**گزارش تمرین چهارم مبانی داده‌کاوی و کاربردهای آن (۲۱۰۱۹)**

استاد: دکتر مانا مس‌کار
دستیار آموزشی: محمد مهدی منتظری هدش، سبحان کسائی
اعضای گروه: صبا عبدی (۴۰۱۱۰۴۲۷۶)، آوا صدیقی (۴۰۱۱۱۰۱۵۹)
دانشکده مهندسی صنایع

## بخش ۱: Decision Tree

دانلود داده این بخش

در این بخش شما با ساخت، تحلیل و بهینه‌سازی درخت تصمیم جهت پیش‌بینی بدخیم یا خوش‌خیم بودن تومور آشنا خواهید شد.

۱. (۵ نمره) **ساخت مدل پایه**

یک مدل درخت تصمیم با استفاده از معیار تقسیم `gini` و مقدار `min_samples_split=10` بسازید. ساختار درخت را رسم کنید.

به این پرسش نیز پاسخ دهید: کدام ویژگی در ریشه درخت قرار گرفت؟ چرا فکر می‌کنید این ویژگی بیشترین قدرت تفکیک را داشته است؟

پیش از هر کاری، کتابخانه‌های مرسوم را فراخوانی می‌کنیم و داده‌های این بخش را بارگذاری می‌کنیم. پیش از ساخت درخت موردنظر، داده‌های خود را به دو بخش train و test تقسیم می‌کنیم تا مدل صحیحی بسازیم. در این بخش از کد ما پارامتر stratisfied را برابر y می‌گذاریم. این پارامتر باعث می‌شود تا نسبت کلاس‌ها در هر دو مجموعه حفظ شوند؛ یعنی تقسیم‌بندی طوری انجام شود که نسبت برچسب‌ها (مثلاً تعداد کلاس ۰ و ۱) در هر دو مجموعه آموزش و تست، تقریباً مشابه با نسبت کل داده‌ها باقی بماند. استفاده از این پارامتر در داده‌های نامتوازن مثل تشخیص سرطان بسیار مهم است تا مدل ارزیابی منصفانه‌ای روی هر دو کلاس داشته باشد.

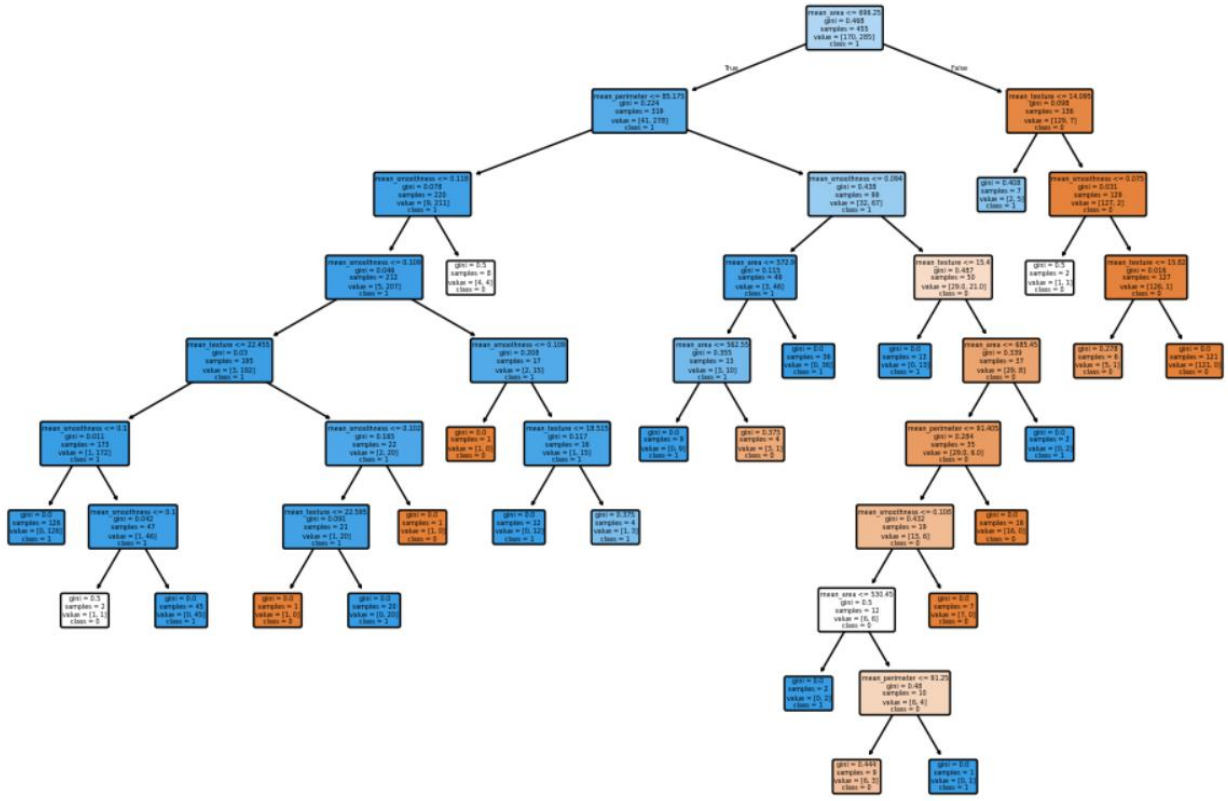
```
[11] X_train, X_test, y_train, y_test = train_test_split(
         X, y, test_size=0.2, random_state=42, stratify=y
     )
```

حال که داده‌ها تقسیم‌بندی شدند، مدل درخت تصمیم را با پارامترهای خواسته شده، می‌سازیم.

```
# constructing the tree
clf = DecisionTreeClassifier(criterion='gini', min_samples_split=10, random_state=42)
clf.fit(X_train, y_train)
```

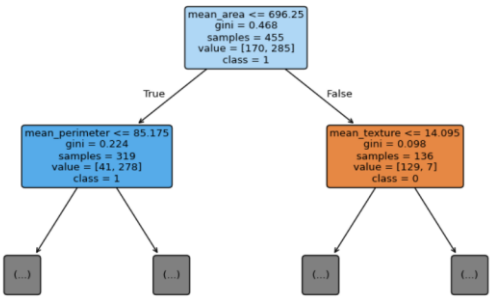
```
DecisionTreeClassifier
DecisionTreeClassifier(min_samples_split=10, random_state=42)
```

ساختار درخت ساخته شده نیز به شرح زیر است:



لازم به ذکر است برای دیدن بهتر درخت، می‌توانید پارامتر max_depth را در تابع plot_tree اضافه کنید. در این گزارش به دلیل خواسته سوال، کل درخت به نمایش گذاشته شده است.

ویژگی mean_area در ریشه درخت قرار گرفته است. قدرت این ویژگی را می‌توان به دو صورت تعبیر کرد. اول آنکه الگوریتم درخت تصمیم به‌صورت خودکار فیچری را در گره ریشه انتخاب می‌کند که بیشترین کاهش در Gini impurity را ایجاد کند. این بدان معناست که با تقسیم داده‌ها بر اساس mean_area، ناخالصی کلاس‌ها در زیرمجموعه‌های حاصل بیشترین کاهش را خواهد داشت. دوم آنکه ویژگی mean_area نشان‌دهنده میانگین مساحت سلول‌های تومور است. بر اساس domain knowledge سلول‌های تومور بدخیم معمولاً مساحت بیشتری نسبت به توده‌های خوش‌خیم دارند. بنابراین mean_area از لحاظ زیستی نیز یک معیار تفکیک‌کننده طبیعی و قوی برای شناسایی نوع تومور به شمار می‌رود.



۲. (۶ نمره) **بررسی overfitting از طریق عمق درخت**

مدل‌هایی با مقادیر max_depth از ۱ تا ۱۰ آموزش دهید. برای هر مدل، دقت (accuracy) روی داده آموزش و تست را محاسبه و نمودار زیر را رسم کنید:

- محور افقی: عمق درخت
- محور عمودی: دقت مدل

نتیجه را تحلیل کنید. در چه عمقی مدل دچار overfitting می‌شود؟

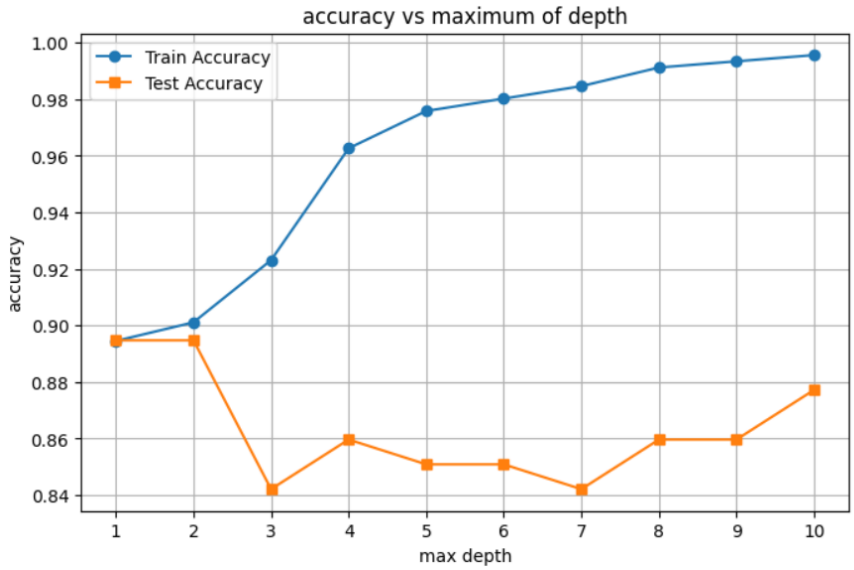
همانند بخش قبل، درخت‌ها را متناسب با عمق‌های خواسته شده، می‌سازیم. لازم به ذکر است داده‌های train و test در سوال اول همین بخش ایجاد شدند و از آن‌ها در این بخش نیز استفاده می‌کنیم. پس از ساخت مدل‌ها نیز مقدار دقت یا همان accuracyها را در لیست‌هایی مجزا ذخیره می‌کنیم.

```
# for each depthwe construct the model and save the accuracies.
for depth in depths:
    model = DecisionTreeClassifier(criterion='gini', max_depth=depth, random_state=42)
    model.fit(X_train, y_train)

    train_acc = accuracy_score(y_train, model.predict(X_train))
    test_acc = accuracy_score(y_test, model.predict(X_test))

    train_accuracies.append(train_acc)
    test_accuracies.append(test_acc)
```

در ادامه، متناسب با خواسته سوال، نمودار عمق و دقت مربوطه را رسم می‌کنیم:




همانگونه که در نمودار مشهود است، مدل ما تا تا عمق۲، در هر دو بخش train و test، دقت خوبی دارد. اما از عمق ۳ به بعد، دقت پیشبینی train افزایش یافته و به یک میل می‌کند و در مقابل آن دقت پیشبینی test کاهش می‌یابد. یعنی از این عمق به بعد،

مدل گویی داده‌های train را حفظ کرده و در تلاش برای انطباق حداکثری است. در نتیجه این اقدام، مدل عملکرد و تعمیم‌پذیری خود را از دست می‌دهد و نمی‌تواند عملکرد مطلوبی در برابر داده‌های جدید داشته باشد. این یعنی از عمق ۳ به بعد دچار overfitting می‌شویم.

### ۳. (۵ نمره) مقایسه Gini و Entropy

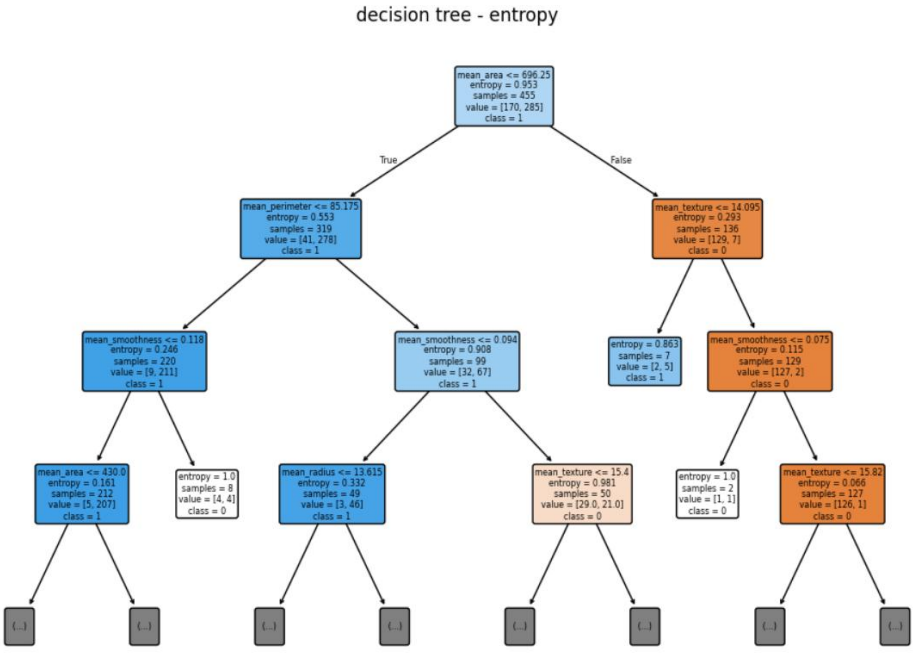
همان مدل سؤال ۱ را این بار با معیار تقسیم entropy بسازید. ساختار درخت و دقت آن را با مدل مبتنی بر gini مقایسه کنید.

تفاوت عمده در انتخاب ویژگی‌ها و عملکرد دو مدل چیست؟ آیا انتخاب معیار تقسیم تفاوت قابل‌توجهی ایجاد کرده است؟

برای ساخت مدل با معیار entropy، از داده‌های train و test سوال اول استفاده کرده و مدل موردنظر را می‌سازیم. لازم به ذکر است که باقی پارامترها را، به دلیل مقایسع آتی، مانند مدل gini مقداردهی کردیم.

```
# constructing tree based on entropy
clf_entropy = DecisionTreeClassifier(
    criterion='entropy',
    min_samples_split=10,
    random_state=42
)
clf_entropy.fit(X_train, y_train)
```

در ادامه درخت با معیار entropy را نیز رسم کردیم.



در ادامه باید تفاوت‌های عملکردی این دو مدل را بررسی کنیم. برای این کار، دقت مدل‌ها و ویژگی ریشه آنها را بررسی می‌کنیم.

```
{'gini accuracy': 0.8245614035087719,
 'gini root feature': 'mean_area',
 'entropy accuracy': 0.868421052631579,
 'entropy root feature': 'mean_area'}
```

نتایج به دست آمده در عکس بالا نمایان است. ویژگی ریشه برای هر دو معیار یکسان است و هر دو از mean_area استفاده می‌کنند. این نشان می‌دهد که در این مجموعه‌داده خاص، mean_area دارای بیشترین قدرت تفکیک کلاس‌ها است و هر دو معیار آن را به‌عنوان بهترین انتخاب تشخیص داده‌اند. بنابراین، انتخاب معیار تقسیم در این مثال خاص منجر به تفاوت ساختاری قابل‌توجهی در درخت نشد. اما دقت مدل با معیار entropy از مدل با معیار gini بیشتر است. بنابراین معیار entropy به ما دقت بیشتری می‌دهد. در نهایت شاهد تغییر قابل توجهی نیستیم ولی دقت مدل با entropy افزایش پیدا می‌کند.

