مدل‌سازی

در این پروژه سه الگوریتم زیر برای ساخت مدل یادگیری ماشینی در نظر گرفته شده است:

1. Logistic Regression
2. Decision Tree
3. Random Forest

قالب اصلی آموزش و پیش‌بینی مدل‌ها به صورت زیر است:

estim=[Estimator](featuresCol='features', labelCol='label')

pipeline = Pipeline(stages=[estim])

model = pipeline.fit(train)

result = model.transform(test)

در مراحل قبل، تمام ویژگی‌های مورد نیاز را در ستون features به صورت بردار تجمیع کرده‌ایم. همچنین خروجی مورد انتظار پس از تبدیل به باینری در ستون label نگه‌داری می‌شود. برای استفاده از حداکثر امکانات pyspark از pipeline استفاده می‌کنیم تا بعدا در صورتی که نیاز به cross\_validation یا پیش‌پردازش‌های خاصی بود، به صورت pipeline انجام شود. برای آموزش از متد fit و برای پیش‌بینی روی داده‌های آزمون از متد transform استفاده می‌کنیم.

روش رگرسیون منطقی (Logistic Regression)

در این روش که برای طبقه‌بندی باینری به کار می‌رود، به دنبال پیش‌بینی طبقه براساس متغیرهای وابسته آن است. مفروض اصلی آن، خطی بودن متغیرهای مستقل است. مزیت این روش، استفاده ساده آن است و عملکرد بالایی که برای داده‌های تفکیک‌پذیر خطی دارد. برای پیاده‌سازی با pyspark از کد زیر استفاده می‌کنیم:

lr=LogisticRegression(featuresCol='features', labelCol='label')

pipeline = Pipeline(stages=[lr])

model = pipeline.fit(train)

lr\_result = model.transform(test)

روش درخت تصمیم (Decision Tree)

درخت تصمیم نیز مانند رگرسیون منطقی برای داده‌های با تفکیک‌پذیر خطی مناسب است. یکی از مفروضات اصلی درخت تصمیم، در نظر گرفتن ویژگی‌های طبقه‌ای است و در صورتی که همه یا بعضی از ویژگی‌ها پیوسته باشند به طبقه‌ای تبدیل می‌کند. ایده این روش، تولید دستورالعمل‌های شرطی برای تقسیم داده‌ها براساس خلوص است تا به جایی برسیم که خلوص برگ‌ها بیشینه شود؛ یعنی هر برگ این درخت نماینده یک طبقه باشد. کد زیر برای پیاده‌سازی درخت تصمیم به کار رفته است:

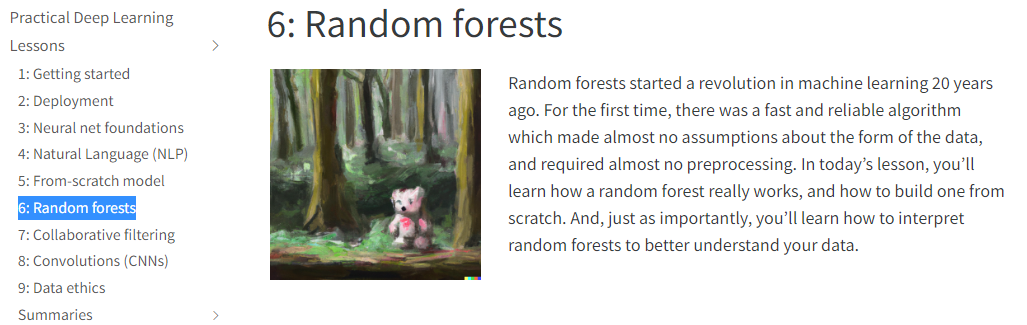
tree = DecisionTreeClassifier()

tree\_pipeline = Pipeline(stages=[tree])

tree\_model = tree\_pipeline.fit(train)

tree\_result = tree\_model.transform(test)

روش جنگل تصادفی(Random Forest)

در این روش، از چندین درخت تصمیم استفاده می‌کنیم و جواب آن‌ها را با هم تجمیع می‌کنیم. در این روش برخلاف روش‌های قبلی هیچ مفروضی بر روی داده نداریم. اهمیت و عملکرد این روش از دیرباز قابل تحسین بوده و دسته‌بندی آن به عنوان روش آماری یا هوشمند محل بحث است؛ به طوری که در دوره آموزش یادگیری عمیق آقای جرمی هاوارد، روش جنگل تصادفی نیز تدریس می‌شود.

