Shiraz University

دانشکده مهندسی کامپیوتر

پروژه درس

هوش مصنوعی

نام و نام خانوادگی : مهدی فقهی

بهمن ۱۳۹۹

۱.۰ فاز یک

در قسمت اول مساله از ما خواسته شده بود که دو فایل را دریافت کنیم . در یکی اطلاعات مربوط به ایمیل‌های هرزنامه و در دیگری ایمیل‌های سالم را داشتیم . در اولین قدم به کمک `Pandas library` طلاعات هر دو فایل را به تبدیل به یک `data frame` کردم و این دو تا دیتا فریم را یکی کردم . در قدم بعدی براساس نوع فایلی که از آن آمده بودند یک ستون به دیتا فریم مربوطه اضافه کردم به اسم `real` که مشخص می‌کرد ایمیل موجود سالم هست یا نه .

```
۱  def faz_one():
۲      pol_fake = pd.read_csv("pol-fake.csv")
۳      pol_real = pd.read_csv("pol-real.csv")
۴      pol_real["real"] = 1
۵      pol_fake["real"] = 0
۶
۷      result = pd.concat([pol_real, pol_fake])
۸
۹      shuffled_resutl = result.iloc[np.random.permutation(len(result))].reset_index(drop=True)
```

در قدم بعد اسم تمامی ستون‌های موجود در دیتافریم ها را ذخیره کردم سپس یک لیست را به عنوان لیست که ستون‌هایی هستند که به کمک آن‌ها قرار هست که درخت تصمیم موردنظرمون را بسازیم را جدا کردیم . بعد دیتاهای خویش را به دو دسته دیتا که یکی برای ساخت مدل و دیگری دیتایی که برای بررسی صحت و کارآمدی مدل موجود به کار ما میاید تقسیم میکنیم . در انتها لیستی از دیتافریم اصلی و دیتافریم‌هایی که برای ساخت مدل اصلی و آزمایش مدل ساخته شده استفاده می‌شوند را به عنوان خروجی تابع بازمیگردانیم

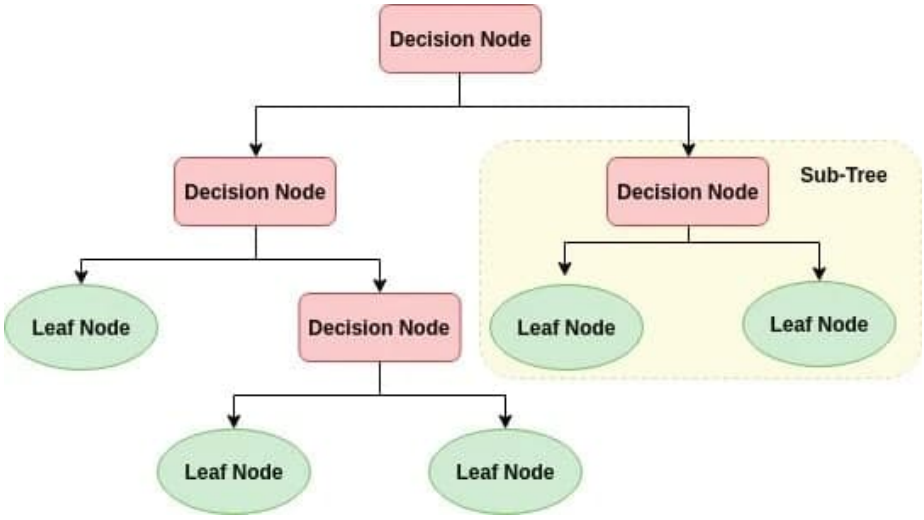
```
۱      lable_name = list(shuffled_resutl.columns)
۲      feature_cols = lable_name[:-1]
۳      variable target and features in dataset split #
۴      x = shuffled_resutl[feature_cols]  Features #
۵      y = shuffled_resutl.real  variable Target #
۶      set test and set training into dataset Split #
۷      X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=2.0,
۸                                                          random_state=1)  test 20% and training 80% #
۹      return [X_train, X_test, y_train, y_test, feature_cols], shuffled_resutl
۱۰
```

.

.