۴. (۵ نمره) **ارزیابی متریک‌ها**

برای بهترین مدل به‌دست‌آمده از سؤال ۲، ماتریس سردرگمی را روی داده تست محاسبه کنید. سپس مقادیر زیر را محاسبه نمایید:

- دقت (Accuracy)
- دقت مثبت (Precision)
- حساسیت (Recall)
- ویژگی منفی (Specificity)
- امتیاز $F_1$

در این مسئله (تشخیص سرطان)، به نظر شما کدام متریک مهم‌تر است؟ چرا؟

در سوال دو متوجه شدیم که عمق بهینه درخت برابر ۲ است. بنابراین برای این مدل باید ماتریس سردرگمی را محاسبه کنیم.

```
# constructing the confusion matrix
cm = confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = cm.ravel()
```

در ادامه، معیارهای خواسته شده را محاسبه کردیم:

```
Confusion Matrix: [[33  9]
 [ 3 69]]
Accuracy: 0.895
Precision: 0.885
Recall (Sensitivity): 0.958
Specificity: 0.786
F1 Score: 0.92
```

در مسئله‌ی تشخیص سرطان (به‌ویژه تمایز بین تومور خوش‌خیم و بدخیم)، مهم‌ترین معیار، حساسیت (Recall) است. زیرا که این معیار میزان توانایی مدل در تشخیص سرطان را نشان می‌دهد. یعنی اگر مقدار این معیار پایین باشد، مدل ما نتوانسته بیماران را به درستی تشخیص دهد که همین مورد صدمات و هزینه‌های جبران‌ناپذیری را به ارمغان می‌آورد. یعنی باید تا حد ممکن تمامی موارد ابتلا را درست شناسایی کنیم. حتی اگر این رویه منجر به تشخیص اشتباه شود. زیرا هزینه تشخیص اشتباه کمتر از هزینه عدم شناسایی فرد مبتلا است.

### ۵. (۷ نمره) هرس درخت با Cost-Complexity Pruning

از روش cost-complexity pruning استفاده کرده و مدل‌هایی با مقادیر مختلف ccp_alpha ایجاد نمایید.

- از cross-validation برای انتخاب بهترین مقدار ccp_alpha استفاده کنید.
- نمودار تغییر دقت نسبت به ccp_alpha را رسم و مدل نهایی را انتخاب نمایید.
- ساختار مدل هرس‌شده را با مدل بدون هرس مقایسه کنید.

ابتدا یک درخت کامل را بدون اعمال محدودیت خاصی، می‌سازیم. سپس با استفاده از متد cost_complexity_pruning_path، مجموعه‌ای از مقادیر ccp alpha را به دست می‌آوریم. هر کدام از مقادیر ccp نشان‌دهنده این است که برای هرس هر گره چه مقدار جریمه باید بپردازیم تا آن هرس، بهینه شود. یعنی ccp عملاً معیاری برای تعادل بین دقت و سادگی مدل است.

```
clf = DecisionTreeClassifier(criterion='gini', random_state=42)
path = clf.cost_complexity_pruning_path(X_train, y_train)
```

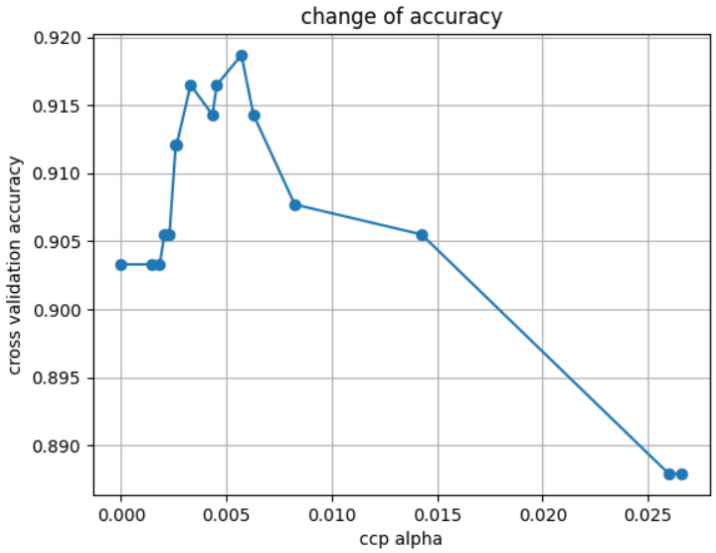
```
ccp_alphas = path.ccp_alphas[:-1]
```

در ادامه برای هر ccp_alpha یک مدل درخت مجزا می‌سازیم و برای ارزیابی آن از cross validation با ۵ fold استفاده می‌کنیم. یعنی داده آموزش به ۵ قسمت تقسیم شده و مدل روی ۴ قسمت آموزش داده شده، و روی قسمت پنجم ارزیابی می‌شود. این فرآیند ۵ بار تکرار می‌شود و میانگین دقت به‌دست می‌آید. این کار باعث می‌شود ارزیابی مدل پایدار، بی‌طرف و قابل‌اعتماد باشد.

```
alpha_scores = []
alpha_models = []

for alpha in ccp_alphas:
    clf_alpha = DecisionTreeClassifier(criterion='gini', random_state=42, ccp_alpha=alpha)
    scores = cross_val_score(clf_alpha, X_train, y_train, cv=5, scoring='accuracy')
    alpha_scores.append(scores.mean())
    alpha_models.append(clf_alpha)
```

نمودار تغییرات دقت نسبت به مقدار ccp_alpha نیز به شرح زیر است:




می‌دانیم که هر مدل ساخته شده یک میانگین از دقت از cross validation دارند و ما مدلی را که بیشترین دقت را داشته باشد به عنوان مدل بهینه انتخاب می‌کنیم:

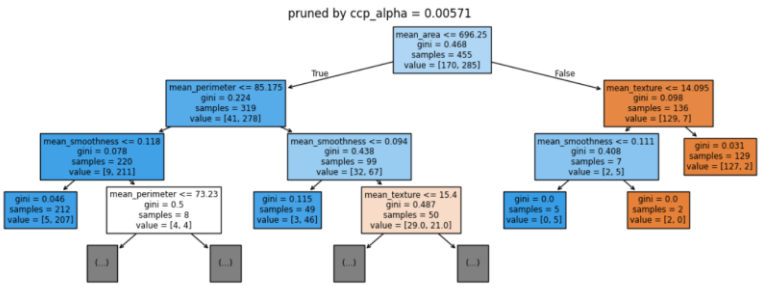
```
best_index = int(np.argmax(alpha_scores))
best_alpha = ccp_alphas[best_index]
print("best alpha:", best_alpha)
print("accuracy cross-validation:", alpha_scores[best_index])
```

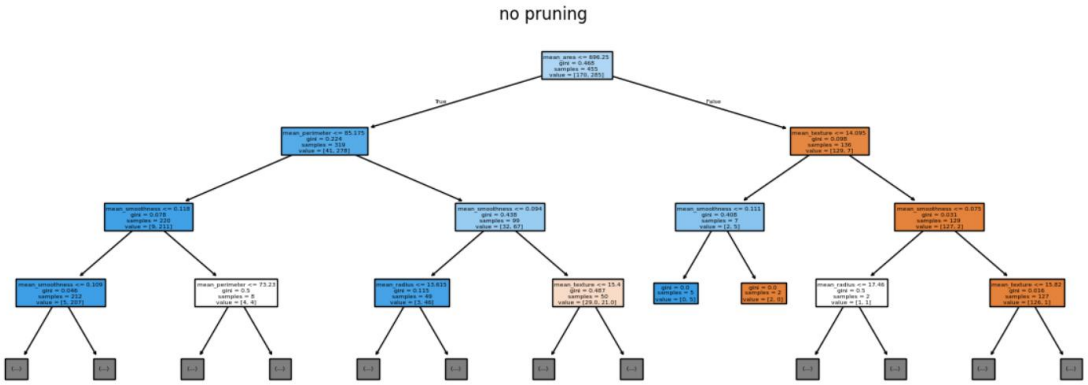
```
best alpha: 0.005709194280622852
accuracy cross-validation: 0.9186813186813186
```

با در دست داشتن مقدار بهینه ccp می‌توانیم درخت بهینه و موردنظر خودد را تشکیل دهیم.

```
pruned_model = DecisionTreeClassifier(criterion='gini', random_state=42, ccp_alpha=best_alpha)
pruned_model.fit(X_train, y_train)
```

برای مقایسه ساختار درخت هرس شده و درخت هرس نشده، ابتدا نمودارهای آنها را، به ترتیب ذکر شده، رسم می‌کنیم:



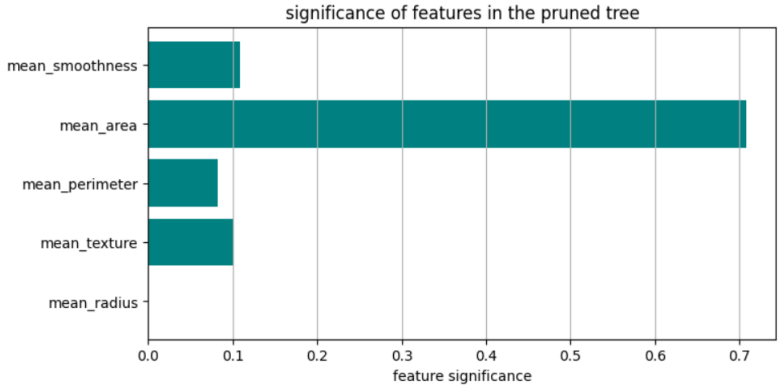



در رابطه با ساختار درخت‌ها، همانگونه که مشهود است، درخت هرس نشده شاخه‌های زیادی دارد و از چندین ویژگی مختلف در گره‌های مختلف استفاده کرده است. علاوه بر این بسیاری از برگ‌ها داده‌های بسیار کمی دارند که این خود نشانه‌ای از overfitting است. درخت هرس شده اما عمق محدودی دارد و فقط از مهم‌ترین ویژگی‌ها برای تقسیم استفاده کرده است. علاوه بر این گره‌های انتهایی معمولاً روی گروه‌های بزرگ‌تری از داده‌ها تصمیم‌گیری می‌کنند. در نهایت، درخت هرس‌شده، با حفظ عملکرد قابل قبول ساختاری ساده‌تر و قابل‌فهم‌تر دارد. این مدل نه‌تنها احتمال بیش‌برازش (Overfitting) را کاهش می‌دهد، بلکه در محیط‌های واقعی و حساس مثل پزشکی نیز قابل اعتمادتر است، زیرا تصمیمات آن شفاف‌تر و قابل بررسی هستند. در مقابل، مدل بدون هرس اگرچه ممکن است روی داده آموزش دقت بیشتری داشته باشد، اما به‌دلیل پیچیدگی زیاد، روی داده‌های جدید احتمالاً دچار افت عملکرد می‌شود.

## ۶. (۷ نمره) تحلیل اهمیت ویژگی‌ها

ویژگی‌های مهم مدل بهینه‌شده‌ی انتخاب‌شده در سؤال قبل را به‌دست آورده و به‌صورت نموداری نمایش دهید. آیا ویژگی‌های مهم با دیدگاه پزشکی در مورد سرطان قابل توجیه هستند؟ اگر آری، چگونه؟

مدل موردنظر در سوال قبل ساخته شد و در این قسمت از نتایج آن استفاده می‌کنیم. برای نشان دادن اهمیت ویژگی‌ها از خاصیت feature_importances_ در مدل استفاده می‌کنیم تا میزان سهم هر ویژگی در ساخت درخت تصمیم اندازه‌گیری شود.



همانگونه که مشهود است، mean_area بیشترین اهمیت را دارد. این نتیجه با دانش پزشکی سازگار است. زیرا که سلول‌های بدخیم معمولاً بزرگ‌تر و پُرحجم‌تر از سلول‌های خوش‌خیم هستند و افزایش سطح تومور (area) یک شاخص مستقیم از شدت بدخیمی است. در نهایت نقش این ویژگی در تصمیم‌های مدل بسیار پررنگ است. پس از mean_area، دو ویژگی mean_texture و mean_perimeter قرار دارند. Mean_texture تنوع در بافت داخلی توده را نشان می‌دهد و معمولاً در تومورهای بدخیم بافت داخلی ناهمگن‌تر است. mean_perimeter نیز محیط مرز سلول‌هاست که معمولاً در توده‌های بدخیم پیچیده‌تر و نامنظم‌تر است. پس از این سه ویژگی، mean_smoothness قرار دارد که با وجود اهمیت کم به دلیل نشان دادن میزان ناهمواری سطح سلول‌ها قابل توجه است. در انتها نیز ویژگی mean_raduis قرار دارد که گویی نقشی در مدل ایفا نمی‌کند. این رخداد را می‌توان به وجود mean_area و mean_perimeter نسبت داد. چون شعاع، محیط و مساحت در هندسه با هم مرتبط‌اند، مدل فقط یکی از آن‌ها را انتخاب می‌کند که بیشترین قدرت تفکیک را داشته باشد. به همین دلیل شعاع تا حد زیادی بدون استفاده می‌ماند.

### ۷. (۶ نمره) تحلیل آستانه تصمیم

با استفاده از قابلیت پیش‌بینی احتمال کلاس توسط درخت تصمیم، مدل را برای آستانه‌های تصمیم ۰/۳، ۰/۵، و ۰/۷ اجرا کنید. برای هر حالت، تعداد موارد False Positive و False Negative را محاسبه کرده و تحلیل نمایید.

اگر هدف جلوگیری از نادیده گرفتن بیماران بدخیم (کاهش FN) باشد، کدام آستانه مناسب‌تر است؟ چرا؟

برای این بخش نیز از مدل ایجاد شده در سوال ۵ استفاده می‌کنیم اما به جای متود predict از متود predict_proba استفاده می‌کنیم تا بتوانیم احتمالات موردنظر را به دست آوریم. متناسب با صورت سوال، با سه آستانه ذکر شده، مدل را ران کرده و تصمیم وجود یا عدم وجود سرطان را گرفتیم. در نهایت با استفاده از ماتریس سردرگمی مقادیر FP و FN را حساب کردیم.

```
thresholds = [0.3, 0.5, 0.7]
results = {}

for t in thresholds:
    y_pred_thresh = (y_proba >= t).astype(int)
    cm = confusion_matrix(y_test, y_pred_thresh)
    tn, fp, fn, tp = cm.ravel()
    results[t] = {
        "False Positive": int(fp),
        "False Negative": int(fn)
    }

results
```

در ادامه، خروجی نهایی متناسب با هر آستانه به شرح زیر است:

```
{0.3: {'False Positive': 3, 'False Negative': 13},
 0.5: {'False Positive': 3, 'False Negative': 13},
 0.7: {'False Positive': 3, 'False Negative': 13}}
```

همانگونه که مشهود است، تغییر مقدار آستانه هیچ تغییری در خروجی مدل ایجاد نکرد. یعنی مدل در تمام حالات، سیزده نمونه بدخیم را تشخیص نداده (FN=13) و سه نمونه خوش‌خیم را اشتباه به‌عنوان بدخیم (FP=3) تشخیص داده است. این پیشامد می‌تواند به علت این باشد که احتمال کلاس‌ها آن‌قدر نزدیک به صفر یا یک بوده‌اند که آستانه‌های ۰.۳ تا ۰.۷ تغییری ایجاد نکردند. لازم به ذکر است که در کاربردهایی مثل تشخیص سرطان پستان، مهم‌ترین هدف کاهش خطای نوع دوم (False Negative) است. یعنی اطمینان از اینکه هیچ بیمار بدخیمی به اشتباه «سالم» تشخیص داده نشود. بنابراین بهتر است آستانه‌های کمتر از ۰.۳ را نیز بررسی کنیم. زیرا که چون مدل با حساسیت بیشتر عمل می‌کند و موارد بیشتری را مشکوک تشخیص می‌دهد.

---

**بخش ۲: Naïve Baysian** **دانلود داده این بخش**

---

۱. (۱۲ نمره) درستی یا نادرستی موارد زیر را با ذکر دلیل مشخص نمایید.

