Y)
[11]: from sklearn.neighbors import KNeighborsClassifier
      from sklearn.metrics import accuracy_score
[12]: K = [2,3,4,5,6,7,8,9,10]
      acc = []
      for k in K:
          knn = KNeighborsClassifier(n_neighbors=k)
          knn.fit(X_train,Y_train)
          y_pred_knn = knn.predict(X_test)
          acc.append(accuracy_score(Y_test,y_pred_knn))
[13]: plt.figure(figsize = (16,6))
      plt.plot(K, acc, '-o')
      plt.xlabel('number of neighbors, k')
      plt.ylabel('accuracy')
      plt.xticks(K)
      plt.show()
```



```
[14]: knn = KNeighborsClassifier(n_neighbors=3)
      knn.fit(X_train,Y_train)
[14]: KNeighborsClassifier(n_neighbors=3)
[15]: y_pred = knn.predict(X_test)
[16]: print(metrics.classification_report(Y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.89	0.96	0.92	90
1	0.91	0.79	0.85	53
accuracy			0.90	143
macro avg	0.90	0.87	0.88	143
weighted avg	0.90	0.90	0.89	143

در این قسمت از کد بعد از تعریف کردن X و y (که در قسمت‌های بالاتر توضیح دادیم که چرا دو ستون از X انتخاب کردیم) یک کار کلیدی دیگر نیز انجام می‌دهیم و آن این است که مقدار بهینه برای k را پیدا می‌کنیم. مقدار بهین را میتوان از نموداری که رسم کرده این تشخیص دهیم.

بنابراین با استفاده از آن مدل را می‌سازیم و عملکرد سیستم را با توجه به داده‌های تستی که جدا کرده بودیم می‌سنجیم.

مدل درخت تصمیم گیری:

Decision Tree

```
[17]: from sklearn.tree import DecisionTreeClassifier
      from six import StringIO
      import pydotplus
      from IPython.display import Image
      from sklearn.tree import export_graphviz
```

```
[18]: X = dataframe.drop(columns='target')
```

```
[19]: y = dataframe['target'].map({'benign':0,'malignant':1})
```

```
[20]: X_train,X_test,y_train,y_test = train_test_split(X, y, test_size = 0.25, stratify = y)
```

```
[21]: tree_model = DecisionTreeClassifier()
      tree_model.fit(X_train,y_train)
```

```
[21]: DecisionTreeClassifier()
```

```
[22]: y_predicted = tree_model.predict(X_test)
```

```
[23]: print(metrics.classification_report(y_test, y_predicted))
```

	precision	recall	f1-score	support
0	0.98	0.98	0.98	90
1	0.96	0.96	0.96	53
accuracy			0.97	143
macro avg	0.97	0.97	0.97	143
weighted avg	0.97	0.97	0.97	143

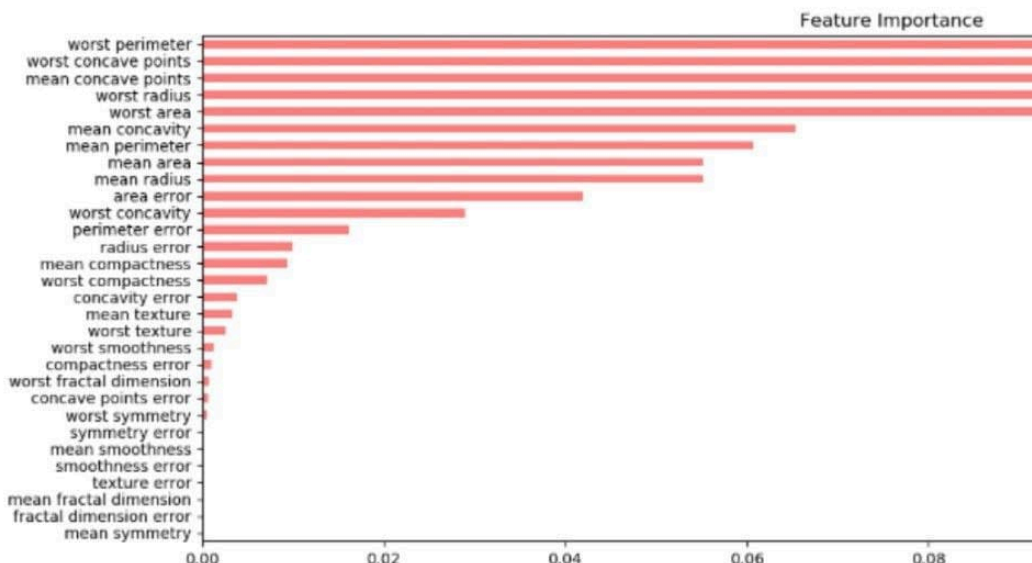
در این قسمت کار اضافه خاصی نیاز نیست که انجام دهیم. تنها تفاوتی که بین یادگیری این الگوریتم و الگوریتم قبلی است (منظور در دادن داده‌ها به الگوریتم است و گرنه کلیت این دو الگوریتم به طور کلی با هم فرق دارند) آن است که در این الگوریتم از تمامی ستون‌های ویژگی در داده‌ها استفاده می‌کنیم ولی در حالت قبل به دلیل بروز مشکلاتی چون نفرین بعد تنها از دوتا از آن‌ها استفاده میشد.

