بسمه تعالی

پروژه پایانی اصول طراحی کامپایلر

استاد : دکتر سعید پارسا

عنوان پروژه:

طراحی و پیاده سازی یک زبان دامنه محور (DSL) برای انتخاب ویژگی ها و مدل های یادگیری ماشین (ML) در تحقیقات مرتبط با بیماری Multiple Sclerosis (MS)

اعضای گروه:

زهرا عباسقلی 400522022

معصومه غفاری 400522085

سانیا مسعودی 401522115

زینب نصیری 401522187

❖ مقدمات:

هدف این پروژه طراحی و پیاده سازی یک زبان دامنه محور (DSL) است که فرآیند انتخاب ویژگی ها، مدل های یادگیری ماشین و تعریف قوانین آموزشی برای تحقیقات MS را ساده سازی کند. این زبان به متخصصان امکان می دهد تا بدون نیاز به دانش برنامه نویسی پیشرفته، مدل های مناسب را انتخاب کرده و ویژگی های مرتبط را برای پیش بینی نتایج پزشکی تعیین کنند.

DSL پیشنهادی قابلیت انتخاب خودکار یا دستی ویژگی ها و مدل ها، انعطاف پذیری در تنظیمات، پشتیبانی از داده های بالینی و تصویری و ارزیابی مدل ها را ارائه خواهد داد.

❖ گام اول: پیاده سازی grammer

برای این پروژه یک گرامر با نام AutoMLDSL تعریف و نوشته شد. گرامر نوشته شده شامل چندین مدل است و هر مدل با کلمه کلیدی model آغاز شده و سپس یک ID به آن اختصاص داده می شود و پس از آیدی، با استفاده از کلمه کلیدی task، وظیفه مدل را تعیین کردیم. عناصر مختلف مدل نیز (مانند متریک‌ها، ویژگی‌ها، و قوانین) در داخل {} تعریف می شوند.

```
model: 'model' ID 'task' task '{' modelElement* '};'
```

پس از تنظیم مدل ها، تسک ها تنظیم شده اند. هر مدل یکی از وظایف پیش بینی، دسته بندی، رگرسیون و خوشه بندی را بر عهده دارد:

```
task: 'predict' | 'classification' | 'regression' | 'clustering;'
```

سپس عناصر مدل تعریف شده اند. مدل های یادگیری ماشین می توانند جنگل تصادفی، درخت تصمیم، SVM، AutoML باشد:

```
mlModelType: 'RandomForest' | 'DecisionTree' | 'SVM' | 'AutoML' |
'KMeans;'
```

سپس پارامترها را تعریف کردیم. هر پارامتر شامل یک نام (name) و یک مقدار (value) است. مقادیر می توانند به صورت عددی (float) یا متنی (string) باشند یا می توانند لیستی از مقادیر را شامل شوند.

```
parameter: 'parameter' 'name' '=' STRING 'value' '=' valueList;
valueList: '[' value (',' value)* ']' | value;
value: STRING | FLOAT;
```

سپس نوبت به تعریف متریک ها رسید. متریک ها برای ارزیابی مدل های یادگیری ماشین استفاده می شوند. متریک های ارزیابی مدل مانند mse (میانگین مربعات خطا)، mae (میانگین قدر مطلق خطاها)، r2_score (ضریب تعیین)،

accuracy (دقت مدل)، precision (نرخ تشخیص موارد مثبت)، recall (نرخ بازیابی)، f1_score (ترکیبی از دقت و بازیابی) و سایر موارد است:

```
metricName: 'mse' | 'r2_score' | 'mae' | 'rmse' | 'accuracy' |
'precision' | 'recall' | 'f1_score' | 'all;'
```

پس از آن تعریف بخش انتخاب ویژگی ها انجام شد. این بخش به کاربر اجازه می‌دهد داده‌های موردنظر خود را برای مدل انتخاب کند و با کلمه کلیدی select آغاز خواهند شد و انتخاب ویژگی با استفاده از شرط ها صورت می گیرد.

```
featureSelection: 'select' ID featureList ('where' conditionList);?
```

برای اینکه بتوان به کاربر اجازه ی تعریف شرط را داد، Rule Set تعریف می شود. اگر شرطی برقرار باشد (مانند یک ویژگی خاص بزرگ‌تر از یک مقدار مشخص باشد)، آنگاه یک یا چندین اقدام خاص انجام می‌شود.