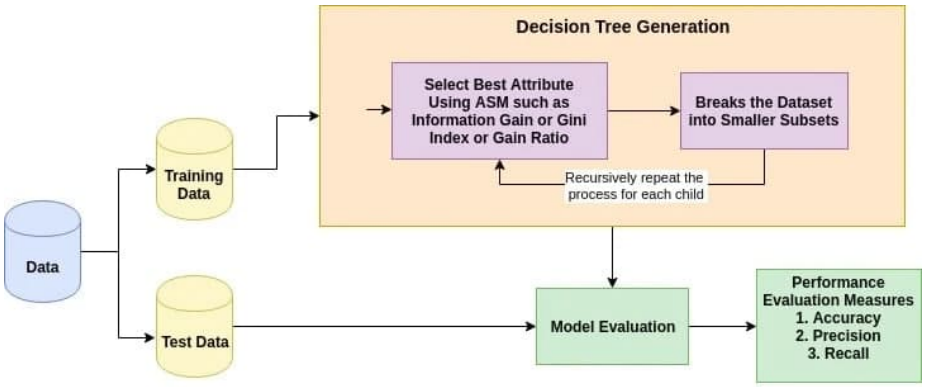
۲.۰ فاز دو

برای حل قسمت دوم پروژه ابتدا لازم هست که با مفهوم `Decision Tree` آشنا بشویم . یکی از پرکاربردترین الگوریتم‌های داده‌کاوی، الگوریتم درخت تصمیم است. در داده‌کاوی، درخت تصمیم یک مدل پیشبینی کننده است به طوری که می‌تواند برای هر دو مدل رگرسیون و طبقه‌ای مورد استفاده قرار گیرد. زمانی که درخت برای کارهای طبقه‌بندی استفاده می‌شود، به عنوان درخت طبقه‌بندی `Classification Tree` هنگامی که برای فعالیت‌های رگرسیونی به کار می‌رود درخت رگرسیون `Classification Tree` نامیده میشود. پیچیدگی زمانی درختان تصمیم تابعی از تعداد سوابق و تعداد صفات در داده های داده شده است. درخت تصمیم روشی بدون توزیع یا غیر پارامتری است که به فرضیات توزیع احتمال بستگی ندارد. درختان تصمیم می توانند داده های با ابعاد بالا را با دقت خوب اداره کنند.



۱.۲.۰ الگوریتم درخت تصمیم چگونه کار می کند؟

- با استفاده از `Attribute Selection Measures (ASM)` بهترین ویژگی را برای تقسیم سوابق انتخاب کنید
- این ویژگی را گره تصمیم گیری کرده و مجموعه داده را به زیرمجموعه های کوچکتر تقسیم می کند.
- درخت سازی را با تکرار این فرآیند به صورت بازگشتی برای هر کودک شروع می کند.



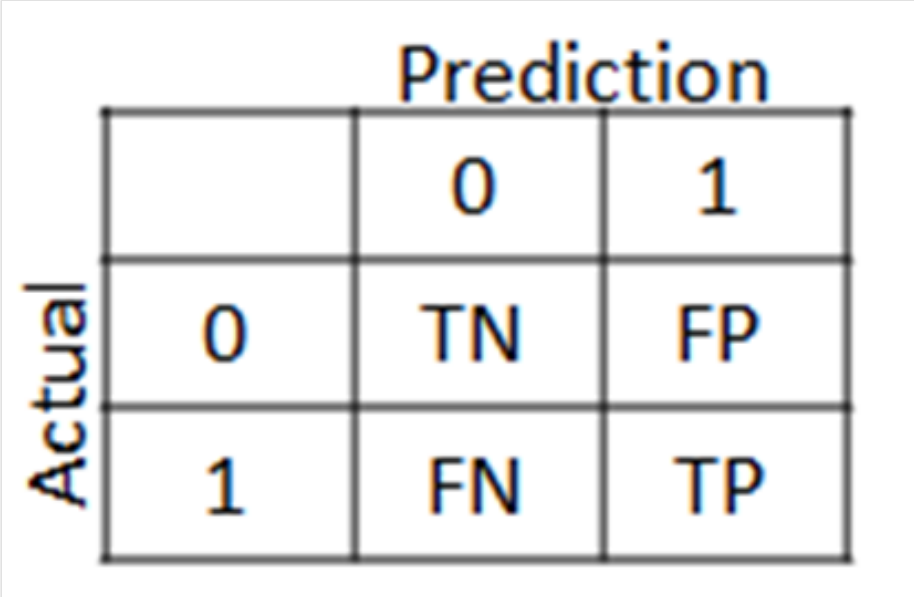
۲.۲.۰ Scikit-learn in Building Classifier Tree Decision

ابتدا مجموعه هایی که در فاز قبلی دسته بندی کرده بودیم را به عنوان ورودی تابع به تابع فاز جدید پاس میدهیم . با استفاده از توابعی که در لایبری فوق موجود هست مدل خودمان را براساس دیتاهای موجود در دیتافریم مربوط به به مدل‌سازی میسازیم و با کمک تابع زیر به ترتیب `accuracy` ، `confusion matrix` ، `precision` ، `recall` ، `F1` را بدست می‌آوریم.

```
def faz_two(input_list):
    X_train, X_test, y_train, y_true, feature_cols = input_list
    Classifer Tree Decision Train #
    clf = DecisionTreeClassifier()
    Classifer Tree Decision Train #
    clf.fit(X_train, y_train)
    dataset test for response the Predict #
    y_pred = clf.predict(X_test)

    accuracy = metrics.accuracy_score(y_true, y_pred)
    print(" "Accuracy:, accuracy)

    print( matrix: "confusion\n", metrics.confusion_matrix(y_true, y_pred))

    print(" "precision:, metrics.precision_score(y_true, y_pred))

    print(" "recall:, metrics.recall_score(y_true, y_pred))

    print(" "F1:, metrics.f1_score(y_true, y_pred))


    return accuracy
```