کد پیاده‌سازی جنگل تصادفی به صورت زیر است:

from pyspark.ml.classification import RandomForestClassifier

rf = RandomForestClassifier(labelCol="label",

                            featuresCol="features")

rf\_pipeline = Pipeline(stages=[ rf])

rf\_model=rf\_pipeline.fit(train)

rf\_result=rf\_model.transform(test)

روش ماشین بردار پشتیبان (Support Vector Machine)

ارزیابی

اولین چالش در ارزیابی مدل، نامتوازن بودن مجموعه داده است. برای مواجهه به داده‌های نامتوازن روش‌های متداول زیر توصیه می‌شود

1. استفاده از سنجه‌هایی مثل Macro F1, Weighted F1، AUC score
2. نمونه‌گیری متوازن و استفاده از متریک‌های معمولی مثل accuracy
3. وزن دهی متناسب به loss که در هنگام ساخت مدل وزن‌دهی را تعریف کنیم

سنجه Macro F1

در حالت کلی سنجه F1 میانگین هارمونیک دو سنجه دقت(precision) و یادآوری (recall) است. برای رفع ابهام، نام‌های دیگر این سنجه همراه محاسبه آن آورده شده است:

1. سنجه دقت(precision) یا positive predictive value (PPV)
2. سنجه یادآوری(Recall) یا sensitivity یا hit rateیا true positive rate (TPR)

به زبان ساده، سنجه دقت نشان می‌دهد که از بین پیش‌بینی‌های مدل برای کلاس True، چه نسبتی صحیح پیش‌بینی شده است. سنجه یادآوری هم نشان می‌دهد از بین تمام داده‌های کلاس True، چه نسبتی درست پیش‌بینی شده است. با توجه به اینکه مقایسه دو سنجه از یک مدل با دو سنجه از یک مدل دیگر کار دشواری است، یک سنجه F1 که میانگین هارمونیک این دو است معرفی شده است که به صورت زیر است:

تمام این سنجه‌ها در حالت عالی نمایانگر قدرت پیش‌بینی مدل است. اما وقتی داده نامتوازن باشد، باید از حالت خاصی از F1 یعنی macro F1 استفاده کرد. سنجه Macro F1 میانگین غیر وزن‌دار F1 برای تمام کلاس‌ها را اندازه می‌گیرد(N تعداد کلاس‌ها است).