(آ) بیز ساده‌لوحانه با فرض استقلال مستقیم بین متغیرها گسترش می‌یابد و به همین دلیل «ساده‌لوحانه» لقب گرفته‌است.

(ب) مدل‌های پایه‌ای بیزساده‌لوحانه می‌توانند بدون تغییری با ویژگی‌های پیوسته و گسسته کار کنند.

(ج) با استفاده از بیزساده‌لوحانه برای طبقه‌بندی، این الگوریتم صراحتاً توزیع مشترک $P(X,Y)$ را بر روی ویژگی‌ها و برچسب‌ها مدل می‌کند و سپس با استفاده از قاعدهٔ بیز، $P(Y \mid X)$ را محاسبه می‌کند.

آ) نادرست است. الگوریتم بیز ساده‌لوحانه (Naïve Bayes) بر یک فرض اساسی و ساده‌کننده استوار است: **استقلال شرطی ویژگی‌ها نسبت به یکدیگر، به شرط مشخص بودن کلاس**. این فرض که اغلب در واقعیت برقرار نیست، به الگوریتم لقب "ساده‌لوحانه" را داده است؛ زیرا پیچیدگی روابط بین ویژگی‌ها را نادیده می‌گیرد و آن‌ها را مستقل از هم فرض می‌کند. این سادگی محاسبات را آسان‌تر می‌کند اما ممکن است بر دقت مدل در برخی مسائل تأثیر بگذارد.

ب) نادرست است. مدل‌های پایه‌ای بیز ساده‌لوحانه برای کار با ویژگی‌های پیوسته و گسسته نیاز به تغییراتی دارند. برای ویژگی‌های گسسته، احتمالات شرطی با شمارش فراوانی‌ها (مثلاً با استفاده از MLE یا MAP ) محاسبه می‌شوند. اما برای ویژگی‌های پیوسته، نیاز است که این ویژگی‌ها با استفاده از توزیع‌های احتمال مانند توزیع گاوسی مدل‌سازی شوند، یا به دسته‌های گسسته تبدیل گردند (Discretization). بنابراین، مدل پایه نمی‌تواند "بدون تغییری" با هر دو نوع ویژگی کار کند.

ج) نادرست است. بیز ساده‌لوحانه یک طبقه‌بند مولد (Generative Classifier) است. به این معنی که سعی می‌کند توزیع مشترک $P(X,Y)$ را مدل کند. با این حال، به جای مدل کردن صریح $P(X,Y)$، این الگوریتم ابتدا توزیع پیشین کلاس $P(Y)$ و توزیع‌های شرطی ویژگی‌ها $P(X|Y)$ را مدل می‌کند. سپس، با استفاده از قاعده بیز، احتمال پسین $P(Y|X)$ را محاسبه می‌کند. قاعده بیز به صورت $P(Y \mid X) = \frac{P(Y|X)P(Y)}{P(X)}$ بیان می‌شود. بنابراین، الگوریتم به طور غیرمستقیم و با فرض استقلال ویژگی‌ها، به توزیع مشترک دست می‌یابد و سپس $P(Y|X)$ را استخراج می‌کند، نه اینکه آن را به طور صریح مدل کند.

۲. (۲۵ نمره) قصد به دسته‌بندی اعداد دست‌نویس داریم و هر تصویر هر عدد را برای پردازش، به تصویری در ابعاد ۸ × ۸ تبدیل می‌کنیم و احتمال روشن و خاموش بودن هر پیکسل را محاسبه می‌کنیم.

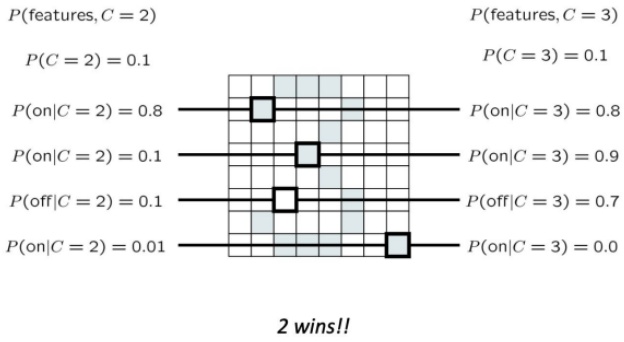
(آ) ادعا می‌شود «با افزایش تعداد پیکسل‌ها در هر بعد به منظور مدل‌سازی، هزینه‌های محاسباتی افزایش می‌یابد اما مدل به طور متوسط نتایج بهتری بر روی داده‌های اعتبارسنجی و آزمون نسبت به مدل‌سازی قبلی کسب خواهد کرد». درستی یا نادرستی این عبارت را مشخص کرده و در صورت درستی آن، علت این بهبود را شرح داده و در صورت نادرستی، بر روی قسمت(های) نادرست آن بحث کنید.

این عبارت صحیح است. با افزایش تعداد پیکسل‌ها در هر بعد یک تصویر، هزینه‌های محاسباتی برای مدل‌سازی به طور قابل توجهی افزایش می‌یابد. این افزایش هزینه ناشی از بزرگ‌تر شدن ابعاد فضای ویژگی است؛ برای مثال، یک تصویر ۸×۸ دارای ۶۴ پیکسل است، در حالی که یک تصویر ۱۶×۱۶ شامل ۲۵۶ پیکسل است که هر کدام به عنوان یک ویژگی در مدل به حساب می‌آیند. این گسترش در ابعاد به پیچیدگی محاسباتی در هر دو فاز آموزش و پیش‌بینی می‌افزاید. در مرحله آموزش، نیاز به محاسبه و ذخیره‌سازی احتمالات شرطی بیشتری برای هر پیکسل و هر کلاس وجود دارد. در فاز پیش‌بینی نیز، برای هر نمونه جدید، عملیات بیشتری برای ترکیب احتمالات شرطی تمامی ویژگی‌ها لازم است که زمان پیش‌بینی را افزایش می‌دهد. همچنین، افزایش ابعاد داده‌ها به معنای نیاز به حافظه بیشتر برای ذخیره‌سازی ماتریس‌های احتمالات و خود داده‌های آموزشی است.

با این حال، این افزایش هزینه با بهبود نتایج مدل در داده‌های اعتبارسنجی و آزمون همراه است، که این ادعا را به طور کلی صحیح می‌کند. دلیل این بهبود در افزایش محتوای اطلاعاتی و تفکیک‌پذیری بهتر الگوها نهفته است. رزولوشن بالاتر تصاویر، جزئیات مکانی بیشتری را فراهم می‌کند؛ این جزئیات می‌توانند شامل خطوط ظریف‌تر، منحنی‌های دقیق‌تر و سایر ویژگی‌های ریز باشند که در رزولوشن پایین‌تر قابل مشاهده یا تمایز نیستند. این اطلاعات غنی‌تر به مدل کمک می‌کند تا الگوهای پیچیده‌تر و خاص‌تر هر عدد دست‌نویس را با دقت بیشتری یاد بگیرد. در نتیجه، مدل قادر خواهد بود ابهام را کاهش داده و مرزهای تصمیم‌گیری دقیق‌تری در فضای ویژگی بیاموزد، که به "قابلیت تعمیم (generalization ability) " بهتر آن بر روی داده‌های دیده نشده منجر می‌شود. این نمایش غنی‌تر، فضای وسیع‌تری برای بیان تفاوت‌های ظریف بین نمونه‌ها فراهم می‌آورد و به الگوریتم‌های یادگیری ماشین امکان می‌دهد ویژگی‌های متمایزکننده‌تر را شناسایی کرده و عملکرد طبقه‌بندی را بهبود بخشند.

شایان ذکر است که این بهبود لزوماً به صورت خطی ادامه نمی‌یابد و در نقطه‌ای ممکن است به "بازده نزولی" منجر شود، به طوری که افزایش بیشتر در ابعاد تنها افزایش ناچیزی در دقت به همراه داشته باشد در حالی که هزینه‌های محاسباتی همچنان رو به رشد است. همچنین، پدیده‌ای به نام "نفرین ابعاد" می‌تواند در ابعاد بسیار بالا رخ دهد که در آن داده‌ها به شدت پراکنده شده و یافتن الگوهای معنی‌دار دشوارتر می‌شود. با این حال، در مورد افزایش از ۸×۸ به ۱۶×۱۶، معمولاً این افزایش ابعاد در محدوده‌ای قرار دارد که منجر به بهبود عملکرد می‌شود.

(ب) مثالی از احتمالات محاسبه‌شده برای یک نمونه ورودی در شکل ۱ آمده‌است. این تصویر اشاره به کدام عیب بیزساده‌لوحانه دارد؟




شکل ۱: نمونه احتمالات برای یک ورودی

تصویر ارائه شده در شکل ۱، به عیب اصلی و بنیادین الگوریتم بیز ساده‌لوحانه، یعنی "فرض استقلال شرطی ویژگی‌ها " (Conditional Independence Assumption of Features) اشاره دارد. در الگوریتم بیز ساده‌لوحانه، فرض می‌شود که ویژگی‌ها (در اینجا، روشن/خاموش بودن هر پیکسل) نسبت به یکدیگر مستقل هستند به شرطی که کلاس (در اینجا، عدد دست‌نویس) معلوم باشد. به عبارت ریاضی، برای یک نمونه ورودی با ویژگی‌های $X=\{x1,x2,\ldots,xn\}$ و یک کلاس C مدل بیز ساده‌لوحانه احتمال شرطی $P(X|C)$ را به صورت حاصل‌ضرب احتمالات شرطی هر ویژگی به طور جداگانه محاسبه می‌کند. در مثال شکل یک فرض کنید این تصویر یک عدد دست‌نویس را نشان می‌دهد و ما می‌خواهیم آن را به عنوان ۲ یا ۳ طبقه‌بندی کنیم. این تصویر یک شبکه پیکسلی را نشان می‌دهد و برای هر پیکسل، احتمال روشن (on) یا خاموش (off) بودن آن پیکسل، به شرطی که عدد مورد نظر ۲ یا ۳ باشد، محاسبه شده است. مشکلی که در تصویر و محاسبات ارائه شده، به ویژه در سطر آخر (پایین‌ترین پیکسل مشخص شده) نمایان می‌شود:

- برای کلاس $C=2$: احتمال روشن بودن آن پیکسل $P(on|C=2)=0.01$ است.
- برای کلاس $C=3$: احتمال روشن بودن آن پیکسل $P(on|C=3)=0.0$ است.

وجود احتمال صفر برای یک ویژگی خاص (پیکسل مشخص شده در پایین) در یک کلاس خاص$(C=3)$ ، یک مشکل جدی برای بیز ساده‌لوحانه ایجاد می‌کند. بر اساس فرض استقلال، اگر یک یا چند احتمال شرطی $P(xi|C)$ برابر با صفر باشند، آنگاه حاصل‌ضرب کلی $P(X|C)$ نیز برای آن کلاس صفر خواهد شد. وقتی $P(X|C)$ برای یک کلاس صفر می‌شود، به این معنی است که مدل هرگز نمی‌تواند نمونه ورودی مورد نظر را به آن کلاس اختصاص دهد، حتی اگر سایر ویژگی‌ها به شدت از آن کلاس حمایت کنند. در این مثال، اگر یکی از پیکسل‌ها برای عدد ۳ دارای احتمال روشن بودن صفر باشد، حتی اگر بقیه ۹۹ پیکسل (در یک تصویر ۱۰x۱۰ فرضی) کاملاً شبیه به یک ۳ باشند، احتمال نهایی $P(features,C=3)$ برای کلاس ۳ به دلیل این یک پیکسل صفر خواهد شد. این منجر به این می‌شود که مدل حتی در صورت وجود شواهد قوی دیگر، هیچگاه کلاس ۳ را انتخاب نکند. در نتیجه، الگوریتم به طور نادرستی فقط به یک کلاس خاص (در اینجا ۲) امتیاز بالاتری می‌دهد، همانطور که 2 wins!! را در تصویر نشان می‌دهد.

(ج) به منظور رفع مشکل بند قبلی، از هموارسازی لاپلاس[1] استفاده می‌شود که به صورت زیر تعریف می‌گردد:

$$P_{\mathrm{LAP},k}(x) = \frac{c(x) + k}{N + k\,|X|} \qquad (۱)$$

هرکدام از عناصر این رابطه چه چیزی را نشان می‌دهند؟ نقش $k$ را به طور کامل توضیح دهید و با افزایش یا کاهش آن چه اتفاقی می‌افتد؟

هموارسازی لاپلاس (Laplace Smoothing)، که گاهی اوقات به آن Additive Smoothing نیز گفته می‌شود، یک تکنیک برای تعدیل تخمین‌های احتمال در مدل‌های آماری مانند بیز ساده‌لوحانه است. هدف اصلی آن جلوگیری از مشکل "احتمال صفر " (Zero Probability Problem) است که در آن، اگر یک رویداد خاص (مثلاً مشاهده یک ویژگی خاص) در داده‌های آموزشی اتفاق نیفتاده باشد، احتمال آن صفر تخمین زده می‌شود. این صفر شدن می‌تواند منجر به نتایج نامطلوب در محاسبات کلی احتمال شود. در فرمول $P_{LAP,k}(x) = \frac{c(x)+k}{N+k|X|}$ :

- $P_{LAP,k}(x)$: نشان‌دهنده احتمال تعدیل‌شده یا هموار شده[1] مشاهده ویژگی x (به عنوان مثال، روشن بودن یک پیکسل خاص در یک تصویر) . این احتمال پس از اعمال هموارسازی لاپلاس به دست می‌آید.
- $c(x)$: نمایانگر تعداد دفعات مشاهده ویژگی x در مجموعه داده آموزشی، برای یک کلاس خاص. به عنوان مثال، اگر در حال محاسبه احتمال روشن بودن پیکسل (i,j) برای کلاس ۲ باشیم، c(x) تعداد تصاویری از عدد ۲ است که در آن‌ها پیکسل (i,j) روشن بوده است.
- $k$: این پارامتر یک شبه‌شمارش[2] است. k یک مقدار مثبتی است که به صورت مصنوعی به شمارش هر ویژگی اضافه می‌شود. هدف اصلی آن اطمینان از این است که هیچ ویژگی، حتی اگر در داده‌های آموزشی مشاهده نشده باشد (c(x)=0) احتمال صفر نداشته باشد. مقدار رایج برای k، عدد ۱ است که به آن هموارسازی افزایشی یک می‌گویند.
- $N$: نمایانگر تعداد کل مشاهدات یا تعداد کل نمونه‌ها در مجموعه داده آموزشی. به عنوان مثال، اگر برای کلاس ۲ محاسبه می‌کنیم، N تعداد کل تصاویری از عدد ۲ در مجموعه آموزشی است.
- $|X|$: این ترم نشان‌دهنده تعداد مقادیر ممکن برای ویژگی x است. برای مثال، در مورد پیکسل‌های روشن/خاموش در تصویر، هر پیکسل دو حالت ممکن دارد (روشن یا خاموش)؛ بنابراین $|X| = 2$. اگر یک ویژگی رنگ مو باشد که می‌تواند مشکی، قهوه‌ای یا بلوند باشد، آنگاه $|X| = 3$. این مقدار در مخرج ضرب در k می‌شود تا اطمینان حاصل شود که مخرج نیز به درستی مقیاس‌بندی شده است.

k نقش حیاتی در تنظیم میزان هموارسازی دارد. همانطور که اشاره شد، هدف اصلی k اطمینان از این است که هیچ احتمالی صفر نمی‌شود. با اضافه کردن k به صورت و k|X| به مخرج، حتی اگر c(x) صفر باشد، احتمال $P_{LAP,k}(x)$ حداقل به $\frac{k}{N+k|X|}$ می‌رسد که یک عدد مثبت کوچک است. این کار از مشکل "صفر شدن کامل" حاصل‌ضرب احتمالات در بیز ساده‌لوحانه جلوگیری می‌کند و تضمین می‌کند که هیچ کلاسی به دلیل یک ویژگی خاص به طور کامل حذف نشود.