تعریف ۱ (Accuracy) *دقت را می توان با مقایسه مقادیر مجموعه آزمایش واقعی و مقادیر پیش بینی شده محاسبه کرد.*

تعریف ۲ (confusion-matrix) *جدولی با دو ردیف و دو ستون است که تعداد مثبت کاذب ، منفی کاذب ، مثبت واقعی و منفی واقعی را گزارش می دهد. این اجازه می دهد تا تجزیه و تحلیل دقیق تر از تناسب طبقه ما به ما میدهد .به عنوان مثال ، اگر ۹۵ گربه و فقط ۵ سگ در داده ها وجود داشته باشد ، یک طبقه بندی کننده خاص ممکن است همه مشاهدات را به عنوان گربه طبقه بندی کند. دقت کلی ٪۹۵ است ، اما با جزئیات بیشتر طبقه بندی کننده دارای ٪۱۰۰ میزان تشخیص (حساسیت) برای کلاس گربه اما ٪۰ میزان تشخیص برای کلاس سگ است.*



تعریف ۳ (precision) . *حاصل تقسیم* `tp` *یا همان تعداد مثبت‌های واقعی بر* `tp+fp` *میباشد*

تعریف ۴ (recall) . *حاصل تقسیم* `tp` *یا همان تعداد مثبت‌های واقعی بر* `tp+fn` *میباشد* fn *همان تعداد منفی های کاذب است .*

تعریف ۵ (F۱) .

. `F1 = 2 * (precision * recall) / (precision + recall)`

$F1$ *یک اندازه گیری کلی از دقت مدل است که دقت و یادآوری را با هم ترکیب می کند ، با این روش عجیب و غریب که جمع و ضرب فقط دو ویژگی قبل را مخلوط می کند تا یک ویژگی جدید به وجود بیاید . یعنی نمره خوب* $F1$ *به این معنی است که شما مثبت کاذب کم و منفی کاذب کم دارید ، بنابراین تهدیدهای*

واقعی را به درستی شناسایی می شود. نمره $F1$ درصورت ۱ کامل در نظر گرفته می شود ، در حالی که مدل درصورت ۰ کاملاً شکست می خورد. .

۳.۲.۰ Trees Decision Visualizing

در این قسمت درخت تصمیم خود را به صورت یک گراف چاپ میکنیم . درخت موردنظر یک مدل هست که به کمک این درخت می‌توانیم به صورت دستی نیز با توجه به داده‌هایی که داریم متوجه بشویم که در انتهای درخت یعنی در برگ یک ایمیل هرزنامه هست یا نه . صرف طی کردن درخت با توجه به فیچرهایی که داریم و رسیدن به برگ موردنظر .

```
from sklearn.tree import export_graphviz
from sklearn.externals.six import StringIO
from IPython.display import Image
import pydotplus

dot_data = StringIO()
export_graphviz(clf, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names = feature_cols,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('faze2.png')
Image(graph.create_png())
```

یا به شکل زیر پیاده سازی کنیم

```
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
dot_data = StringIO()
export_graphviz(clf, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True, feature_names = feature_cols,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('teeFaz2.png')
Image(graph.create_png())
```

درنهایت درخت موردنظر را به صورت عکس ذخیره میکنیم .

۳.۰ فاز سه

برای اجرای فاز سوم ابتدا باید `Attribute Selection Measures` آشنا بشویم . انتخاب ویژگی که به کمک آن داده ها را به بهترین شکل تقسیم کنیم ، ابتکاری است. پس `ASM` به عنوان قوانین تقسیم شناخته می شود زیرا به ما کمک می کند تا نقاط شکست tuples را در یک گره مشخص تعیین کنیم. ASM با توضیح مجموعه داده داده شده به هر ویژگی (یا ویژگی) رتبه می دهد.

مشهورترین معیارهای انتخاب عبارتند از: `Gain` ، `Gain Ratio` همچنین `Gini Index`

۱.۳.۰ Measures Selection Attribute

تعریف ۶ (Gain-Information) .

شانون مفهوم آنتروپی را ابداع کرد که ناخالصی مجموعه ورودی را اندازه گیری می کند. در فیزیک و ریاضیات ، آنتروپی به عنوان تصادفی یا ناخالصی در سیستم گفته می شود. در نظریه اطلاعات ، به ناخالصی در گروهی از داده ها اشاره دارد. کسب اطلاعات ، کاهش آنتروپی است. `Information Gain` *بین آنتروپی قبل از تقسیم و میانگین آنتروپی پس از تقسیم مجموعه داده را بر اساس مقادیر مشخصه محاسبه می کند.*

$$\text{Info}(D) = -\sum_{i=1}^{m} pi \log_2 pi$$

$$\text{Info}_A(D) = \sum_{j=1}^{V} \frac{|Dj|}{|D|} \quad \text{X Info}(D_j)$$

$$\text{Gain}(A) = \text{Info}(D) - \text{Info}_A(D)$$

تعریف ۷ (Ratio-Gain) `Information Gain` *برای attrbute که outcomes زیادی دارند به صورت جانبدارنه عمل میکند . به این معنی که صفت ای که values distinct بیشتری دارند را ترجیح میدهد. نسبت سود را می توان به این صورت تعریف کرد:*

