AutoML Agent Interface

Select the dataset type

tabular

Choose a dataset

Iris

Select a GROQ LLM Model

meta-llama/llama-4-maverick-17b-128e-instruct

Run AutoML Agent

Generated Configuration Space Code

```
from ConfigSpace import ConfigurationSpace, Categorical, Float, Integer, EqualsCon

def get_configspace():
    cs = ConfigurationSpace(seed=1234)

    classifier = Categorical('classifier', ['svm', 'rf', 'knn'])
    cs.add_hyperparameter(classifier)

    C = Float('C', (1e-5,1e5), default=1, log=True)
    cs.add_hyperparameter(C)
    kernel = Categorical('kernel', ['linear', 'poly', 'rbf', 'sigmoid'])
    cs.add_hyperparameter(kernel)
    degree = Integer('degree', (2,5), default=3)
    cs.add_hyperparameter(degree)
    gamma = Float('gamma', (1e-5,1e2), default=1, log=True)
    cs.add_hyperparameter(gamma)
    coef0 = Float('coef0', (0,1), default=0)
    cs.add_hyperparameter(coef0)

    n_estimators = Integer('n_estimators', (10,1000), default=100)
    cs.add_hyperparameter(n_estimators)
    max_depth = Integer('max_depth', (1,10), default=5)
    cs.add_hyperparameter(max_depth)
```

```
    n_neighbors = Integer('n_neighbors', (1,100), default=5)
    cs.add_hyperparameter(n_neighbors)
    weights = Categorical('weights', ['uniform', 'distance'])
    cs.add_hyperparameter(weights)

    cond_1 = EqualsCondition(C, classifier, 'svm')
    cond_2 = EqualsCondition(kernel, classifier, 'svm')
    cond_6 = EqualsCondition(n_estimators, classifier, 'rf')
    cond_7 = EqualsCondition(max_depth, classifier, 'rf')
    cond_8 = EqualsCondition(n_neighbors, classifier, 'knn')
    cond_9 = EqualsCondition(weights, classifier, 'knn')

    cond_degree = AndConjunction(EqualsCondition(degree, classifier, 'svm'), InCon
    cond_coef0 = AndConjunction(EqualsCondition(coef0, classifier, 'svm'), InCondi
    cond_gamma = AndConjunction(EqualsCondition(gamma, classifier, 'svm'), InCondi

    cs.add_condition(cond_1)
    cs.add_condition(cond_2)
    cs.add_condition(cond_6)
    cs.add_condition(cond_7)
    cs.add_condition(cond_8)
    cs.add_condition(cond_9)
    cs.add_condition(cond_degree)
    cs.add_condition(cond_coef0)
    cs.add_condition(cond_gamma)

    return cs
```

Generated Scenario Code

```
from smac import Scenario

def generate_scenario(cs):
    scenario = Scenario(
        configspace=cs,
        objectives=["validation_loss"],
        output_directory="./automl_results",
        deterministic=False,
        n_workers=4,
        min_budget=1,
        max_budget=100,
    )
    return scenario
```