[1] smoothed probability
[2] pseudocount

تأثیر افزایش k:

- افزایش هموارسازی یا (Stronger Smoothing): با افزایش مقدار k، وزن "شبه‌شمارش‌ها" در فرمول افزایش می‌یابد. این به این معنی است که احتمالات تخمین‌زده شده بیشتر به سمت یک توزیع یکنواخت سوق داده می‌شوند، یعنی تفاوت بین احتمالات ویژگی‌های مختلف کمتر می‌شود، مدل کمتر به داده‌های آموزشی خاص وابسته می‌شود و بیشتر به فرض یکنواختی تمایل پیدا می‌کند.
- کاهش واریانس و جلوگیری از بیش‌برازش: افزایش k باعث می‌شود مدل کمتر تحت تأثیر نویز یا ویژگی‌های بسیار نادر، (که ممکن است تصادفی باشند)، در داده‌های آموزشی قرار گیرد. با هموارتر کردن احتمالات، مدل کمتر جزئیات بیش از حد خاص داده‌های آموزشی را یاد می‌گیرد و در نتیجه، می‌تواند به جلوگیری از بیش‌برازش کمک کند و مدل ممکن است در داده‌های جدید عملکرد پایدارتری داشته باشد.
- افزایش بایاس و خطر کم‌برازش: با این حال، هموارسازی بیش از حد (با k بسیار بزرگ) می‌تواند منجر به نادیده گرفتن الگوها و تفاوت‌های واقعی و مهم موجود در داده‌های آموزشی شود. این امر می‌تواند منجر به افزایش بایاس مدل شود، به این معنی که مدل به اندازه کافی پیچیده نیست تا رابطه واقعی بین ویژگی‌ها و کلاس‌ها را درک کند و در نهایت، ممکن است دقت کلی مدل (به ویژه در داده‌های تست) کاهش یابد، که به آن کم‌برازش می‌گویند.

تأثیر کاهش k:

- کاهش هموارسازی: با کاهش مقدار k تأثیر شبه‌شمارش‌ها کمتر می‌شود و مدل بیشتر به داده‌های مشاهده‌شده واقعی، یعنی $c(x)$ و $N$، وابسته می‌شود.
- نزدیک شدن به تخمین بیشینه درست‌نمایی (MLE): هنگامی که $k=0$ باشد، فرمول هموارسازی لاپلاس دقیقاً به فرمول تخمین بیشینه درست‌نمایی (MLE) بازمی‌گردد. در این حالت، اگر $c(x)$ صفر باشد، $P(x)$ نیز صفر خواهد شد و مشکل احتمال صفر که هموارسازی لاپلاس برای رفع آن طراحی شده بود، دوباره ظاهر می‌شود.

- افزایش واریانس و خطر بیش‌برازش: کاهش k باعث می‌شود مدل به جزئیات و نویز موجود در داده‌های آموزشی حساس‌تر شود. این می‌تواند منجر به بیش‌برازش شود، به این معنی که مدل الگوهای بسیار خاص و احتمالاً نویز موجود در داده‌های آموزشی را حفظ می‌کند و در نتیجه در داده‌های جدید عملکرد ضعیفی از خود نشان می‌دهد.

- کاهش بایاس: در صورتی که k کوچک باشد، مدل می‌تواند جزئیات بیشتری از داده‌های آموزشی را یاد بگیرد، که می‌تواند منجر به کاهش بایاس شود. اما همانطور که ذکر شد، این موضوع در معرض خطر بالای بیش‌برازش قرار دارد.

انتخاب مقدار مناسب برای kیک گام حیاتی در تنظیم هایپرپارامتر مدل بیز ساده‌لوحانه است. این انتخاب یک تعادل بین بایاس و واریانس را ایجاد می‌کند. یک k کوچک ممکن است به بیش‌برازش و حساسیت بالا به نویز منجر شود، در حالی که یک k بزرگ می‌تواند باعث کم‌برازش و از دست دادن اطلاعات مفید در داده‌ها شود. معمولاً مقدار بهینه k از طریق روش‌هایی مانند اعتبارسنجی متقابل (Cross-validation) بر روی مجموعه داده اعتبارسنجی تعیین می‌شود تا بهترین عملکرد تعمیم‌پذیری مدل حاصل شود.

(د) مدل پایه‌ای بیزساده‌لوحانه با استفاده از تخمین بیشینه درست‌نمایی (MLE)[۲] احتمالات را محاسبه می‌کند. اصلاحات بند قبل، ما را به استفاده از تخمین بیشینه پسین (MAP)[۳] می‌رساند. نحوهٔ کار تخمین‌گر جدید را شرح داده و آن را با تخمین‌گر اولیه مقایسه کنید.

مدل پایه‌ای بیز ساده‌لوحانه، احتمالات مورد نیاز خود را، از جمله احتمال هر کلاس (P(C)) و احتمالات ویژگی‌های شرطی (P(xi|C))، با استفاده از تخمین بیشینه درست‌نمایی (MLE) محاسبه می‌کند. درMLE ، هدف یافتن پارامترهایی است که احتمال مشاهده داده‌های آموزشی را حداکثر کنند؛ به عبارت دیگر، پارامترهایی که بهترین برازش را با داده‌های موجود نشان دهند. برای محاسبه احتمال یک کلاس خاص، تعداد نمونه‌های آن کلاس بر تعداد کل نمونه‌ها در داده‌های آموزشی تقسیم می‌شود. به همین ترتیب، احتمال یک ویژگی خاص (مثلاً روشن بودن یک پیکسل) به شرط یک کلاس، از تقسیم تعداد دفعاتی که آن ویژگی در نمونه‌های آن کلاس مشاهده شده، بر تعداد کل نمونه‌های آن کلاس به دست می‌آید. با این حال، مشکل اصلی MLE در بیز ساده‌لوحانه، مشکل احتمال صفر است. اگر یک ترکیب خاص از ویژگی و کلاس هرگز در داده‌های آموزشی مشاهده نشود، MLE احتمال آن را صفر تخمین می‌زند. این صفر شدن می‌تواند منجر به آن شود که حاصل‌ضرب کلی احتمالات برای آن کلاس (که در بیز ساده‌لوحانه از ضرب احتمالات شرطی ویژگی‌ها به دست می‌آید) نیز به طور کامل صفر شود. در نتیجه، مدل هرگز آن کلاس را پیش‌بینی نخواهد کرد، حتی اگر شواهد دیگر به شدت از آن کلاس حمایت کنند. مثالی از این مشکل در شکل ۱ نشان داده شده است، جایی که احتمال روشن بودن یک پیکسل برای کلاس ۳ برابر 0.0 است، که باعث می‌شود حتی اگر سایر ویژگی‌ها به شدت به سمت کلاس ۳ متمایل باشند، آن کلاس نادیده گرفته شود.

برای رفع این مشکل، هموارسازی لاپلاس (Laplace Smoothing) معرفی می‌شود که در واقع یک روش پیاده‌سازی تخمین بیشینه پسین (MAP) است. تخمینMAP علاوه بر داده‌های مشاهده شده، دانش پیشین را نیز در فرآیند تخمین پارامترها دخیل می‌کند. . در فرمول هموارسازی لاپلاس $P_{LAP,k}(x) = \frac{c(x)+k}{N+k|X|}$، نقس اصلی k جلوگیری از صفر شدن احتمالات است. با افزودن k>0 به صورت‌کسر، حتی اگر یک ویژگی هیچ‌گاه در داده‌های آموزشی دیده نشده باشد (c(x)=0)، احتمال آن صفر نخواهد شد، بلکه به یک مقدار مثبت کوچک $\frac{k}{N+k|X|}$ می‌رسد. این کار تضمین می‌کند که حاصل‌ضرب احتمالات هرگز به طور کامل صفر نشود و مدل بتواند تمامی کلاس‌ها را در نظر بگیرد.

| ویژگی مقایسه | تخمین اولیه (MLE) | تخمین‌گر جدید (MAP با لاپلاس) |
|---|---|---|
| **روش محاسبه** | صرفاً بر اساس فراوانی‌های مشاهده شده در داده‌های آموزشی. | بر اساس فراوانی‌های مشاهده شده و یک "دانش پیشین" (pseudocount $k$). |
| **مشکل احتمال صفر** | بله، در صورت عدم مشاهده رویداد، احتمال آن صفر می‌شود. | خیر، با افزودن $k > 0$، هیچ احتمالی صفر نمی‌شود. |
| **حساسیت به داده‌های نادر** | بسیار حساس؛ حتی یک مشاهده نادر می‌تواند تأثیر زیادی داشته باشد. | کمتر حساس؛ "شبه‌شمارش‌ها" اثر داده‌های نادر را تعدیل می‌کنند. |
| **پایداری مدل** | کمتر پایدار، به ویژه با مجموعه داده‌های کوچک. | پایدارتر، به خصوص در مواجهه با داده‌های جدید یا دیده نشده. |
| **بایاس و واریانس** | بایاس کمتر اما واریانس بالا (حساس به نویز). | بایاس بیشتر (به دلیل اطلاعات پیشین) اما واریانس کمتر (مدل "هموارتر" است). |
| **قابلیت تعمیم** | در صورت وجود احتمالات صفر، عملکرد ضعیفی در داده‌های جدید ممکن است داشته باشد. | قابلیت تعمیم بهبود یافته، زیرا می‌تواند رویدادهای دیده نشده را مدیریت کند. |

(ه) با استفاده از تخمین بیشینه درست‌نمایی و تخمین بیشینه پسین به ازای $k = ۰, ۱, ۱۰۰$ احتمال قرمز و آبی بودن را در شکل ۲ محاسبه نمایید.


شکل ۲: شکل مربوط به بند «ه»

در شکل ۲، ما سه نمونه داریم: دو نمونه قرمز (r) و یک نمونه آبی (b). بنابراین تعداد کل نمونه‌ها، یعنی N، برابر با ۳ است. تعداد مقدارهای ممکن برای این ویژگی، |X|، نیز ۲ است؛ قرمز و آبی. در ادامه احتمال قرمز و آبی بودن را باید برای kهای ۰ و ۱ و ۱۰۰ حساب کنیم. لازم به ذکر است که MLE ربطی به مقدار k ندارد و زمانی که k برابر صفر باشد، فرمول MLE و MAP با یکدیگر برابر می‌شود:

۱) محاسبه با استفاده از تخمین بیشینه درست‌نمایی - (MLE) معادل k=0:

فرمول MLE به صورت $P(x) = \frac{c(x)}{N}$ است. حال:

احتمال قرمز بودن:

- تعداد مشاهده قرمز: $c(r) = 2$
- $P(r) = \frac{2}{3} = 0.667$

احتمال آبی بودن:

- تعداد مشاهده آبی: $c(b) = 1$
- $P(b) = \frac{1}{3} = 0.333$

۲) محاسبه با استفاده از تخمین بیشینه پسین MAP با هموارسازی لاپلاس k=1:

فرمول $P_{LAP,k}(x) = \frac{c(x)+k}{N+k|X|}$. در اینجا k=1 و |X|=2 و N=3.

احتمال قرمز بودن:

- تعداد مشاهده قرمز: $c(r) = 2$
- $P_{LAP,1}(r) = \frac{2+1}{3+1\times2} = \frac{3}{5} = 0.6$

احتمال آبی بودن:

- تعداد مشاهده آبی: $c(b) = 1$
- $P_{LAP,1}(b) = \frac{1+1}{3+1\times2} = \frac{2}{5} = 0.4$

۳) محاسبه با استفاده از تخمین بیشینه پسین MAP با هموارسازی لاپلاس k=100:

فرمول $P_{LAP,k}(x) = \frac{c(x)+k}{N+k|X|}$. در اینجا k=100 و |X|=2 و N=3.

احتمال قرمز بودن:

- تعداد مشاهده قرمز: $c(r) = 2$
- $P_{LAP,1}(r) = \frac{2+100}{3+100\times2} = \frac{102}{203} = 0.50246$

احتمال آبی بودن:

- تعداد مشاهده آبی: $c(b) = 1$
- $P_{LAP,1}(b) = \frac{1+100}{3+100\times2} = \frac{101}{203} = 0.49754$

هنگامی که k=0، احتمال قرمز بیشتر است و احتمال آبی کمتر است که مستقیماً منعکس‌کننده فراوانی‌های مشاهده شده در داده است. با افزایش k، احتمالات به سمت یک توزیع یکنواخت‌تر (یعنی نزدیک‌تر به ۰.۵) هموار می‌شوند. این به دلیل شبه‌شمارش‌هایی است که به هر دسته اضافه می‌شود، و باعث می‌شود مدل کمتر به فراوانی‌های مشاهده شده خاص در داده‌های کوچک حساس باشد. در k=100، P(r) و P(b) تقریباً برابر با ۰.۵ هستند که نشان‌دهنده یک هموارسازی بسیار قوی است و فرض می‌کند که هر دو رنگ تقریباً به یک اندازه محتمل هستند حتی اگر در داده‌های اصلی تعداد قرمزها دو برابر آبی‌ها باشد. این مثال به وضوح تأثیر k را بر میزان هموارسازی و تغییر احتمالات از تخمین صرفاً مبتنی بر فراوانی به سمت توزیع یکنواخت‌تر نشان می‌دهد.

۳. (۲۴ نمره) در سال‌های اخیر، ارسال پیام‌های اسپم در رسانه‌های الکترونیکی به معضلی فزاینده تبدیل شده‌است. در این مسئله، از الگوریتم بیزساده‌لوحانه به منظور ساخت یک دسته‌بند اسپم استفاده خواهیم کرد. مجموعه دادهٔ مورد بررسی، یک مجموعه دادهٔ واقعی و پژوهشی می‌باشد که توسط دانشگاه کمپیناس در برزیل جمع‌آوری شده‌است. جداسازی داده‌ها انجام‌شده و از جداسازی مجدد داده‌ها عمیقاً خودداری فرمایید. توضیحات بیشتر داده‌ها در فایل `readme.txt` موجود می‌باشد. همچنین شما تنها مجاز به استفاده از کتابخانه‌های مذکور در فایل `spam.py` می‌باشید.

(آ) در این بند، باید کدی بنویسید که داده‌های پیام‌ها را پردازش کرده و به آرایه‌های نامپای تبدیل کند تا بتوان آن‌ها را به مدل یادگیری ورودی داد. به همین منظور، توابع `create_dictionary`، `get_words` و `transform_text` را تکمیل نمایید. توجه داشته باشید که برای هر تابع، توضیحاتی به صورت کامنت در کد داده‌شده‌اند که مشخص می‌کنند دقیقاً چه نوع پردازشی باید انجام شود.

پیش از هر کاری، برای آنکه بتوانیم از تابع util استفاده کنیم، فایل موجود را در محیط گوگل کولب بارگذاری می‌کنیم. علاوه بر این، کد موجود در فایل spam را نیز کپی کرده و در سلول اول نوت‌بوک قرار می‌دهیم. در ادامه باید با پر کردن توابع ذکر شده داده‌های متنی خام، یا همان SMS، را به یک فرم عددی قابل استفاده تبدیل کنیم.