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2\left(\frac{|D_j|}{|D|}\right)$$

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo_A(D)}$$

Ratio Gain

تعریف ۸ (Gini-index) `Gini index` *الگوریتم درخت تصمیم دیگر CART (درخت طبقه بندی و رگرسیون) از روش جینی برای ایجاد نقاط تقسیم استفاده می‌کند.*

$$\text{Gini}(D)=1-\sum_{i=1}^{m} P_i^2$$

$$\text{Gini}_A(D)=\frac{|D1|}{|D|}\text{Gini}(D_1)+\frac{|D2|}{|D|}\text{Gini}(D_2)$$

$$\Delta Gini(A) = Gini(D) - Gini_A(D).$$

ویژگی با حداقل شاخص جینی به عنوان ویژگی تقسیم انتخاب می شود.

Implementatio ۲.۳.۰

خوب در این مرحله ما درخت خودمون را به کمک `information Gain` پیاده سازی میکنیم . توجه داشته باشید که در مرحله قبل چون این قسمت از تابع را که در پایین می بینید مقدار دهی نکرده بودم درواقع

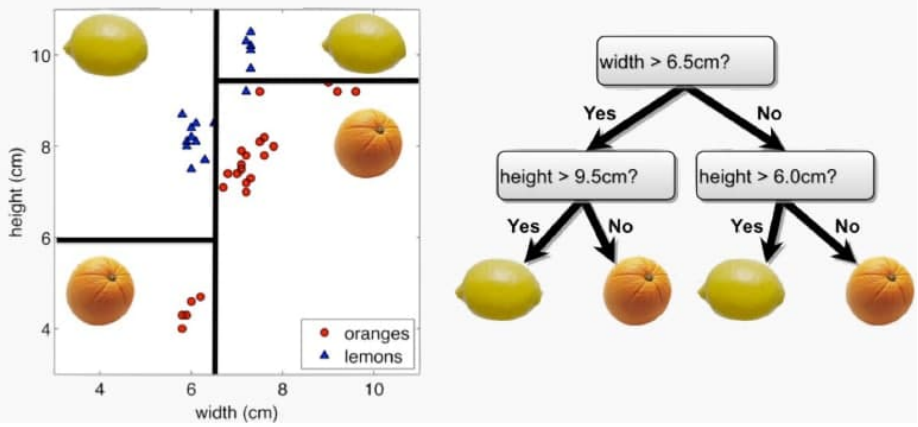
به صورت پیش فرض `Gini index` پیاده سازی شد .

```
clf = DecisionTreeClassifier(criterion="entropy")
```

بقیه موارد هم همانند فاز قبل با `Library sklearn` پیاده سازی شده است .

۴.۰ فاز چهار

برای اینکه این قسمت را بهتر متوجه بشویم ابتدا لازم هست که مفهوم درخت تصمیم رو دوباره بررسی کنیم .



یادگیری کوچکترین درخت تصمیم گیری، برای مجموعه داده های آموزش داده شده کار دشواری است. در هر گره ، ما باید پیش بینی کننده مطلوب را برای تقسیم انتخاب کنیم و مقداری بهینه برای تقسیم انتخاب کنیم. در شکل بالا ، ابتدا با بررسی عرض میوه و استفاده از آستانه ۵.۶ سانتی متر تصمیم می گیریم. ما همچنین می توانستیم با ارتفاع میوه یا انتخاب یک مقدار آستانه متفاوت برای عرض شروع کنیم. انتخاب ما در شکل مناطق درخت تأثیر دارد.