جنگل تصادفی:

```

1 X = dataframe.drop(columns='target')
2 y = dataframe['target'].map({'benign':0,'malignant':1})
3 X_train,X_test,y_train,y_test = train_test_split(X, y, test_size = 0.25, stratify = y)
4 from sklearn.ensemble import RandomForestClassifier
5 rf = RandomForestClassifier(n_estimators = 500, min_samples_leaf = 0.13, random_state = 1)
6 rf.fit(X_train,y_train)
7 RandomForestClassifier(min_samples_leaf=0.13, n_estimators=500, random_state=1)
8 y_pred_rf = rf.predict(X_test)
9 importances_rf = pd.Series(rf.feature_importances_,
10                             index = X.columns)
11 sorted_importances_rf = importances_rf.sort_values()
12 plt.figure(figsize = (16,6),dpi = 100)
13 sorted_importances_rf.plot(kind = 'barh',
14                             color = 'red',alpha = 0.5)
15 plt.title("Feature Importance")
16 <matplotlib.text.Text at 0x7f8c14149910>

```



```

17 print(metrics.classification_report(y_test, y_pred_rf))

```

	precision	recall	f1-score	support
0	0.91	0.98	0.94	90
1	0.96	0.83	0.89	53
accuracy			0.92	143
macro avg	0.93	0.90	0.92	143
weighted avg	0.93	0.92	0.92	143

داده‌های داده شده به الگوریتم در این روش کاملاً یکسان با حالت قبلی می‌باشد. اما این روش نکته جالب دیگری را نیز می‌تواند به ما برگرداند و آن میزان اهمیت هر کدام از ویژگی‌های موجود در داده‌ها می‌باشد. که در انتهای کد آن‌ها را به صورت یک نمودار دریافت کردیم.

مقایسه نهایی بر اساس نتایج:

در نتایجی که از به دست آمد ضعیف ترین عملکرد مربوط به k همسایه نزدیک با تعداد بعد بالا بود که دلیل ایجاد اشکال در این حالت مشکل نفرین بعد در این حالت بود. که به دلیل عملکرد بسیار ضعیف مقدار محاسبه شده آن را در نوت بوک نیاوردیم اما به راحتی میتوان با اضافه کردن چندین ستون دیگر به آن تغییرات و کاهش دقت را در آن مشاهده کرد. knn در حالت ۲ بعدی عملکرد بسیار خوبی داشت و تا نزدیک به ۹۰ درصد نیز توانست خود را تقویت کند. اما در بهترین حالت خود نیز از درخت تصمیم گیری عملکرد ضعیف تری داشت. یکی از علل احتمالی میتواند این باشد که در دسته بندی به کمک درخت تصمیم گیری ما صرفاً به دو ویژگی مدل را محدود نمی کردیم و مدل بدون مشکل میتواند ابعاد دیگر و ویژگی های دیگر را بررسی کند و به درصد بالاتر از ۹۰ برسد. هرچند در این حالت نیز بدون مشکل نبودیم. همانطور که گفته شد یکی از اصلی ترین مشکلات موجود در مدل های درخت تصمیم گیری $overfit$ شدن مدل میباشد که این مساله را نیز با کمک مدل جنگل تصادفی حل کردیم و اینگونه توانستیم کیفیت مدل را باز هم کمی بهبود ببخشیم.

حدس میزنیم که در صورت استفاده از شبکه های عصبی میتوانستیم کیفیت مدل را از این مقدار هم بیشتر کنیم اما به دلیل ترجمه نابذیری این مدل ها و همچنین حساس بودن داده ها و لزوم نیاز به ترجمه پذیری ترجیح دادیم که از این مدل استفاده نکنیم.

منابع:

کتاب *hands on machine learning*

<https://towardsdatascience.com>

دوره هوش مصنوعی *quera*

<https://lawtomated.com/accuracy-precision-recall-and-f1-scores-for-lawyers>

[https://www.semanticscholar.org/paper/Diagnostic-classification-scheme-in-Iranian-](https://www.semanticscholar.org/paper/Diagnostic-classification-scheme-in-Iranian-breast-Malehi/fbad2686e02bc546b7140a006f80b9eb6c8c3503)

[breast-Malehi/fbad2686e02bc546b7140a006f80b9eb6c8c3503](https://www.semanticscholar.org/paper/Diagnostic-classification-scheme-in-Iranian-breast-Malehi/fbad2686e02bc546b7140a006f80b9eb6c8c3503)

[/https://dimensionless.in/introduction-to-random-forest](https://dimensionless.in/introduction-to-random-forest)

kaggle.com