STT Lab 06 - Weights & Biases and Autogluon

Github repo link : https://github.com/Soham-Gaonkar/CS_Lab_06

Group 7

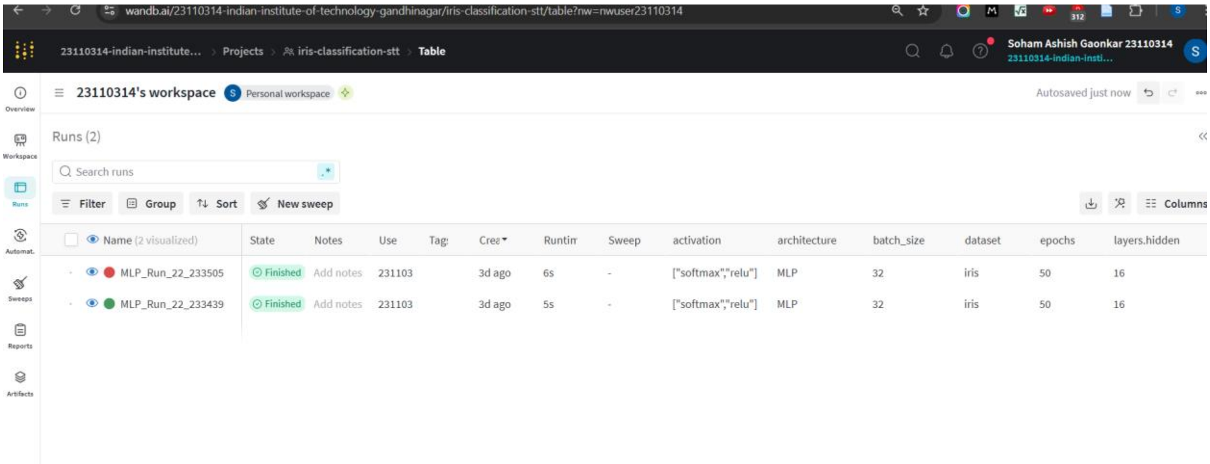
1.Soham Gaonkar 23110314

2.Chaitanya Sharma 23110072

Section 1: Weights & Biases

Results:

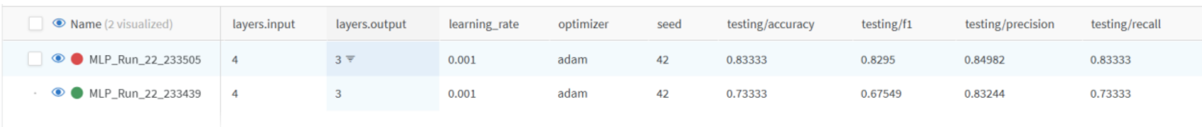
1. Training hyperparameters:



Runs (2)

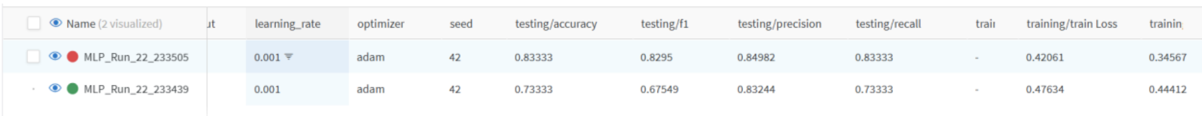
Name (2 visualized)	State	Notes	Use	Tag:	Crea	Runtir	Sweep	activation	architecture	batch_size	dataset	epochs	layers.hidden
MLP_Run_22_233505	Finished	Add notes	231103		3d ago	6s	-	["softmax","relu"]	MLP	32	iris	50	16
MLP_Run_22_233439	Finished	Add notes	231103		3d ago	5s	-	["softmax","relu"]	MLP	32	iris	50	16