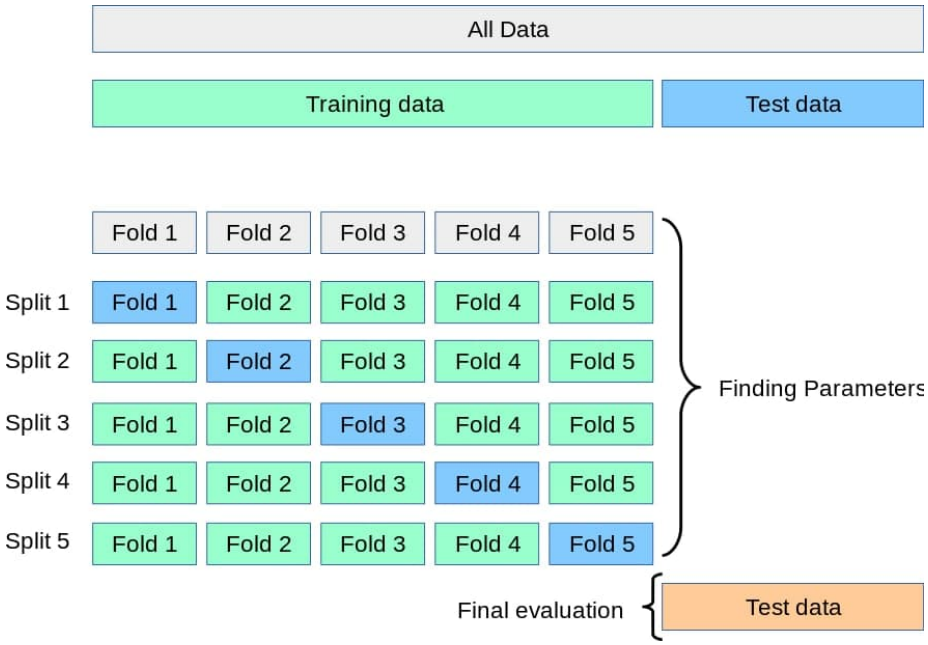
در حین تمرین ، رشد درخت ادامه خواهد یافت تا جایی که هر منطقه دقیقاً یک نقطه تمرین داشته باشد (۱۰۰٪ دقت آموزش). این منجر به یک درخت طبقه بندی کامل می شود که داده های آموزش را تقسیم می کند تا هر مرخصی شامل یک مشاهده باشد. به ترتیب ، درخت کامل بیش از حد با داده های آموزشی سازگار است.

اینجاست که باید با مفهوم `Over fitting` آشنا بشویم.

یعنی اینکه مدل ما به گونه‌ای با داده‌های آموزشی اول ست شود که با هر تغییر کوچکی در دیتای ورودی که قبلا ندیده است پیش بینی جدید تغییرات بسیاری داشته باشد . عملا مدل ما روی یک سری دیتا خوب کار خواهد کرد که قبلا دیده است . و برای دیتاهای جدید ممکن است خیلی بد پیش بینی انجام بدهد. برای حل این مشکل ما از `trade of` استفاده میکنیم .

ما باید یک شرط توقف تعریف کنیم درواقع در این نوع مسئله توقف در سطحی از درخت که هرچقدر که واریانس داده‌ها را کم میکنیم بایاس داده‌ها نیز افزایش زیادی نداشته باشد. درواقع به دنبال یک سازش بین واریانس و بایاس هستیم .

در اینجا از `Cross Validation` کمک میگیریم. عمق مناسب را می توان با ارزیابی درخت از طریق یک مجموعه داده نگهدارنده از طریق `cross validation` تعیین کرد. با بارگیری مجدد داده ها به دفعات ، تقسیم به داده‌های آموزش و داده های تأیید اعتبار ، بررسی درختان با اندازه ارتفاع مختلف و ارزیابی دقیق طبقه بندی به وجود آمده نسبت به داده مورد تأیید ، می توانیم عمق درخت را پیدا کنیم که بهترین مدل را به ما می دهد وعملا سازشی بین واریانس و بایاس ایجاد میکنیم.



در ادامه این مسیر نیاز هست که از `K-fold CrossValidation` استفاده کنیم. K در اینجا مشخص میکند که که داده های خودمون را به چند قسمت تقسیم کنیم و در هر قسمت یکی را به عنوان داده ای

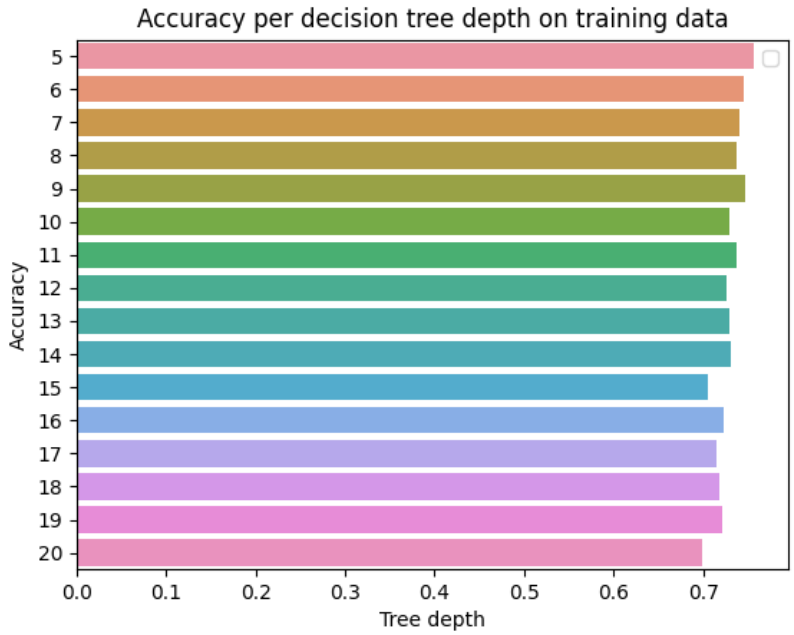
که میخواهیم براساس آن داده‌های موجود را تست کنیم درنظر میگیریم .