برای شروع فرآیند از کلمه کلیدی start استفاده می کنیم که نشان می دهد فرآیند با کدام مدل آغاز شود:

```
start: 'start' ID 'with' ID;
```

با قانون show می توان به کاربر دسترسی نمایش اطلاعاتی مانند مدل ها، ویژگی ها، متریک ها و ... را داد:

```
show: 'show' visualization (',' visualization)* ('from' ID);?
visualization: 'models' | 'features' | 'metrics' | 'rules;'
```

در آخر توکن های مورد استفاده مانند آیدی ها، استرینگ ها، اعداد اعشاری و ... را تعریف خواهیم کرد:

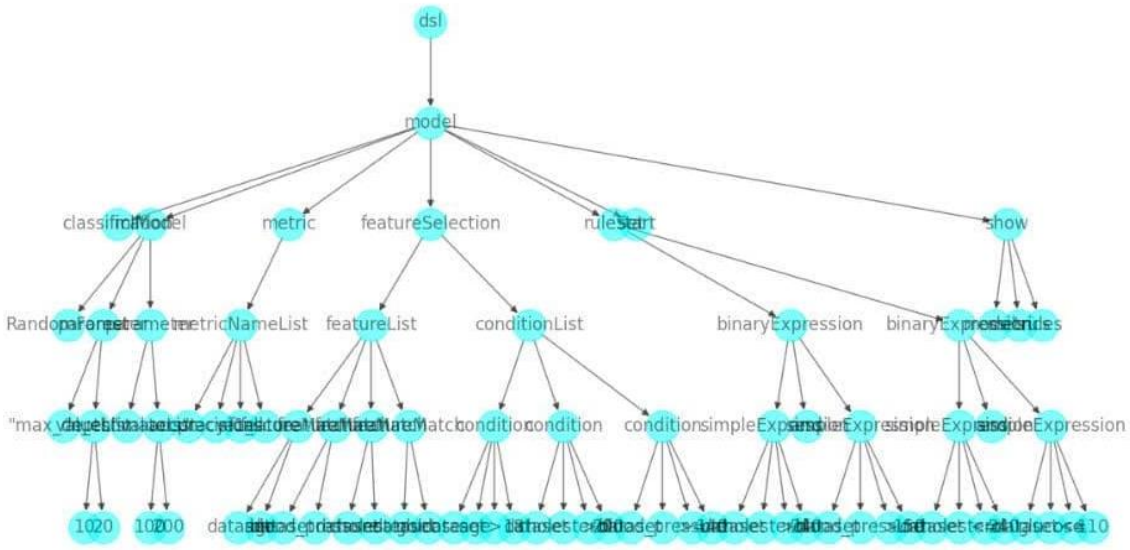
```
ID: [a-zA-Z][a-zA-Z0-9_];*
STRING;'"' ?*. '"' :
FLOAT: [0-9;?(+[0-9]'.') +[
WS: [ \t\r\n]+ -> skip;
COMMENT: '//' ~[\r\n]* -> skip;
```

از گرامر های نوشته شده در بخش بعدی پروژه برای ساخت درخت AST استفاده شده است.

❖ گام دوم: ساخت درخت AST

برای تولید درخت نحوی، از ابزار ANTLR استفاده شد. ANTLR با استفاده از گرامر نوشته‌شده در فایل AutoMLDSL.g4، کلاسی برای تحلیل و تولید درخت نحوی ایجاد می‌کند. این درخت شامل تمام جزئیات زبان از جمله نام مدل‌ها، وظایف (tasks)، متریک‌ها، انتخاب ویژگی‌ها و قوانین شرطی است.

برای هر بخش از گرامر، گره‌های متناظر در AST تعریف شدند. یک مدل یادگیری ماشین (mlModel) به‌عنوان یک گره مستقل در AST ذخیره می‌شود و متریک‌ها و قوانین شرطی نیز به‌صورت گره‌های جداگانه در نظر گرفته می‌شوند.



در فایل AutoMLCustomListener.py کدهای مربوط به ساخت درخت AST نوشته شده اند:

```
def __init__(self, rule_names):
    self.overridden_rules = [
        'dsl', 'model', 'mlModel', 'parameter', 'featureSelection',
        'metric', 'ruleSet', 'start', 'show'
    ]
    self.rule_names = rule_names
    self.ast = AST()

def exitEveryRule(self, ctx):
    rule_name = self.rule_names[ctx.getRuleIndex()]
    if rule_name not in self.overridden_rules:
        make_ast_subtree(self.ast, ctx, rule_name)
```

در لیست overridden_rules گره های درخت AST قرار داده شده اند و سپس در تابع exitEveryRule بررسی می شود که هر متغیر گره (والد) هست یا خیر و اگر نبود عمل ساخت زیردرخت را انجام می دهد.

پس از ساخت درخت AST، بخش لیسنر کد پیاده سازی شده است.