ابتدا تابع get_words را تکمیل می‌کنیم. در این تابع باید لیستی از کلمات را از متن پیام، با نرمال‌سازی و جداسازی بر حسب فاصله، به دست بیاوریم. برای این کار ابتدا پیام را به حروف کوچک تبدیل کرده و سپس آنها را جدا می‌کنیم.

```
return message.lower().split()
```

در ادامه تابع create_dictionary را کامل می‌کنیم. برای این کار ابتدا یک دیکشنری تعریف می‌کنیم که در ابتدا مقدار هر کلید را برابر با صفر در نظر می‌گیرد. سپس هر پیام به مجموعه‌ای از کلمات تبدیل می‌شود و متناسب با وجود کلمه، به value هر کلید افزوده می‌شود.

```
word_count = collections.defaultdict(int)

for message in messages:
    words = set(get_words(message))  # فقط یکبار برای هر پیام
    for word in words:
        word_count[word] += 1
```

پس از به دست آوردن کلمات و مقدار تکرارشان، تنها کلماتی که حداقل ۵ بار تکرار شده‌اند را نگه می‌داریم.

```
dictionary = {}
for word, count in word_count.items():
    if count >= 5:  # آستانه تعریفشده: حداقل در ۵ پیام ظاهر شده باشد
        dictionary[word] = len(dictionary)
```

سپس به سراغ تابع transform_text می‌رویم. در ابتدا یک ماتریس صفر تشکیل می‌دهیم که تعداد سطرهای آن با تعداد پیام‌ها و تعداد ستون‌های آن با تعداد کلمات پرتکرار برابر است. در ادامه، پیام‌های ورودی را به کلمات تجزیه می‌کنیم. اگر آن کلمات جزء کلمات پرتکرار باشند، به مقدار آن کلمه در ماتریس یک واحد اضافه می‌کنیم.

```
matrix = np.zeros((len(messages), len(word_dictionary)), dtype=int)

for i, message in enumerate(messages):
    words = get_words(message)
    for word in words:
        if word in word_dictionary:
            j = word_dictionary[word]
            matrix[i, j] += 1
```

(ب) حال یک دسته بند بیزساده‌لوحانه را با استفاده از مدل رویداد چندجمله‌ای[۴] و هموارسازی لاپلاس پیاده‌سازی کنید. کافی است توابع `fit_naive_bayes_model` و `predict_from_naive_bayes_model` را تکمیل نمایید. پس از تکمیل، فایل `spam.py` باید بتواند یک مدل بیزساده‌لوحانه را آموزش دهد و دقت پیش‌بینی شما را محاسبه نماید.

**نکتهٔ مهم:** اگر بیزساده‌لوحانه را به روش مستقیم پیاده‌سازی کنید، متوجه خواهید شد که مقدار $P(x \mid y) = \prod_i P(x_i \mid y)$ اغلب برابر صفر می‌شود. این به این دلیل است که حاصل‌ضرب تعداد زیادی عدد کوچکتر از یک، به عددی بسیار کوچک منتهی می‌شود. در این حالت، نمایش عددی کامپیوتر نمی‌تواند چنین مقادیری را به‌درستی نشان دهد و آن را به صفر گرد می‌کند. این پدیده را underflow می‌نامند.

برای جلوگیری از این مشکل، باید راهی برای محاسبه برچسب‌های پیش‌بینی‌شده مدل بیزساده‌لوحانه پیدا کنید که در آن نیازی به محاسبه مستقیم اعدادی مانند $P(x \mid y)$ نباشد.

**راهنمایی:** از لگاریتم استفاده کنید.

در این قسمت، باید توابع ذکر شده را کامل کنیم تا بتوانیم مدل بیز ساده‌لوحانه خود را به دست آوریم. اما پیش از انجام هر کاری باید به مسئله underflow موجود رسیدگی کنیم. در بیز ساده‌لوحانه ما احتمالات را در یکدیگر ضرب می‌کنیم. از آنجایی که مقدار احتمالات بین صفر و یک است، متناسب با صورت سوال، مقدار احتمال نهایی بسیار کوچک می‌شود و در کامپیوتر صفر در نظر گرفته می‌شود. برای جلوگیری از رخ دادن این مشکل، از لگاریتم احتمال‌ها استفاده می‌کنیم. بنابراین از فرمول پایه‌ای زیر برای محاسبه احتمالات استفاده می‌کنیم: $\log P(y \mid x) \propto \log P(y) + \sum x\_j \times log\, P(w\_j \mid y)$.

حال به سراغ تابع fit_naive_bayes_model(matrix, labels) می‌رویم. ابتدا از ماتریس ورودی، تعداد پیام‌ها و کلمات پرتکرار را مشخص کرده و پیام‌های اسپم و غیر اسپم را جدا می‌کنیم.

```
n_messages, n_words = matrix.shape

# تقسیم داده ها بر اساس برچسب کلاس
spam_indices = labels == 1
ham_indices = labels == 0
```

در ادامه احتمالات هر کلاس را حساب می‌کنیم و برای جلوگیری از پدیده underflow از آن‌ها لگاریتم می‌گیریم.

```
# تعداد پیام های هر کلاس
n_spam = np.sum(spam_indices)
n_ham = np.sum(ham_indices)

# احتمال اولیه کلاس‌ها (با log)
log_class_priors = np.log(np.array([
    n_ham / len(labels),
    n_spam / len(labels)
]))
```

سپس تعداد دفعاتی که هر کلمه در پیام‌های اسپم/غیر اسپم آمده را حساب می‌کنیم و برای جلوگیری از صفر شدن احتمالات، با استفاده از هموارسازی لاپلاس، یک واحد به آنها اضافه می‌کنیم.

```
# مجموع کلمات در هر کلاس + لاپلاس
spam_word_counts = np.sum(matrix[spam_indices], axis=0) + 1
ham_word_counts = np.sum(matrix[ham_indices], axis=0) + 1
```

سپس مجموع کلمات هر کلاس را پیدا کرده و لگاریتم احتمالات شرطی هر یک را حساب می‌کنیم. در نهایت احتمالات به دست آمده را در یک ماتریس ذخیره می‌کنیم.

```
# مجموع کل کلمات + |V| (برای لاپلاس)
spam_total = np.sum(spam_word_counts)
ham_total = np.sum(ham_word_counts)

# احتمال شرطی کلمات برای هر کلاس (با log)
log_word_probs = np.vstack([
    np.log(ham_word_counts / ham_total),  # 0 کلاس = ham
    np.log(spam_word_counts / spam_total)  # 1 کلاس = spam
])
```

در نهایت تابع ما به عنوان خروجی احتمالات شرطی و پسین را برمی‌گرداند.

در ادامه به سراغ تابع predict_from_naive_bayes_model(model, matrix) می‌رویم. هدف اصلی ما در این تابع این است که با استفاده از مدل آموزش دیده برای داده‌های جدید بتوانیم برچسب، یعنی مقدار ۰ و ۱ یا همان اسپم بودن و نبودن، را تخصیص دهیم. در ابتدا احتمالات موجود در ماتریس ورودی را ذخیره می‌کنیم و پس از آن مقدار log_liklihood را برای کلاس‌هایمان حساب می‌کنیم.

```
log_class_priors = model["log_class_priors"]
log_word_probs = model["log_word_probs"]  # شکل: [2 × V]

# ضرب لگاریتم احتمال‌ها در بردار تعداد کلمات
log_likelihood_ham = matrix @ log_word_probs[0] + log_class_priors[0]
log_likelihood_spam = matrix @ log_word_probs[1] + log_class_priors[1]
```

در نهایت با مقایسه احتمالات، برچسب را برای داده‌ها تعیین می‌کنیم.

```
return (log_likelihood_spam > log_likelihood_ham).astype(int)
```

(ج) برخی کلمات (توکن‌ها) ممکن است نشانه‌ی قوی‌تری باشند که یک پیام واقعاً اسپم است. به عبارت دیگر، بعضی کلمات نسبت به پیام‌های معمولی، بیشتر در پیام‌های اسپم ظاهر می‌شوند. برای اینکه بفهمیم یک توکن تا چه حد نشانه‌ی اسپم بودن پیام است، می‌توانیم از این فرمول استفاده کنیم:

$$\log\left(\frac{P(x_j = i \mid y = ۱)}{P(x_j = i \mid y = ۰)}\right) \qquad (۲)$$

که $y = ۱$ نشان‌دهندهٔ اسپم بودن آن پیام و $y = ۰$ نشان‌دهندهٔ معمولی بودن آن پیام می‌باشد. اگر مقدار این نسبت زیاد باشد، یعنی آن توکن بیشتر در اسپم‌ها دیده شده و می‌تواند نشانه‌ی خوبی برای شناسایی اسپم باشد. تابع `get_top_five_naive_bayes_words` را در فایل کد خود کامل کنید. این تابع باید با استفاده از فرمول بالا، پنج توکن (کلمه) را پیدا کند که بیشترین نسبت را دارند و بنابراین قوی‌ترین نشانه برای اسپم بودن پیام هستند.

در این بخش می‌خواهیم ۵ کلمه‌ای که بیشترین نشانه برای اسپم بودن یک پیام هستند را با استفاده از فرمول ذکر شده، شناسایی کنیم. بنابراین به سراغ تابع get_top_five_naive_bayes_words(model, dictionary) می‌رویم. در ابتدا احتمالات لگاریتمی موجود در مدل را ذخیره می‌کنیم. در ادامه باید نسبت بین احتمالات را حساب کنیم. به دلیل اینکه با لگاریتم احتمالات کار می‌کنیم تنها کافی است تا تفاضل لگاریتم احتمالات را به دست بیاوریم.

```
log_word_probs = model["log_word_probs"]  # شکل: [2 × V]

# محاسبه تفاوت لگاریتمی: log(P(w | spam)) - log(P(w | ham))
log_ratios = log_word_probs[1] - log_word_probs[0]
```

در ادامه، دیکشنری را به حالت ایندکس: کلمه تبدیل می‌کنیم. یعنی مقدار تکرار کلمات را به عنوان کلید در نظر می‌گیریم تا بتوانیم ۵ تای اول آنها را فیلتر کنیم. سپس لیستی از کلمات و مقدار تکرار یا همان log_ratio می‌سازیم.

```
index_to_word = {index: word for word, index in dictionary.items()}
word_scores = [(index_to_word[i], log_ratios[i]) for i in range(len(log_ratios))]
```

در قدم آخر، لیست بالا را را به ترتیب نزولی مرتب می‌کنیم و ۵ تای نخست را برمی‌گردانیم.

```
top_5 = sorted(word_scores, key=lambda x: x[1], reverse=True)[:5]

return [word for word, score in top_5]
```

در نهایت با ران کردن تابع main خروجی به شرح زیر به دست می‌آید:

```
Naive Bayes had an accuracy of 0.978494623655914 on the testing set
The top 5 indicative words for Naive Bayes are:  ['claim', 'won', 'prize', 'tone', 'urgent!']
```

## بخش ۳: KNN

دانلود فایل این بخش

۱. (۱۲ نمره) درستی و نادرستی عبارات زیر را با ذکر دلیل مشخص کنید.

(آ) برخلاف الگوریتم k-means، الگوریتم KNN فاز آموزشی ندارد.

(ب) با استفاده از تکنیک Elbow، می‌توان مقدار بهینهٔ $k$ را در الگوریتم KNN پیدا کرد.

(ج) الگوریتم KNN یک روش به منظور دسته‌بندی می‌باشد و به منظور وظایفی مانند پیش‌بینی قیمت‌خانه که متغیری پیوسته‌است، استفاده از آن بی‌معنا است.

**آ)** عبارت **درست** است.

- الگوریتم KNN (k-nearest neighbors) یک الگوریتم بدون فاز آموزشی (lazy learning) است.
  - در این الگوریتم، داده‌ها فقط ذخیره می‌شوند و هیچ مدل یا پارامتر خاصی در مرحله آموزش ساخته نمی‌شود.
  - هنگام پیش‌بینی (یا دسته‌بندی)، الگوریتم فاصله‌ی داده‌ی جدید را با همه‌ی داده‌های آموزشی اندازه می‌گیرد و بر اساس نزدیک‌ترین همسایه‌ها تصمیم‌گیری می‌کند.
- در مقابل، الگوریتم k-means برای خوشه‌بندی داده‌ها به کار می‌رود و دارای فاز آموزشی (training phase) است.
  - این الگوریتم ابتدا k مرکز خوشه را انتخاب کرده و سپس به‌صورت تکراری مراکز خوشه‌ها و تخصیص داده‌ها را به‌روزرسانی می‌کند تا به همگرایی برسد.
- در نتیجه KNN بدون فاز آموزشی، فقط در زمان پیش‌بینی محاسبه انجام می‌دهد اما K-means دارای فاز آموزشی برای بهبود حدس مرکز خوشه‌ها می‌باشد.

**ب)** عبارت **نادرست** است.

۱. تکنیک Elbow (آرنج) برای پیدا کردن مقدار بهینهٔ k در الگوریتم k-means به کار می‌رود، نه در KNN.

۲. در KNN، k تعداد همسایه‌هاست و با cross-validation انتخاب می‌شود، نه با Elbow.

چرا؟!

- در k-means، معیار سنجش کیفیت خوشه‌بندی، مجموع فاصله‌ها (WSS) است Elbow .کاهش WSS را بررسی می‌کند تا نقطه‌ای که کاهش آن کند می‌شود (نقطه‌ی زانو) را پیدا کند.
- اما در KNN، هدف پیش‌بینی درست است (مثلاً دسته‌بندی). معیار مهم دقت (accuracy) است، نه فاصله‌ها.
- اگر بخواهیم برای KNN هم نمودار بکشیم، باید محور y را خطای پیش‌بینی یا accuracy بگذاریم و آن را برای مقادیر مختلف k بررسی کنیم. این اسمش Elbow نیست، بلکه یک روش ساده‌ی اعتبارسنجی است.

**ج)** عبارت **نادرست** است.

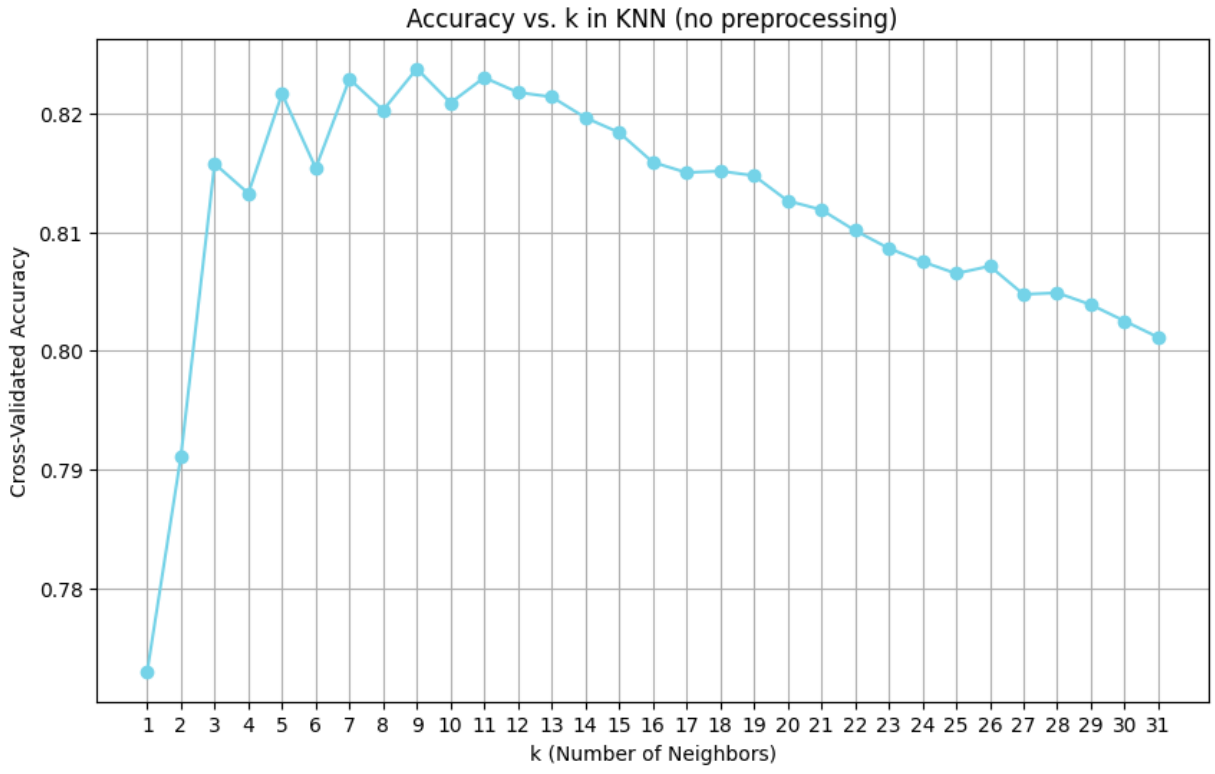
- در KNN دسته‌بندی (Classification)، برچسب کلاس اکثریت بین k همسایه تعیین می‌شود.
- در KNN رگرسیون (Regression)، مقدار خروجی برای نمونه جدید، میانگین (یا میانگین وزنی) خروجی‌های k همسایه‌ی نزدیک است.
- الگوریتم KNN فقط مخصوص دسته‌بندی نیست، بلکه می‌تواند برای رگرسیون هم استفاده شود بنابراین، KNN برای پیش‌بینی قیمت خانه (که یک متغیر پیوسته است) قابل استفاده و کاملاً معنادار است.