در خواست مسئله از ما خواسته است که آن را ۱۰ در نظر بگیریم .

```
def faz_four(input_list):

    kf = KFold(n_splits=10, shuffle=False, random_state=None)
    lable_name = list(input_list.columns)
    feature_cols = lable_name[:-1]
    X = input_list[feature_cols]  Features #
    y = input_list.real  variable Target #

    best_acuracy = 0
    best_depth = 0
    list_of_depth = []
    list_of_acurrcy = []
    for val in range(5, 21):
        score = cross_val_score(DecisionTreeClassifier(max_depth=val, random_state=None), X, y, cv=kf,
                                scoring="accuracy")

        list_of_depth.append(str(val))
        list_of_acurrcy.append(score.mean())
        if score.mean() > best_acuracy:
            best_acuracy = score.mean()
            best_depth = val


    make_ser = pd.Series(list_of_acurrcy, index=list_of_depth)
    sns.barplot(x=make_ser, y=make_ser.index)
    graph your to labels Add #
    plt.xlabel('depth Tree')
    plt.ylabel('Accuracy')
    plt.title(data" training on depth tree decision per "Accuracy)
    plt.legend()
    plt.show()

    return best_acuracy, best_depth
```

براساس عمق ۵ تا ۲۰ `10-fold CrossValidation` را بدست میاریم و در عر عمق متوسط دقت را در نظر میگیریم که اگر بهتر از دقت در عمق قبلی بود همین عمق را به عنوان خروجی در نظر میگیریم و در غیر اینصورت عمقی که درخت را تشکیل میدهیم را به‌روزرسانی میکنیم . درنهایت بهترین عمق و دقتش را به عنوان خروجی بازمیگردانیم.

همچینن در کد قسمتی را برای نشان دادن دقت هر عمق و میزان عمق به کمک `Matplotlib` به شکل یک نمودار نشان میدهیم که شکل زیر یکی از خروجی های آن هست برای داده‌های موجود .



که نشان میدهد عمق پنج عمق بهتری نسبت به بقیه عمق‌ها برای پیدا کردن حاصل و جواب هست .

۵.۰ فاز نهایی

Random Forests Classifiers

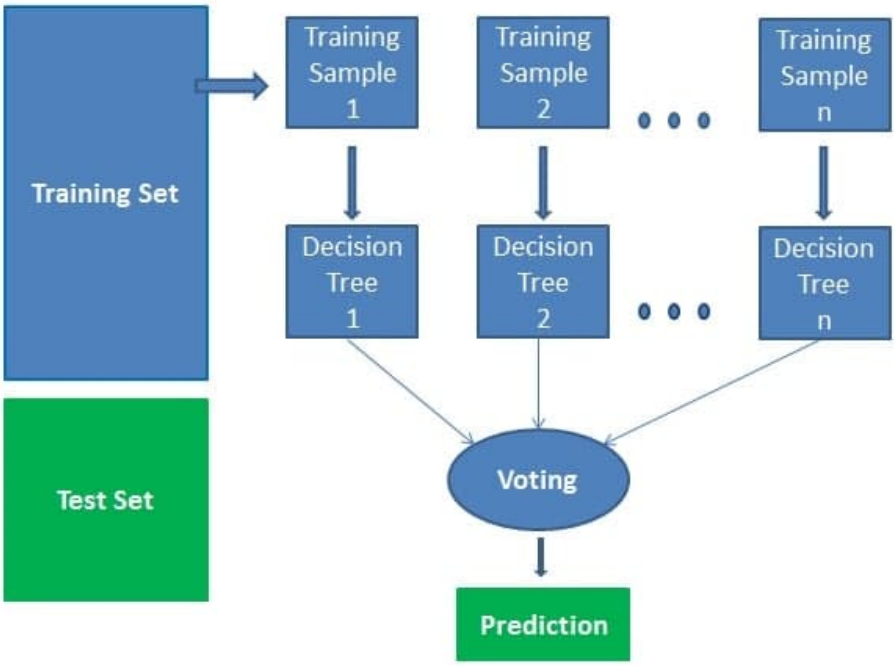
جنگل های تصادفی یک الگوریتم یادگیری `Superviese learning` است. هم برای طبقه بندی و هم برای رگرسیون قابل استفاده است. همچنین انعطاف پذیرترین و آسان ترین کاربرد الگوریتم است. یک جنگل از درختان تشکیل شده است. گفته می شود هرچه تعداد درختان آن بیشتر باشد ، جنگل نیز از استحکام بیشتری برخوردار است. جنگل های تصادفی درختان تصمیم گیری را بر روی نمونه های داده ای که به طور تصادفی انتخاب شده اند ایجاد می کنند ، از هر درخت پیش بینی می کنند و با استفاده از رای گیری بهترین راه حل را انتخاب می کنند. همچنین یک شاخص بسیار خوب از اهمیت ویژگی را فراهم می کند.