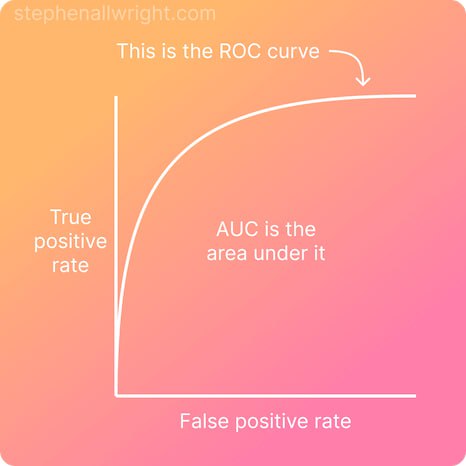
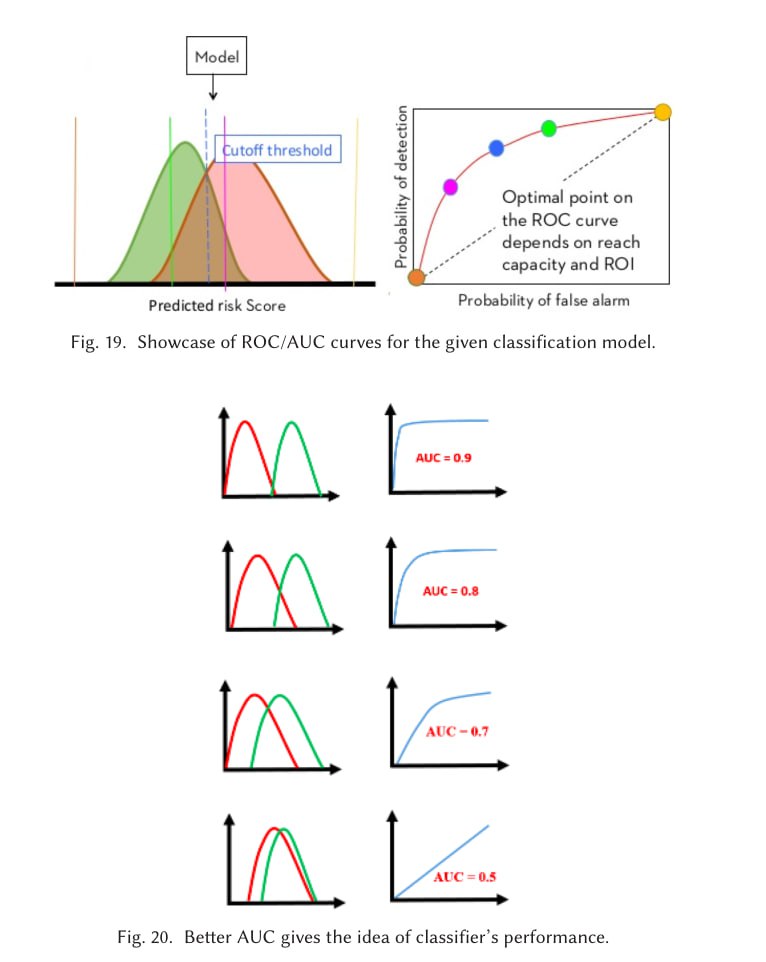
حالت خاصی از این سنجه به نام Weighted F1 وجود دارد که معیار support هر طبقه را نیز به f1 آن طبقه ضرب کرده و سپس میانگین را حساب می‌کند.

سنجه AUC score

قبل از اینکه به امتیاز AUC بپردازیم باید با منحنی ROC آشنا شویم. منحنی ROC تغییرات نرخ مثبت‌های واقعی به نرخ مثبت‌های کاذب را در طول آستانه‌های مختلف نشان می‌دهد.

نمودار ROC براساس تغییرات نرخ‌ها می‌تواند حالت‌های مختلفی داشته باشد. هرچقدر نمودار بالاتر باشد نشان‌دهنده عملکرد بهتر طبقه‌بند است که داده‌های طبقات را بهتر تفکیک کرده است.

برای محاسبه نمودار بهتر از معیار امتیاز AUC یا مساحت زیر منحنی استفاده می‌کنیم تا شهودی که مطرح شد را به صورت معیار کمیتی تبدیل کنیم.



برای ارزیابی دو رویکرد مختلف پیاده‌سازی شده است. در رویکرد اول، ابتدا به صورت دستی سنجه‌ها را از پیش‌بینی‌ها محاسبه می‌کنیم. تابع evaluate\_from\_scratch برای محاسبه دستی سنجه‌های ارزیابی استفاده می‌شود:

def evaluate\_from\_scratch(pred,

                          model\_name='Logistic Regression'):

    pred.groupBy('label', 'prediction').count().show()

    # Calculate the elements of the confusion matrix

    TN = pred.filter('prediction = 0 AND label = prediction').count()

    TP = pred.filter('prediction = 1 AND label = prediction').count()

    FN = pred.filter('prediction = 0 AND label = 1').count()

    FP = pred.filter('prediction = 1 AND label = 0').count()

    # Accuracy measures the proportion of correct predictions

    accuracy = (TN + TP) / (TN + TP + FN + FP)

    recall = (TP) / (TP+FN)

    precision= (TP) / (TP+FP)

    f1=2\*(precision\*recall)/(precision+recall)

    print(f'EVALUATION SUMMARY for {model\_name}:')

    print(f" accuracy:{accuracy}")

    print(f" precision:{precision}")

    print(f" recall:{recall}")

    print(f" f1-score:{f1}")

با استفاده کد بالا، سنجه‌های مورد نیاز را به صورت دستی با استفاده از ماتریس آشفتگی بدست می‌آوریم.