2. Model architecture:



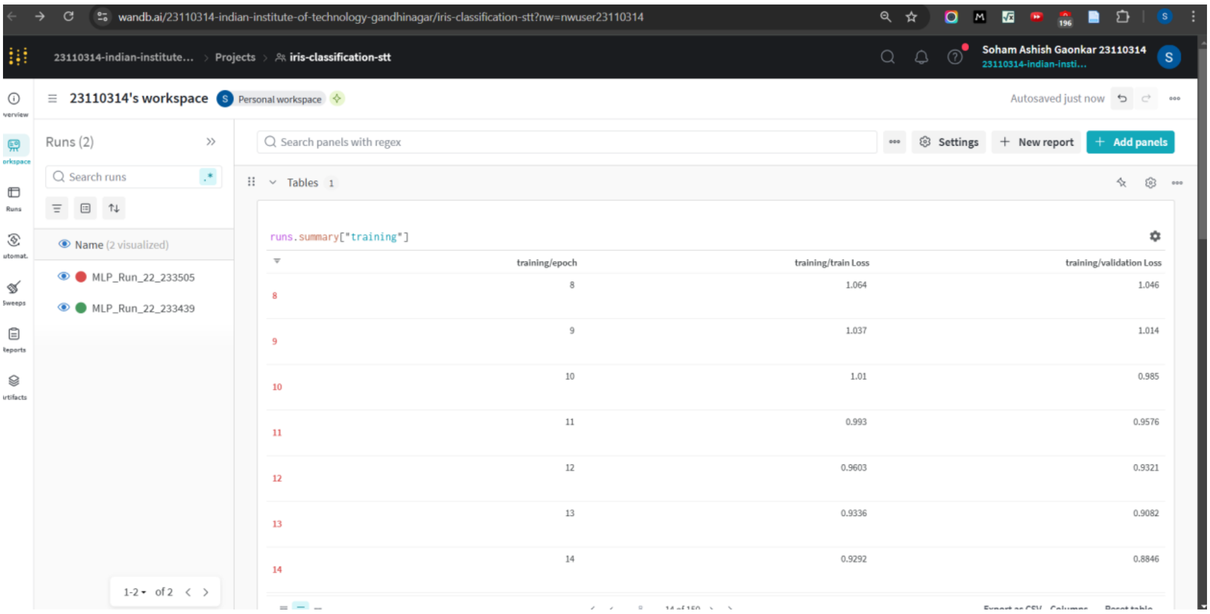
Name (2 visualized)	layers.input	layers.output	learning_rate	optimizer	seed	testing/accuracy	testing/f1	testing/precision	testing/recall
MLP_Run_22_233505	4	3	0.001	adam	42	0.83333	0.8295	0.84982	0.83333
MLP_Run_22_233439	4	3	0.001	adam	42	0.73333	0.67549	0.83244	0.73333

3. Training and testing metrics:

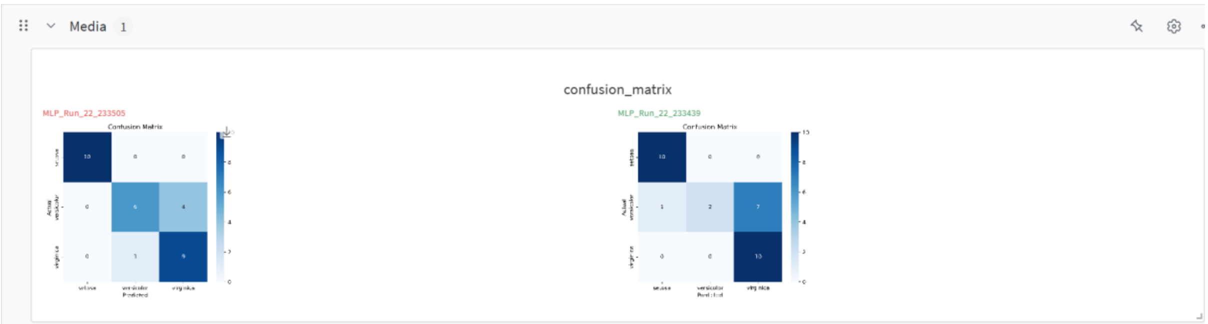


Name (2 visualized)	ıt	learning_rate	optimizer	seed	testing/accuracy	testing/f1	testing/precision	testing/recall	trair	training/train Loss	trainin
MLP_Run_22_233505		0.001	adam	42	0.83333	0.8295	0.84982	0.83333	-	0.42061	0.34567
MLP_Run_22_233439		0.001	adam	42	0.73333	0.67549	0.83244	0.73333	-	0.47634	0.44412