بیایید الگوریتم را از نظر عرفی درک کنیم. فرض کنید می خواهید به سفر بروید و دوست دارید به مکانی سفر کنید که از آن لذت بیشتری می‌ببرید. بنابراین برای یافتن مکانی که دوست داشته باشید چه می کنید؟ می توانید به صورت آنلاین جستجو کنید ، نظرات را در وبلاگ ها و پورتال های مسافرتی بخوانید ، یا می

توانید از دوستان خود نیز سوال کنید. فرض کنید شما تصمیم گرفته اید از دوستان خود سوال کنید و با آنها در مورد تجربه سفر گذشته آنها به مکان های مختلف صحبت کنید. از هر دوستی توصیه هایی دریافت خواهید کرد. اکنون شما باید لیستی از آن مکانهای پیشنهادی تهیه کنید. سپس ، از آنها می خواهید از لیست مکانهای پیشنهادی که ایجاد کرده اید رأی دهند (یا یکی از بهترین مکانها را برای سفر انتخاب کنید). مکانی که بیشترین تعداد آرا را داشته باشد انتخاب نهایی شما برای سفر خواهد بود. در روند تصمیم گیری فوق ، دو قسمت وجود دارد. ابتدا ، از دوستان خود در مورد تجربه سفر شخصی خود می‌پرسید و از چندین مکان که بازدید کرده اند ، یک توصیه دریافت می‌کنید. این قسمت مانند استفاده از الگوریتم درخت تصمیم است. در اینجا ، هر دوست انتخاب مکانهایی را که تاکنون بازدید کرده است ، انجام می دهد. قسمت دوم ، پس از جمع آوری تمام توصیه ها ، روش رأی گیری برای انتخاب بهترین مکان در لیست توصیه ها است. کل این فرایند دریافت توصیه ها از دوستان و رای گیری درباره آنها برای یافتن بهترین مکان به عنوان الگوریتم جنگل های تصادفی شناخته می شود. این از نظر فنی یک روش گروهی است (براساس conquer and divide)درختان تصمیم که در یک مجموعه داده تقسیم شده تصادفی تولید می شوند. این مجموعه از طبقه بندی کننده درخت تصمیم به جنگل نیز معروف است. درختان تصمیم گیری فردی با استفاده از یک شاخص انتخاب ویژگی مانند کسب اطلاعات ، نسبت سود و شاخص جینی برای هر ویژگی تولید می شوند. هر درخت به یک نمونه تصادفی مستقل بستگی دارد. در یک مسئله طبقه بندی ، هر درخت رأی می دهد و محبوب ترین کلاس به عنوان نتیجه نهایی انتخاب می شود. در حالت رگرسیون ، میانگین تمام خروجی های درخت به عنوان نتیجه نهایی در نظر گرفته می شود. در مقایسه با الگوریتم های طبقه بندی غیر خطی ساده تر و قدرتمندتر است.

Algorithm

- نمونه های تصادفی را از یک مجموعه داده مشخص انتخاب کنید.
- برای هر نمونه یک درخت تصمیم بسازید و از هر درخت تصمیم یک نتیجه پیش بینی بگیرید.
- برای هر نتیجه پیش بینی شده رأی دهید.
- نتیجه پیش بینی را با بیشترین آرا به عنوان پیش بینی نهایی انتخاب کنید.