برای رویکرد دوم، از متدهای pyspark استفاده می‌کنیم. تابع evaluate\_from\_spark ، تمام سنجه‌ها را با استفاده از متدهای pyspark محاسبه می‌کند و منحنی ROC را نیز نمایش می‌دهد.

def evaluate\_from\_spark(predictions,

                        model\_name='Logistic Regression'):

    eval = BinaryClassificationEvaluator(rawPredictionCol="probability",

                                         labelCol="label")

    eval2= MulticlassClassificationEvaluator(predictionCol="prediction",

                                             labelCol="label")

    AUC  = eval.evaluate(predictions)

    ACC  = eval2.evaluate(predictions,

                          {eval2.metricName:"accuracy"})

    PREC  = eval2.evaluate(predictions,

                           {eval2.metricName:"weightedPrecision"})

    REC  = eval2.evaluate(predictions,

                          {eval2.metricName:"weightedRecall"})

    F1  = eval2.evaluate(predictions,

                         {eval2.metricName:"f1"})

    WeightedFMeasure=eval2.evaluate(predictions,

                                    {eval2.metricName:"weightedFMeasure"})

    print(f"{model\_name} Performance Measure")

    print(" Accuracy = %0.8f" % ACC)

    print(" Weighted Precision = %0.8f" % PREC)

    print(" Weighted Recall = %0.8f" % REC)

    print(" F1 = %0.8f" % F1)

    print(" Weighted F Measure = %0.8f" % WeightedFMeasure)

    print(" AUC = %.8f" % AUC)

    print(" ROC curve:")

    PredAndLabels           = predictions.select("probability", "label")

    PredAndLabels\_collect   = PredAndLabels.collect()

    PredAndLabels\_list      = [(float(i[0][0]), 1.0-float(i[1])) for i in PredAndLabels\_collect]

    PredAndLabels           = sc.parallelize(PredAndLabels\_list)

    fpr = dict()                                                        # FPR: False Positive Rate

    tpr = dict()                                                        # TPR: True Positive Rate

    roc\_auc = dict()

    y\_test = [i[1] for i in PredAndLabels\_list]

    y\_score = [i[0] for i in PredAndLabels\_list]

    fpr, tpr, \_ = roc\_curve(y\_test, y\_score)

    roc\_auc = auc(fpr, tpr)

    plt.figure(figsize=(5,4))

    plt.plot(fpr, tpr, label='ROC curve (area = %0.8f)' % roc\_auc)

    plt.plot([0, 1], [0, 1], 'r--')

    # plt.xlim([0.0, 1.0])

    # plt.ylim([0.0, 1.05])

    plt.yticks(np.arange(0,1.03,0.1))

    plt.xlabel('False Positive Rate')

    plt.ylabel('True Positive Rate')

    plt.title(f'ROC Curve - {model\_name}')

    plt.legend(loc="lower right")

    plt.show()

برای تجمیع توابع بالا از تابع evaluate استفاده می‌کنیم تا خروجی به صورت یکپارچه چاپ شود.

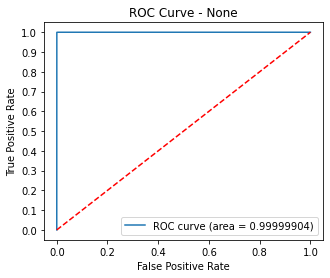
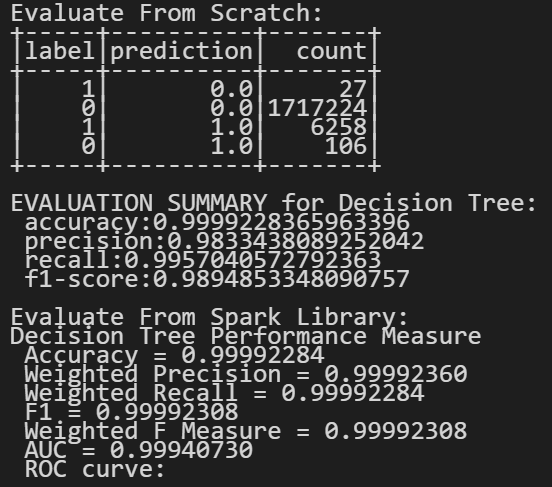
def evaluate(predictions,model\_name=None):