۲. (۱۵ نمره) کد اولیهٔ موجود در فایل `KNN.py` را در نظر بگیرید. هایپرپارامتر `random_state` را با چهار رقم آخر یکی از اعضای گروه جایگزین کرده و به سوالات زیر پاسخ دهید.

در این بخش از ۴ رقم آخر شمارهٔ دانشجویی صبا استفاده می‌کنم (۴۲۷۶) چراکه ۴ رقم شمارهٔ دانشجویی خودم با ۰ شروع می‌شه.

(آ) در ابتدا بدون هیچ پیش‌پردازشی، تعداد بهینهٔ $k$ را با استفاده از cross-validation از $k = ۱$ تا $k = ۳۱$ را بدست آورید و نمودار دقت را همراه با $k$های متفاوت رسم نمایید.

ابتدا داده‌های دسته‌بندی را با make_classification تولید می‌کنیم. سپس با استفاده از KNeighborsClassifier و cross_val_score، دقت مدل را برای مقادیر k از ۱ تا ۳۱ با 5-fold cross-validation محاسبه می‌کنیم. در نهایت، دقت‌ها را روی نمودار رسم می‌کنیم تا بهترین مقدار k را بر اساس بیشترین دقت گزارش شود. توضیحات نمودار در صفحهٔ بعد آورده شده است.



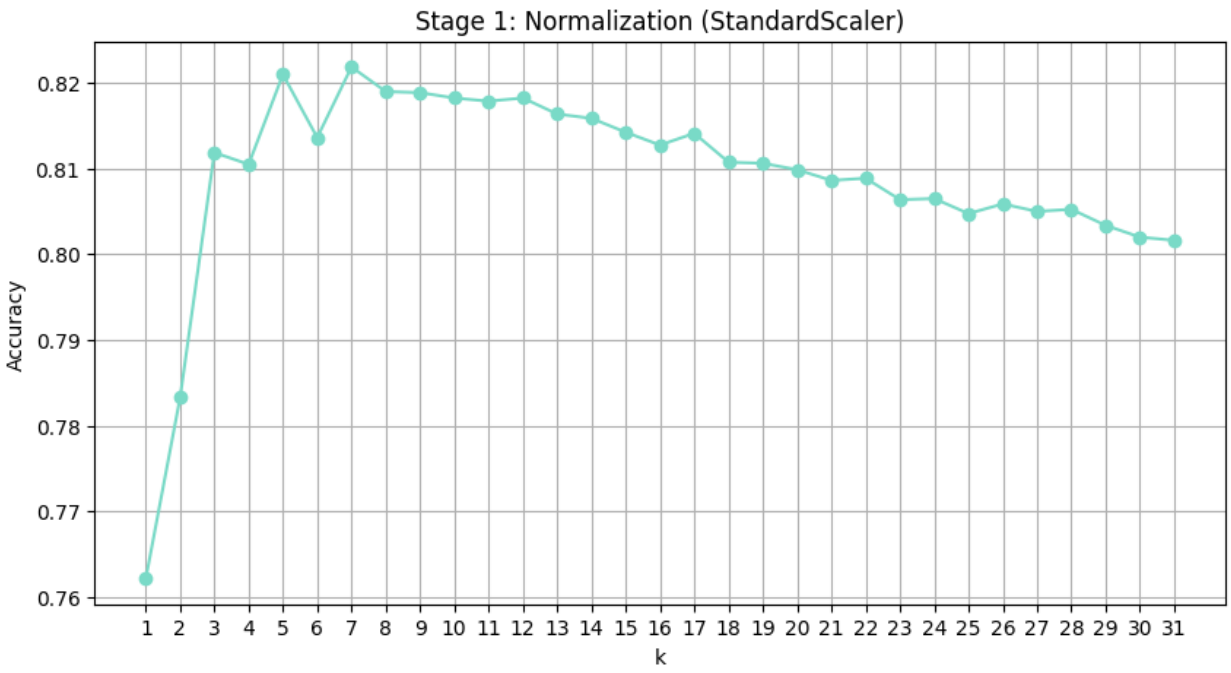
**دقت بر حسب kهای متفاوت بدون اعمال آماده سازی بر روی داده‌ها**

روند کلی دقت کم و بیش تا k=۹ در حال افزایش بوده است و در ۹ به ماکسیمم دقت می‌رسد و بعد از آن آرام آرام از دقت کاسته می‌شود. اگرچه بعد از پردازش ممکن است این مقدار عوض شود.

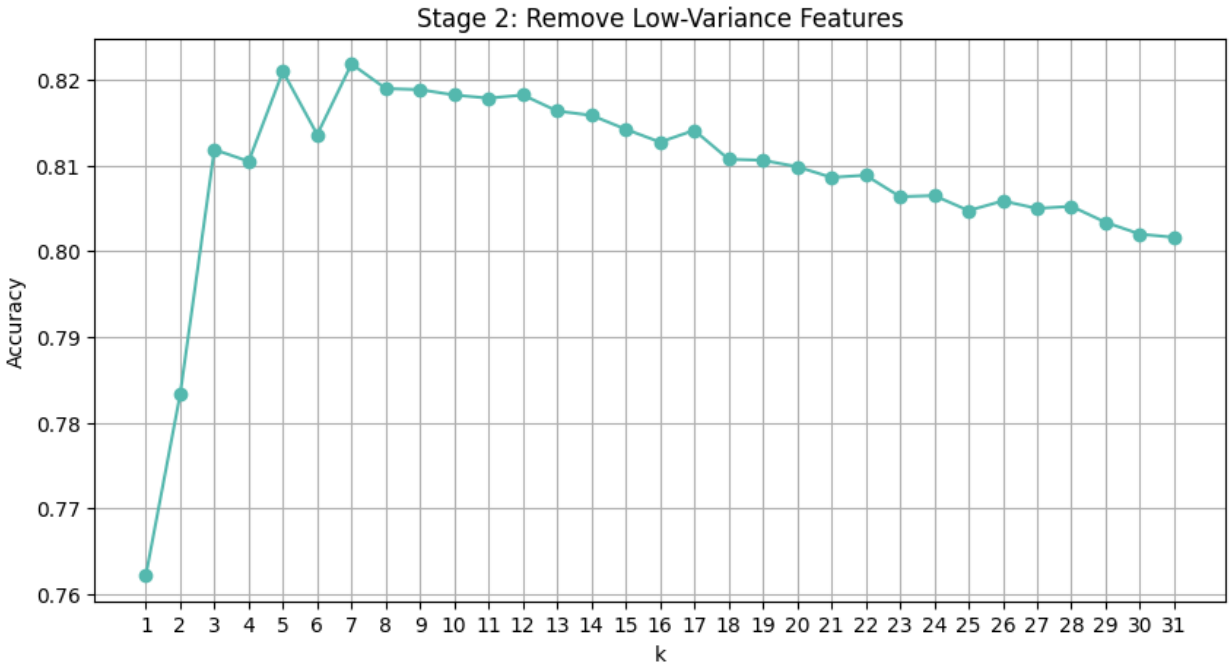
در k=۱ دقت نسبتا پایینی را شاهد هستیم (حدود ۰.۷۷۵)، که نشان‌دهنده‌ی over fitting است؛ چون فقط به نزدیک‌ترین همسایه تکیه دارد و نسبت به نویز حساس است. به همین دلیل هم از cross validation برای رفع این مشکل استفاده می‌کنیم و می‌بینیم که تا حد خوبی هم می‌تواند عملکرد مدل را بهتر کند و به دقتی حدود ۰.۸۲۵ برسد.

(ب) در هر مرحله و حداقل ۴ مرحله، سعی بر رفع یکی از چالش‌های داده بپردازید (رفع هر یک از چالش‌ها را یک مرحله در نظر بگیرید) و ذکر کنید چرا موردی که بررسی می‌کنید، برای الگوریتم KNN می‌تواند چالش باشد. مقدار $k$ بهینه، دقت آن و نمودار دقت به همراه $k$های متفاوت را نیز در هر قدم رسم کنید.
**نکته:** ممکن است پس از پیش‌پردازش در قدم‌ها، دقت مدل کاهش بیابد. سعی کنید علت این اتفاق را توجیه کنید و ادامه دهید.

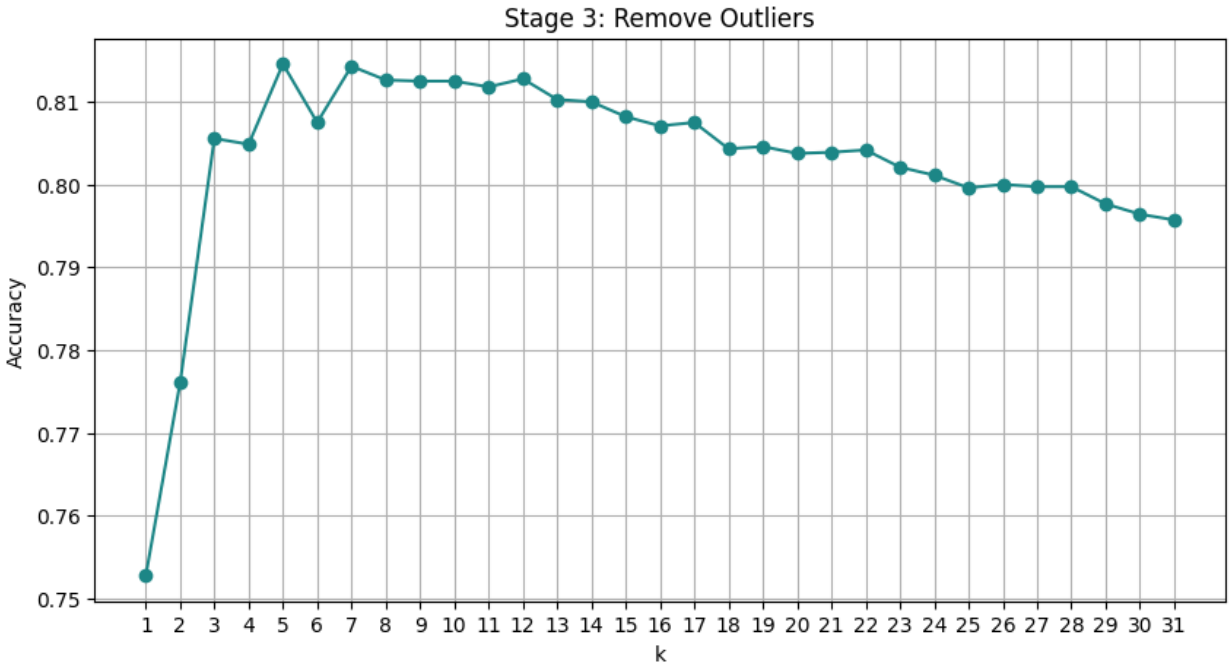
**قدم اول: استانداردسازی**



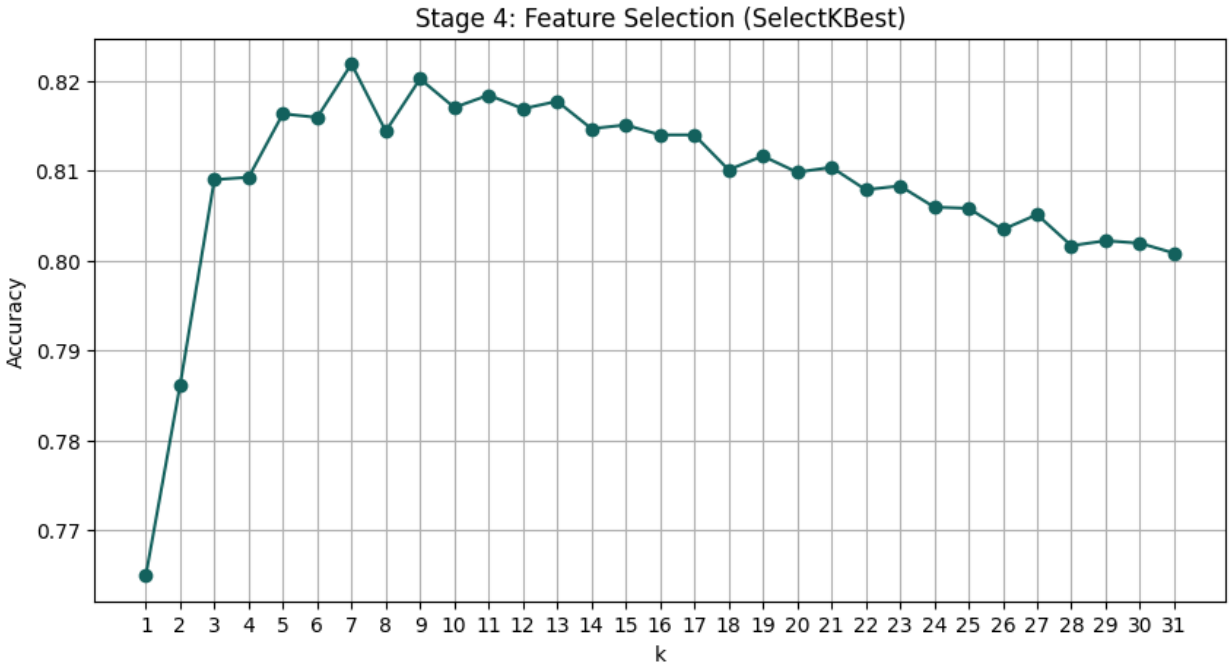
**قدم دوم: حذف ویژگی‌های کم‌اهمیت با Variance Threshold**




**قدم سوم: حذف داده‌های پرت**



**قدم چهارم: Feature Selection**




می‌بینیم که در هر گام دقت مدل در حدود ۰.۸۲ باقی می‌ماند و در گام ۳ کمی افت می‌کند. **اما چرا ممکن پس از پیش‌پردازش، دقت مدل کاهش پیدا کند؟**

- **حذف اطلاعات مفید:(noise vs. signal)**

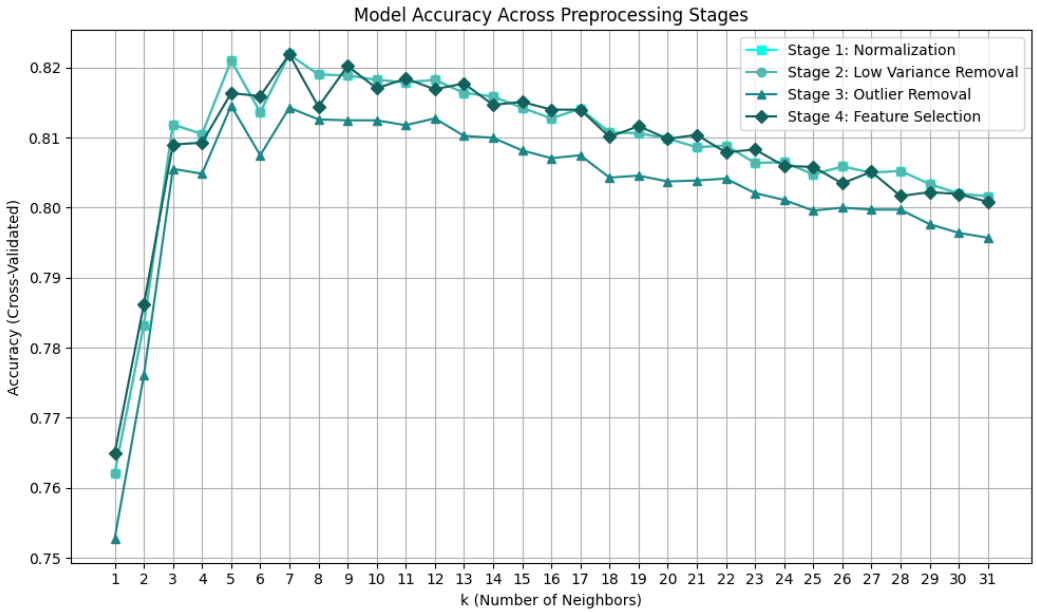
ممکن است در مرحله‌ی حذف داده‌های پرت، ممکنه بعضی داده‌هایی که در ظاهر پرت بوده‌اند اما حاوی الگوی واقعی بودند، اشتباهی حذف شده باشند. این باعث کاهش قدرت مدل در تشخیص درست خواهد شد.