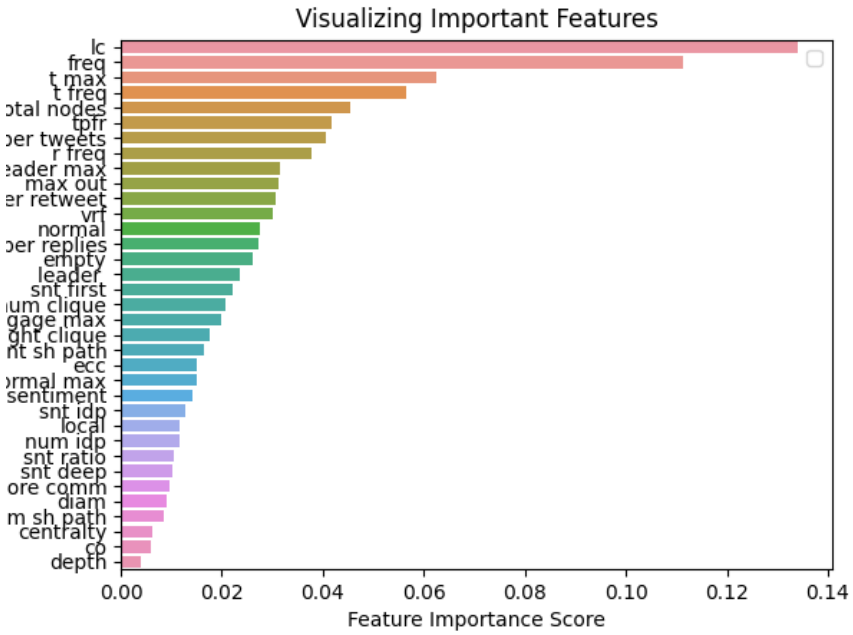


مراحل زیر در شکل زیر توسط کد زیر اجرا شده است .

```
def faz_final(shuffled_resutl):
    lable_name = list(shuffled_resutl.columns)
    feature_cols = lable_name[:-1]

    variable target and features in dataset split #
    x = shuffled_resutl[feature_cols]  Features #
    y = shuffled_resutl.real  variable Target #

    set test and set training into dataset Split #
    X_train, X_test, y_train, y_true = train_test_split(x, y, test_size=2.0,
                                                        random_state=1)  test 20% and training 80% #

    Classifier Gaussian a Create #
    clf = RandomForestClassifier(n_estimators=100)

    y_pred=clf.predict(X_test) sets training the using model the Train #
    clf.fit(X_train, y_train)

    y_pred = clf.predict(X_test)

    accurecy = metrics.accuracy_score(y_true, y_pred)
    ) ,accurecyprint("Accuracy:" #
```

Finding important features

یکی از موارد خوبی که این نوع پیاده‌سازی دارد این هست که به کمک این پیاده‌سازی میتوانیم `feature` که تاثیر بیشتری بر روی نتیجه‌گیری درخت ما دارد را پیدا کنیم. در پیدا سازی پایین من این موارد رو پیدا کردم و به صورت یک نمودار نشان دادم که نمودار را در پایین می‌بینید.

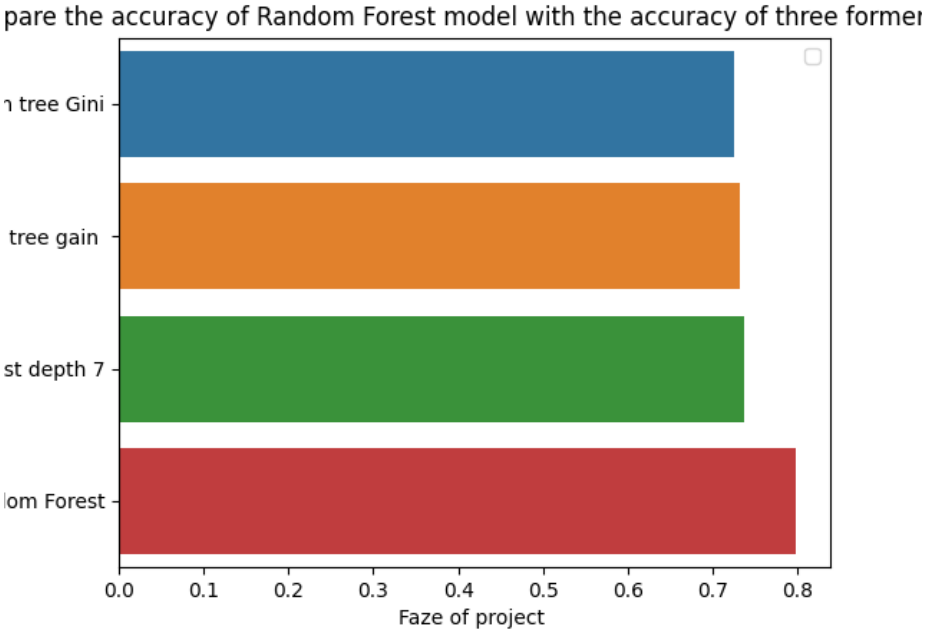


درنهایت می‌توانیم مواردی که کمتر از یک مقدار خاص هستند را حذف کنیم و دقت مدل را دوباره بررسی می‌کنیم اگر بهتر شده بود که مدل جدید با `feature` حذف شده را مورد استفاده قرار میدهیم در غیر این صورت همان مدل قبلی با دقت و accuracy را برمیگردانیم .

```
    feature_imp = pd.Series(clf.feature_importances_, index=feature_cols).sort_values(ascending=False)

    sns.barplot(x=feature_imp, y=feature_imp.index)
    graph your to labels Add #
    plt.xlabel('Score Importance Feature')
    plt.ylabel('Features')
    plt.title(Features" Important "Visualizing)
    plt.legend()
    plt.show()

    new_feature_cols = []
    number = 0

    feature_imp_dict = feature_imp.to_dict()

    for item in feature_imp_dict:

        if feature_imp_dict[item] >= 009.0:
            new_feature_cols.append(item)

    x = shuffled_resutl[new_feature_cols]  Features #

    X_train, X_test, y_train, y_true = train_test_split(x, y, test_size=2.0,
                                                        random_state=1)  test 20% and training 80% #

    clf = RandomForestClassifier(n_estimators=100)
    clf.fit(X_train, y_train)

    y_pred = clf.predict(X_test)

    accurecy2 = metrics.accuracy_score(y_true, y_pred)
    accurecy2) ,Accuracy:" print("new #

    return max(accurecy, accurecy2)
```

۶.۰ نتیجه گیری

درنهایت اینجانب دقت‌های بدست آمده در هر ۳ روش قبل را باهم مقایسه کردم که شکل زیر نشان میدهد که به ترتیب چه روشی بهتر جواب میدهد.



.

۷.۰ مراجع

.

. datacamp.com .

. satishgunjal.com .

medium.com .