Generated Training Function Code

```python
from ConfigSpace import Configuration
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import svm, ensemble, neighbors
import torch
import torch.nn as nn
import torch.optim as optim
from typing import Any

def train(cfg: Configuration, seed: int, dataset: Any) -> float:
    X = dataset['X']
    y = dataset['y']

    # Check if the data is image-shaped
    is_image_data = len(X.shape) == 4

    # Infer input and output dimensions dynamically
    if is_image_data:
        input_size = X.shape[1] * X.shape[2] * X.shape[3]
    else:
        input_size = X.shape[1]
    num_classes = len(np.unique(y))

    # Split the dataset into training and validation sets
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_

    # Scale the data if not image data
    if not is_image_data:
        scaler = StandardScaler()
        X_train = scaler.fit_transform(X_train)
        X_val = scaler.transform(X_val)

    classifier = cfg.get('classifier')

    if classifier in ['svm', 'rf', 'knn']:
        # Use scikit-learn models
        if classifier == 'svm':
            model = svm.SVC(
                C=cfg.get('C'),
                kernel=cfg.get('kernel'),
                degree=cfg.get('degree') if cfg.get('kernel') == 'poly' else 3,
                gamma=cfg.get('gamma') if cfg.get('kernel') in ['rbf', 'poly', 'si
```

```
                coef0=cfg.get('coef0') if cfg.get('kernel') in ['poly', 'sigmoid']
                random_state=seed
            )
        elif classifier == 'rf':
            model = ensemble.RandomForestClassifier(
                n_estimators=cfg.get('n_estimators'),
                max_depth=cfg.get('max_depth'),
                random_state=seed
            )
        else:  # classifier == 'knn'
            model = neighbors.KNeighborsClassifier(
                n_neighbors=cfg.get('n_neighbors'),
                weights=cfg.get('weights')
            )

        model.fit(X_train, y_train)
        loss = 1 - model.score(X_val, y_val)
        return loss

    else:
        # Use PyTorch for neural networks
        if is_image_data:
            # Reshape the data
            X_train = X_train.reshape(-1, X_train.shape[1], X_train.shape[2], X_tr
            X_val = X_val.reshape(-1, X_val.shape[1], X_val.shape[2], X_val.shape[
            input_size = X_train.shape[1]

            # Check if the input size is a perfect square
            if X_train.shape[2] * X_train.shape[3] != X_train.shape[2] ** 2:
                raise ValueError("Input size is not a perfect square.")

            X_train = torch.tensor(X_train, dtype=torch.float32)
            X_val = torch.tensor(X_val, dtype=torch.float32)
            y_train = torch.tensor(y_train, dtype=torch.long)
            y_val = torch.tensor(y_val, dtype=torch.long)

            # Define the CNN model
            model = nn.Sequential(
                nn.Conv2d(X_train.shape[1], 10, kernel_size=5),
                nn.ReLU(),
                nn.MaxPool2d(2, 2),
                nn.Flatten(),
                nn.Linear(320, 50),
                nn.ReLU(),
                nn.Linear(50, num_classes)
            )
```

```
    else:
        X_train = torch.tensor(X_train, dtype=torch.float32)
        X_val = torch.tensor(X_val, dtype=torch.float32)
        y_train = torch.tensor(y_train, dtype=torch.long)
        y_val = torch.tensor(y_val, dtype=torch.long)

        # Define the MLP model
        model = nn.Sequential(
            nn.Linear(input_size, 128),
            nn.ReLU(),
            nn.Linear(128, num_classes)
        )

    # Define the optimizer
    learning_rate = cfg.get('learning_rate', default=0.01)
    optimizer_type = cfg.get('optimizer', default='constant')

    if optimizer_type == 'constant':
        optimizer = optim.SGD(model.parameters(), lr=learning_rate)
    elif optimizer_type == 'invscaling':
        optimizer = optim.SGD(model.parameters(), lr=learning_rate, momentum=0
    elif optimizer_type == 'adaptive':
        optimizer = optim.Adam(model.parameters(), lr=learning_rate)

    # Train the model
    loss_fn = nn.CrossEntropyLoss()
    for epoch in range(10):
        model.train()
        optimizer.zero_grad()
        outputs = model(X_train)
        loss = loss_fn(outputs, y_train)
        loss.backward()
        optimizer.step()

    # Evaluate the model
    model.eval()
    with torch.no_grad():
        outputs = model(X_val)
        loss = loss_fn(outputs, y_val).item()

    return loss
```

AttributeError: 'HyperparameterOptimizationFacade' object has no attribute 'get_runhistory'

Traceback:

```
File "/Users/amirrezaalasti/Desktop/master/semester 2/AutoML-Agent/main.py", l
    app_ui.display()
File "/Users/amirrezaalasti/Desktop/master/semester 2/AutoML-Agent/scripts/Aut
    agent.generate_components()
File "/Users/amirrezaalasti/Desktop/master/semester 2/AutoML-Agent/scripts/Aut
    self.run_scenario(self.scenario_obj, train)
File "/Users/amirrezaalasti/Desktop/master/semester 2/AutoML-Agent/scripts/Aut
    runhistory = smac.get_runhistory()
```

Ask Google Ask ChatGPT