❖ گام سوم: پیاده سازی Listener:

لیسنر از الگوی بازدید کننده (Visitor/Listener Pattern) برای پیمایش درخت نحوی یا AST استفاده می‌کند. در این پروژه، لیسنر با استفاده از ابزار ANTLR تولید شده و وظیفه دارد که دستورات DSL را تفسیر و پردازش کند.

لیسنر اطلاعات را از گره های AST استخراج کرده، سپس داده ها را در ساختارهایی مانند دیکشنری یا لیست ذخیره می کند و اطلاعات را به مولد کد انتقال می دهد.

❖ گام چهارم: مولد کد

مولد کد پس از پردازش لیسنر، کدهای قابل اجرا تولید می‌کند. این کدها وظایف تعریف‌شده در زبان DSL را پیاده‌سازی می‌کنند.

در نهایت کد تولید شده در یک فایل نهایی ذخیره خواهد شد.

در فایل AutoMLCodeGenerator.py ابتدا از چند پشته برای ذخیره اطلاعات مربوط به هر بخش استفاده شده است. در پشته code_stack کدهای میانی را نگهداری می کنیم. در پشته operand_stack تمام اطلاعات ورودی ذخیره می شود.

سپس در تابع generate_code() با استفاده از پیمایش post-order که قبلا انجام شده و در post_order_array نگهداری شده، اطلاعات استخراج شده درخت را در استک های مربوطه (مانند feature_stack یا operand_stack ذخیره می کنیم.

```
def generate_code(self, post_order_array):
    for item in post_order_array:
        if not self.is_operand(item['label']):
            self.generate_code_based_on_non_operand(item['label'], item)
        else:
            if item['label'] == 'featureMatch':
                if self.prev_item:
                    self.feature_stack.append(self.prev_item['label'])
            else:
                self.operand_stack.append(item['label'])
            self.prev_item = item

    return self.generate_program()
```

سپس تمام کدهای میانی تولید می شوند. پس از تولید کدهای میانی، در تابع `generate_program()` کد نهایی را تولید می کنیم.

```python
def generate_program(self):
    imports = self._generate_imports()
    dataset_code = self._generate_dataset()
    model_training_code = self._generate_model_training()

    program_code = imports + dataset_code + model_training_code
    return program_code
```

❖ گام پنجم: تست کیس ها

1. ورودی اول:

example3_input.dsl

```
model PatientRiskPrediction task classification {
    mlModel RFModel type RandomForest {
        parameter name = "max_depth" value = [10, 20]
        parameter name = "n_estimators" value = [100, 200]
    }

    metric accuracy, precision, recall, f1_score

    select featureSet
        dataset.age, dataset.blood_pressure, dataset.cholesterol, dataset.glucose
    where dataset.age > 18 and dataset.cholesterol > 200 or dataset.blood_pressure >= 140

    ruleSet RiskRules {
        rule: if (dataset.cholesterol > 240 and dataset.blood_pressure > 150) then high_risk, immediate_attention
        rule: if (dataset.cholesterol <= 240 and dataset.glucose <= 110) then low_risk, routine_check
    }

    start PatientRiskPrediction with RFModel

    show models, metrics, rules from PatientRiskPrediction
}
```

خروجی اول:

output.py > ...

```python
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Load dataset
X = data['age', 'blood_pressure', 'cholesterol', 'glucose']
y = data['target']

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train model
model.fit(X_train, y_train)

# Evaluate model
predictions = model.predict(X_test)
accuracy = accuracy_score(y_test, predictions)
precision = precision_score(y_test, predictions, average='weighted')
recall = recall_score(y_test, predictions, average='weighted')
f1 = f1_score(y_test, predictions, average='weighted')

print(f"Accuracy: {accuracy}, Precision: {precision}, Recall: {recall}, F1 Score: {f1}")

```

2. ورودی دوم:

example_input.dsl

```
model AutoMLRegressionModel task regression {
    mlModel autoML type AutoML {
        parameter name = "generation" value = 100
        parameter name = "population_size" value = 50
    }
    select feature
        data.t2lesvol,
        data.t2overbv,
        data.t2voljux
    metric mse, r2_score
    start autoML with feature
    show models, metrics
}
```

خروجی دوم:

example_output.py > ...

```python
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Load dataset
X = data['t2lesvol', 't2overbv', 't2voljux']
y = data['target']

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train model
model.fit(X_train, y_train)

# Evaluate model
predictions = model.predict(X_test)
accuracy = accuracy_score(y_test, predictions)
precision = precision_score(y_test, predictions, average='weighted')
recall = recall_score(y_test, predictions, average='weighted')
f1 = f1_score(y_test, predictions, average='weighted')

print(f"Accuracy: {accuracy}, Precision: {precision}, Recall: {recall}, F1 Score: {f1}")

```