    print('Evaluate From Scratch:')

    evaluate\_from\_scratch(predictions,model\_name)

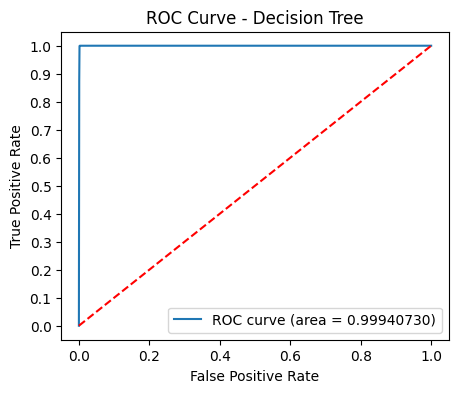
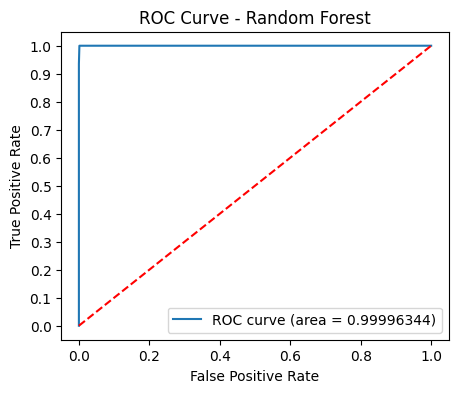
    print('\nEvaluate From Spark Library:')

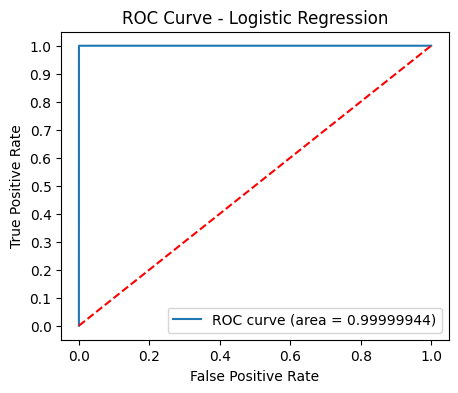
    evaluate\_from\_spark(predictions,model\_name)

نمونه خروجی ارزیابی یک مدل به صورت زیر است:

در جدول زیر سنجه‌های ارزیابی برای مدل‌های مختلف آورده شده است:

|  |  |  |  |
| --- | --- | --- | --- |
| سنجه | رگرسیون منطقی | درخت تصمیم | جنگل تصادفی |
| Accuracy | 0.99998376 | 0.99992284 | 0.99982305 |
| Precision | 0.99840688 | 0.98334381 | 0.99983283 |
| Recall | 0.99713604 | 0.99570406 | 0.95163087 |
| Weighted F1 | 0.99998375 | 0.99992308 | 0.99982087 |
| AUC score | 0.99999943 | 0.99940730 | 0.99996344 |

همچنین منحنی ROC به تفکیک مدل‌ها را نیز به صورت زیر رسم شده است:



مقایسه عملکرد مدل‌ها

برای نشان دادن تفاوت عملکرد مدل‌ها به صورت ملموس‌تر، نمودار میله‌ای عملکرد آن‌ها را به صورت زیر رسم می‌کنیم. مشاهده می‌شود که مدل رگرسیون منطقی از صحت و f1 بالاتری برخوردار است. همچنین امتیاز AUC آن نیز بیشتر بوده که به معنی طبقه‌بندی قابل اعتماد‌تر است.