- **کاهش اندازه‌ی داده‌ها:**

عملکرد الگوریتمی مانند KNN بسیار به چگالی و تعداد داده‌ها وابسته است. در مرحله‌ی حذف داده‌های پرت، تعداد نمونه‌ها کم می‌شود و این مسئله به روی تخمین در cross-validation تأثیر می‌گذارد.

- **حساسیت KNN به ابعاد:**

مثلاً وقتی فیچر سلکشن انجام می‌دهیم، ممکن است فیچرهایی حذف شوند که برای بعضی کلاس‌ها مهم بوده باشند. در نتیجه ممکن است موجب آن شود که دقت کمی پایین‌تر بیاید یا بهینه‌ی k تغییر کند.




نکات نمودارمقایسه‌ای:

- این نمودار نیز یک مقایسه جامع و روند کلی است که نشان می‌دهد علرغم تغییرات جزئی، تغییرات و روند دقت در مجموع تقریبا ثابت می‌ماند و در **k = ۷ با دقت ۰.۸۲۲** ماکسیمم باقی می‌ماند.
- **حذف داده‌ها با واریانس کم** هیچ اثر محسوسی به روی روند ندارد چرا که در داده‌های جنریت شده داده‌ای تولید نشده که در این الگوریتم حذف شود.
- **فیچر سلکشن** به خوبی کاستی جزئی ناشی از حذف داده‌های پرت را جبران می‌کند.

## بخش ۴: Random Forest

دانلود داده این بخش

در این بخش با استفاده از داده‌های مربوط به مشتریان یک شرکت مخابراتی، به بررسی مدل Forest Random برای پیش‌بینی قطع سرویس (Churn) می‌پردازیم.

۱. (۵ نمره) **ساخت مدل پایه**

یک مدل `Random Forest` با `n_estimators=100` و `max_features='sqrt'` بسازید. از مقدار پیش‌فرض `max_depth` استفاده کنید.

دقت (accuracy) مدل را روی داده آموزش و تست گزارش کرده و ساختار کلی مدل را بررسی کنید. آیا تفاوتی بین دقت آموزش و تست وجود دارد؟ نتیجه را تحلیل کنید.

ابتدا داده‌ها را تمیز و آمادهٔ پردازش می‌کنیم. سپس الگوریتم ساخته شده رو ران می‌کنیم تا دقت برای داده‌های تست و آموزش را گزارش کند.

```
train acc:  0.9977
test acc: 0.7771
number of trees: 100
average trees depth: 23.2
```

بنظر می‌رسد علرغم آنکه الگوریتم روی داده‌های آموزش عملکرد خوبی دارد به روی داده‌های تست عملکرد نه چندان مطلوبی دارد که حکایت از احتمالا overfitting دارد.

دلایل احتمالی بیش‌برازش:

- تعداد زیاد درخت‌ها بدون محدودیت عمق (max_depth مشخص نشده.)
- داده‌های پرت یا نامتوازن (مثلاً اگر کلاس 'No' خیلی بیشتر از 'Yes' باشه در ستون Churn)
- عدم انتخاب ویژگی مناسب
- عدم استفاده از الگوریتم‌های کاهش پیچیدگی

برای آنکه بفهمیم که مشکل واقعا همین است این بار محدودیت به روی ماکسیمم عمق قرار می‌دهیم.

```
train acc:  0.8459
test acc: 0.8048
```

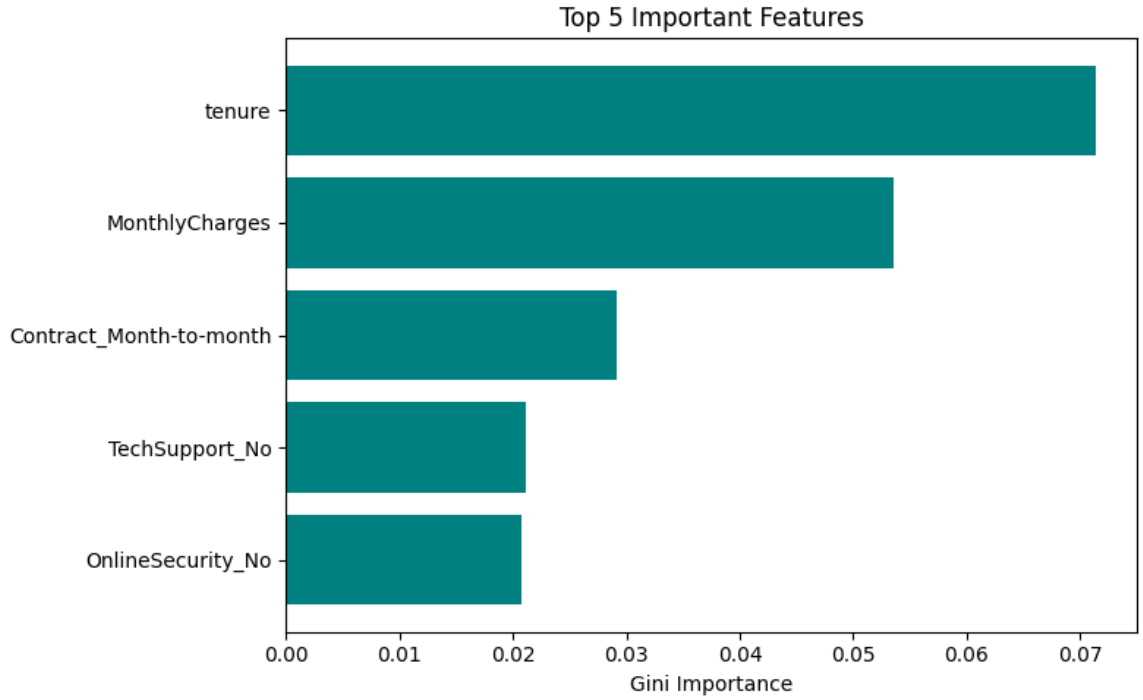
می‌بینیم که بله! اگر عمق را کمتر کنیم (مثلا ۹ یا ۸) به نتایجی بسیار بهتر دست خواهیم یافت و بله! دلیل عملکرد نسبتا صعیف مدل بیش‌برازش بوده است.

هرچند پس از چندین بار پیاپی ران کردن کد متوجه شدم که این مسئله تا حد زیادی وابسته به انتخاب برای تست و ترین و تصادفی است چراکه در یکی از دفعات با اورج عمق ۱۲۰، بهترین نتایج در ماکسیمم عمق ۵۵ حاصل می‌شد.

۲. (۷ نمره) **تحلیل اهمیت ویژگی‌ها**

اهمیت ویژگی‌ها در مدل ساخته‌شده را به‌دست آورید و ۵ ویژگی برتر را به‌صورت نموداری نمایش دهید.
آیا این ویژگی‌ها با تحلیل انسانی و شهودی درباره قطع سرویس (Churn) سازگار هستند؟ مثال بیاورید.

برای یافتن مهم‌ترین ویژگی‌ها از الگوریتم جینی استفاده می‌کنیم. این الگوریتم فقط می‌تواند اهمیت معیارها را نشان بدهد و قادر به نشان دادن جهت اهمیت آن‌ها نخواهد بود اما برای تفسیر نیاز داریم جهت آن‌ها را نیز بدانیم.




مهم‌ترین ویژگی‌ها عبارتند از: مدت زمانی که مشتری با شرکت همراه بوده، پرداخت ماهانه، نوع قرارداد، نوع خدمت

اما هنوز مشخص نیست که این عوامل اثر مثبت دارند یا منفی. مشخص نیست با افزایش آن‌ها احتمال رفتن مشتری بیشتر یا کمتر می‌شود. پس برای فهم بهتر این مسئله جهت ۵ پارامتر اول را نیز بررسی می‌کنیم. به این منظور از یک الگوریتم رگرسیون استفاده می‌کنیم.

```
ضریب -0.0505 ← اثر منفی :'tenure' ویژگی
ضریب 0.0176 ← اثر مثبت :'MonthlyCharges' ویژگی
ضریب -0.0253 ← اثر منفی :'Contract_Month-to-month' ویژگی
ضریب 0.0666 ← اثر مثبت :'TechSupport_No' ویژگی
ضریب 0.0662 ← اثر مثبت :'OnlineSecurity_No' ویژگی
```

مدل ادعا می‌کند که افزایش پرداخت ماهانه ریسک رفتن مشتری را افزایش می‌دهد. از طرفی افزایش مدت همراهی مشتری با شرکت یا مدت قرارداد ریسک رفتن مشتری را کاهش می‌دهد. از طرفی مدل می‌گوید عدم استفاده از خدمات جانبی (مثل online security یا tech support) نیز احتمال چرن را افزایش می‌دهد.

❖ **Tenure**

مدت‌زمانی که مشتری با شرکت بوده.

- همان‌طور که انتظار داریم مشتریان با سابقهٔ بیشتر با احتمال کمتری churn می‌کنند که یعنی هرچه مدت سابقه بیشتر شود این احتمال کوچکتر می‌شود.

❖ **MonthlyCharges**

میزان هزینه‌ای که هر ماه از مشتری گرفته می‌شود.

- برای این مورد می‌توان اینطور استدلال کرد که اصولا مشتریانی که اعتماد دارند و وفادارند با احتمال بیشتری مبالغ بالاتری را ممکن است پرداخت کنند اما همچنان داده‌ها نشان می‌دهند که ارتباط میان این دو پارامتر منفی است و با افزایش هزینهٔ ماهیانه افراد بیشتر مایلند churn کنند که احتمالا به همان دلیل است که انتظارات بیشتری دارند یا با احتمال بیشتری ممکن است شرکت‌های رقیب پیشنهادهای جالب‌تری به ارائه دهند.

❖ **Contract_Month-to-month**

نوع قرارداد مشتری

- مشتریان با قراردادهای کوتاه مدت آزادی بیشتری برای خروج دارند، بنابراین نرخ churn در این گروه معمولاً بیشتر است در مقابل، قراردادهای بلندمدت معمولاً با تعهد مالی همراهند و ترک سرویس برای مشتری هزینه‌بر یا دشوارتر است.

❖ **Tech Support یا Online Security**

خدمات جانبی محصول

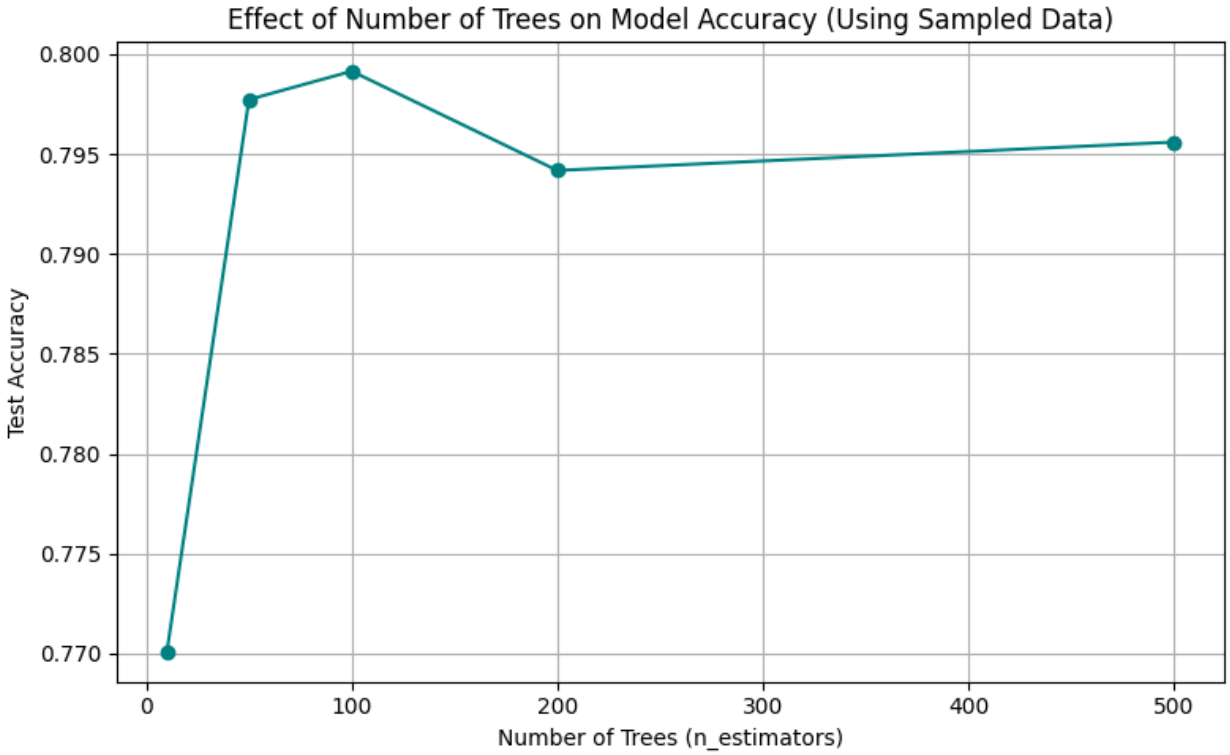
- مدل می‌گوید که مشتریانی که از خدمات امنیت آنلاین یا تک ساپورت استفاده نمی‌کنند، با احتمال بیشتری چرن می‌کنند که تا حدودی قابل انتظار است چراکه معمولا کسانی که اعتمادشان جلب شده است از خدمات و فیچرهای جانبی استفاده می‌کنند و از طرفی دیگر کسانی که از بخشی از خدمات استفاده نمی‌کنند با احتمال بیشتری ممکن است در نهایت ناراضی باشند یا به عبارتی دیگر کسانی که از فیچرهای جانبی استفاده می‌کنند با احتمال بیشتری راضی خواهند بود و با احتمال کمتری چرن خواهند کرد.

۳. (۷ نمره) **اثر تعداد درخت‌ها در عملکرد مدل**

مدل‌هایی با `n_estimators` برابر با ۱۰، ۵۰، ۱۰۰، ۲۰۰ و ۵۰۰ بسازید. برای هر مدل، دقت (accuracy) روی داده تست را محاسبه کرده و روی نمودار رسم کنید.

- محور افقی: تعداد درخت‌ها
- محور عمودی: دقت مدل

آیا افزایش تعداد درخت‌ها بهبود قابل توجهی در عملکرد ایجاد کرده است؟ از چه نقطه‌ای به بعد، افزایش درخت‌ها بی‌اثر می‌شود؟




بطور کلی تغییرات در دقت در بازهٔ وسیعی نوسان نمی‌کند و بین ۰.۷۷ تا ۰.۸ تغییر می‌کند. تا یک جایی دقت با افزایش تعداد افزایش می‌یابد اما از یک جایی به بعد دچار کمی افت و سپس بی تفاوت می‌شود.

ماکسیمم دقت در تعداد ۱۰۰ حاصل می‌شود که دقتی حدود ۰.۷۹۸ را رقم می‌زند. از نقطهٔ ۲۰۰ به بعد افزایش درخت‌ها حاصلی ندارد.

۴. (۷ نمره) **بررسی داده نامتوازن**

برچسب Churn دارای عدم تعادل است. فراوانی هر کلاس (Churn/No Churn) را گزارش کرده و نسبت آن‌ها را محاسبه کنید.

برای مقابله با این مشکل، یک مدل با `class_weight='balanced'` آموزش دهید و با مدل اولیه (سؤال ۱) از نظر متریک‌های زیر مقایسه کنید:

- دقت (Accuracy)
- دقت مثبت (Precision)
- حساسیت (Recall)
- امتیاز $F_1$

کدام مدل برای این داده مناسب‌تر است؟ چرا؟

```
Default Model
-------------
Accuracy:  0.7963
Precision: 0.6667
Recall:    0.4611
F1-score:  0.5452

Balanced Model
--------------
Accuracy:  0.7977
Precision: 0.6517
Recall:    0.5067
F1-score:  0.5701
```

```
Frequency of each class:
 Churn
0    5163
1    1869
Name: count, dtype: int64

Ratio of each class:
 Churn
0    0.734215
1    0.265785
Name: proportion, dtype: float64
```

**کدام مدل مناسب‌تر است و چرا؟**

- مدل **balanced** به‌دلیل اختصاص وزن بیشتر به کلاس اقلیت، معمولاً عملکرد بهتری در شناسایی موارد Churn (برچسب‌های کم‌تعداد) دارد و **مقادیر بالاتری برای Recall و F1-score** ارائه می‌دهد.
- هرچند **دقت کلی (Accuracy)** این مدل تنها اندکی کمتر از مدل اولیه است، ولی **حساسیت (Recall)** آن به‌طور قابل‌توجهی بیشتر است. این موضوع در مسائل پیش‌بینی ترک مشتری اهمیت زیادی دارد، چرا که شناسایی نکردن مشتریان در معرض ترک (False Negatives) می‌تواند هزینه‌های بالایی برای کسب‌وکار به همراه داشته باشد.
- از طرف دیگر، مدل اولیه با وجود Precision بالاتر، Recall پایین‌تری دارد، به این معنا که تعداد زیادی از مشتریان واقعی ترک‌کننده را شناسایی نمی‌کند. بنابراین، در کاربردهایی مانند حفظ و نگهداشت مشتری، که شناسایی حداکثری مشتریان در معرض ریسک اهمیت دارد، استفاده از مدل balanced با وجود افت جزئی در دقت کلی، گزینه‌ی مناسب‌تری است.

۵. (۱۰ نمره) **مقایسه مفهومی با درخت تصمیم**

در بخش ۱ با مدل درخت تصمیم روی داده‌ای در حوزه پزشکی کار کردید. در این بخش نیز با مدل Random Forest در زمینه پیش‌بینی رفتار مشتری کار می‌کنید.

به‌صورت مفهومی، این دو مدل را از نظر معیارهای زیر مقایسه و تحلیل کنید:

- میزان دقت و انعطاف‌پذیری مدل‌ها (مثلاً overfitting یا generalization)
- قابلیت تفسیرپذیری و توضیح‌پذیری مدل
- مناسب بودن هر مدل برای نوع مسئله (پزشکی vs بازاریابی)

در نهایت، نظر خود را درباره این‌که "آیا Random Forest ارزش کاهش تفسیرپذیری را دارد؟" بیان کنید.

مقایسه‌ی مفهومی بین **درخت تصمیم (Decision Tree)** و **جنگل تصادفی (Random Forest)** در سه محور خواسته‌شده به‌صورت زیر است:

**دقت و انعطاف‌پذیری(Overfitting vs Generalization)**

- **درخت تصمیم (Decision Tree)**

مدل‌های درخت تصمیم معمولاً بسیار انعطاف‌پذیر هستند اما تمایل به overfitting دارند، یعنی ممکن است روی داده‌های آموزشی خیلی خوب عمل کنند ولی روی داده‌های جدید عملکرد ضعیف‌تری داشته باشند. این به‌دلیل یادگیری بیش از حد جزئیات و نویز داده‌ها است.

- **رندوم فارست (Random Forest)**

با ترکیب تعداد زیادی درخت تصمیم و استفاده از نمونه‌گیری تصادفی ویژگی‌ها و داده‌ها، Random Forest معمولاً overfitting کمتری نسبت به یک درخت تنها دارد و قابلیت تعمیم (generalization) بهتری روی داده‌های جدید ارائه می‌دهد. بنابراین دقت آن روی داده‌های تست معمولاً بالاتر است.

| معیار | Decision Tree | Random Forest |
|---|---|---|
| **دقت روی آموزش** | بسیار بالا؛ معمولاً دچار **overfitting**می‌شود | نسبتاً بالا ولی نه بیش از حد؛ کمتر overfit می‌کند |
| **دقت روی تست** | معمولاً پایین‌تر از آموزش؛ به دلیلoverfit | بهتر **generalize** می‌کند، دقت تست بیشتر از درخت منفرد است. |
| **انعطاف‌پذیری** | بسیار انعطاف‌پذیر، ولی کنترل‌نشده | انعطاف‌پذیر ولی با کنترل بیشتر به‌واسطه ترکیب چند درخت |

نتیجه: **رندوم فارست** معمولاً دقت بالاتری روی داده‌های تست دارد و generalization بهتری ارائه می‌دهد.

**تفسیرپذیری و توضیح‌پذیری (Interpretability)**

- **درخت تصمیم (Decision Tree)**

بسیار تفسیرپذیر است و می‌توان به راحتی مسیر تصمیم‌گیری را دنبال کرد. این ویژگی مخصوصاً در حوزه‌های حساس مثل پزشکی بسیار مهم است، زیرا متخصصان می‌توانند منطق مدل را بررسی کنند و توضیح دهند.

- **رندوم فارست (Random Forest)**

تفسیرپذیری کمتری دارد، چون ترکیبی از صدها یا هزاران درخت است و مسیر تصمیم‌گیری برای هر نمونه پیچیده و غیرشفاف می‌شود. اگرچه با روش‌هایی مانند اهمیت ویژگی‌ها (feature importance) می‌توان اطلاعاتی کسب کرد، اما قابل مقایسه با یک درخت ساده نیست.

| معیار | Decision Tree | Random Forest |
|---|---|---|
| **تفسیر درخت** | بسیار بالا؛ قابل ترسیم و قابل دنبال‌کردن برای انسان | تفسیرپذیری پایین؛ ترکیبی از صدها درخت است |
| **اهمیت ویژگی‌ها** | قابل استخراج، ولی حساس به تغییرات داده | قابل استخراج به‌صورت آماری؛ ثبات بیشتری دارد |
| **درک استنتاج مدل** | ساده و شفاف | پیچیده و اغلب به عنوان جعبه سیاه (black-box) شناخته می‌شود |

نتیجه: اگر تفسیرپذیری بالا نیاز باشد (مثلاً در کاربردهای حساس مانند پزشکی)، **درخت تصمیم** مناسب‌تر است.

**مناسب بودن برای نوع مسئله (پزشکی vs بازاریابی)**

**در حوزه پزشکی**

به دلیل حساسیت و نیاز به توضیح دلایل تصمیمات مدل (مثل تشخیص بیماری‌ها)، مدل‌های تفسیرپذیر مثل درخت تصمیم ترجیح داده می‌شوند، حتی اگر دقت کمی کمتر داشته باشند. قابلیت اعتماد پزشکان به مدل اهمیت بالایی دارد.

**در حوزه بازاریابی (پیش‌بینی رفتار مشتری)**

هدف معمولاً به حداکثر رساندن دقت پیش‌بینی و شناسایی درست مشتریان است. از آنجا که داده‌ها بزرگ و پیچیده‌تر هستند و اهمیت تفسیر کمتر از دقت است، مدل‌های قدرتمندی مثل Random Forest مناسب‌ترند.

**نظر نهایی: آیا Random Forest ارزش کاهش تفسیرپذیری را دارد؟**

بستگی به هدف مسئله و نیازهای کسب‌وکار دارد. اگر دقت پیش‌بینی و کاهش خطا اولویت اصلی باشد، به خصوص در حوزه‌هایی مانند بازاریابی و مسائل بزرگ داده‌ای، استفاده از Random Forest ارزشمند است. اما اگر توضیح‌پذیری و اعتماد به مدل حیاتی باشد، مثل پزشکی، بهتر است از مدل‌های ساده‌تر و تفسیرپذیرتر مثل درخت تصمیم استفاده کرد.

در بسیاری موارد می‌توان ترکیبی از هر دو را به کار برد؛ مثلاً ابتدا از Random Forest برای پیش‌بینی دقیق استفاده کرد و سپس با روش‌هایی مانند تفسیر ویژگی‌ها (SHAP, LIME) یا مدل‌های ساده‌تر، دلایل تصمیمات را توضیح داد.

۶. (۵ نمره) **تحلیل اثر `max_features`**

مدل‌هایی با `max_features` برابر با `'sqrt'`، `'log2'`، و مقادیر عددی ۳ و ۵ آموزش دهید. دقت روی داده تست را برای هر حالت گزارش کرده و نتیجه را تحلیل کنید.

کدام مقدار `max_features` بهترین تعادل بین عملکرد و پیچیدگی را فراهم کرده است؟

```
max_features = sqrt -> Accuracy = 0.7963, AUC = 0.8418
max_features = log2 -> Accuracy = 0.7956, AUC = 0.8422
max_features = 3 -> Accuracy = 0.8013, AUC = 0.8456
max_features = 5 -> Accuracy = 0.7984, AUC = 0.8419
```

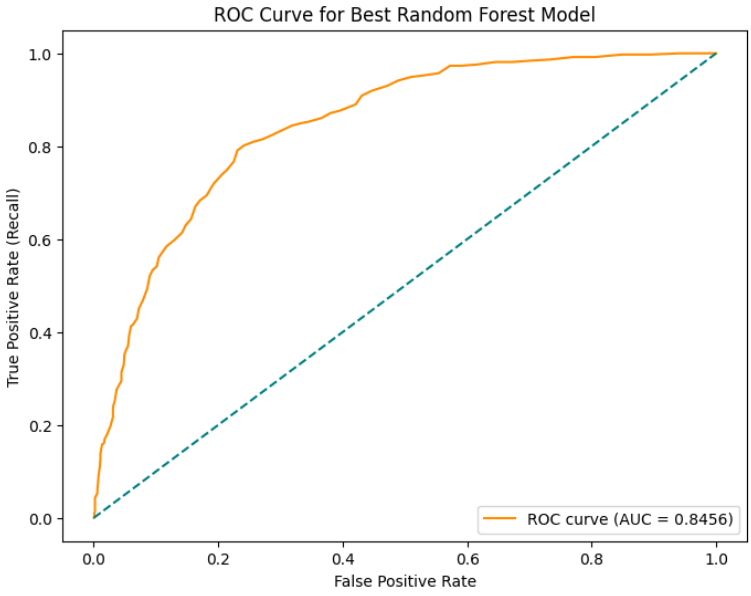
- مقدار max_features = 3 بهترین عملکرد را در بین گزینه‌ها داشته است و بالاترین دقت (۰.۸۴۵۶) را روی داده تست ثبت کرده است.
- هرچند اختلاف دقت بین گزینه‌ها زیاد نیست، اما این مقدار عددی (۳) توانسته تعادلی بهتر میان عملکرد مدل و سطح پیچیدگی ایجاد کند.
- مقادیر sqrt و log2 نیز انتخاب‌های مناسبی هستند، چراکه به‌طور پیش‌فرض در کتابخانه Scikit-learn برای جنگل تصادفی به‌کار می‌روند، اما مقدار دستی 3 در این داده خاص، کمی دقت بالاتری داشته است.

۷. (۷ نمره) **تحلیل متریک‌ها و AUC**

برای مدل نهایی (بهترین مدل تا این مرحله)، منحنی ROC را رسم کرده و مقدار AUC را محاسبه نمایید. سپس تحلیل کنید:

در این مسئله، آیا استفاده از AUC اطلاعات بیشتری نسبت به Accuracy در اختیار ما قرار می‌دهد؟ چرا؟

خب بهترین مدل تا این مرحله چیه؟؟ max_features = 3



**آیا AUC اطلاعات بیشتری از Accuracy می‌دهد؟**

بله. در این مسئله، AUC اطلاعات کامل‌تری نسبت به Accuracy ارائه می‌دهد. چرا؟

۱. **داده نامتوازن است**

در Accuracy فقط درصد کل پیش‌بینی درست را می‌بینی.
اما چون کلاس "No Churn" حدود ۷۳ درصد از کل داده را شامل می‌شود، حتی یک مدل ساده که همیشه "No" پیش‌بینی کند می‌تواند دقت بالایی داشته باشد.
AUC عملکرد مدل را در تمایز دادن بین کلاس‌ها بدون وابستگی به threshold بررسی می‌کند.

۲. **عملکرد کلی مدل را در تمام آستانه‌ها می‌سنجد**

AUC نشان می‌دهد که مدل چقدر خوب می‌تواند نمونه‌های Churn و No Churn را جدا کند در تمام سطوح تصمیم‌گیری ممکن.
اگر AUC به ۰.۸ یا بالاتر نزدیک باشد، مدل به‌خوبی تفکیک می‌کند حتی اگر دقت نهایی متوسط باشد.

۳. **برای تصمیم‌گیری بازاریابی مهم‌تر است**

در بازاریابی، هزینه اشتباه در تشخیص مشتری‌ای که قطع سرویس می‌کند بیشتر از تشخیص اشتباه کسی است که نمی‌خواهد برود.
AUCکمک می‌کند مدل‌هایی را انتخاب کنیم که حساسیت بالاتری به کلاس مثبت (Churn) دارند.

AUC معیار قوی‌تر و جامع‌تری نسبت به Accuracy برای مسائل با داده‌های نامتوازن مثل پیش‌بینی Churn است. حتی اگر Accuracy خوب باشد، ممکن است مدل در شناسایی مشتریانی که قصد قطع سرویس دارند ضعیف عمل کند و اینجاست که AUC واقعاً مفید است.

در نهایت در مسائل با داده‌های نامتوازن مانند پیش‌بینی ریزش مشتری، AUC معیار دقیق‌تر و جامع‌تری نسبت به Accuracy به‌شمار می‌رود. حتی اگر Accuracy نسبتاً بالا باشد، مدل ممکن است در شناسایی صحیح مشتریان در حال ریزش ضعیف عمل کند. در اینجاست که AUC می‌تواند تصویر کامل‌تری از عملکرد واقعی مدل به ما بدهد. مقدار AUC معادل ۰.۸۴۵۶ نشان‌دهنده‌ی توان بالای مدل در تمایز بین مشتریان باقی‌مانده و در حال ریزش است.