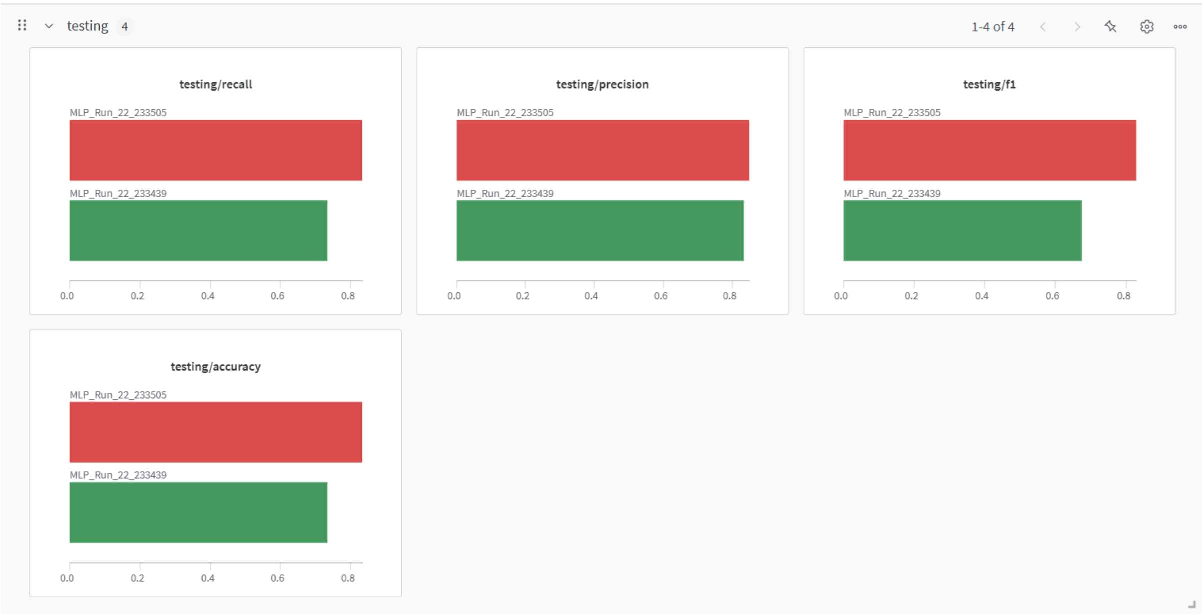
4. Training and validation loss table:



5. Confusion matrix:



6. Evaluation metrics:



7. Graphs of training and validation loss:

Code:

Wandb init :

```
from datetime import datetime

# run name consisting of timestamp only with date and time in the format DD_HHMMSS
run_name = f"MLP_Run_{datetime.now().strftime('%d_%H%M%S')}"

wandb.init(
    # Set the project where this run will be logged
    project="iris-classification-stt",
    # Set the name of the run
    name=run_name,
    # Hyperparameters
    config= {
    "architecture": "MLP",
    "learning_rate": 0.001,
    "epochs": 50,
    "batch_size": 32,
    "layers":{
        "input": 4,
        "hidden": 16,
        "output": 3
    },
    "dataset": "iris",
    "seed":42,
    "activation": {"relu","softmax"},
    "optimizer": "adam",
    })

config = wandb.config
```

Train – test – split :

Train - Val - Test Split (70-10-20)

```
# Train-test split 80 20
from sklearn.model_selection import train_test_split

X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.2, random_state=42,stratify = y)
X_train,X_val , y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=(0.1/0.8), random_state=42,stratify = y_train_val)
print('X_train shape:', X_train.shape)
print('X_test shape:', X_test.shape)
print('X_val shape:', X_val.shape)

print('y_train shape:', y_train.shape)
print('y_test shape:', y_test.shape)
print('y_val shape:', y_val.shape)
```

```
X_train shape: (105, 4)
X_test shape: (30, 4)
X_val shape: (15, 4)
y_train shape: (105,)
y_test shape: (30,)
y_val shape: (15,)
```

Scale:

Standard Scaler

```
# Normalize the data
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)
```

[27]

Use Min Max Scaler for making data from [0,1]

Model architecture:

Model

```python
# Define the model
import torch.nn as nn

class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.hidden = nn.Linear(4, 16)  # Hidden layer with 16 neurons
        self.output = nn.Linear(16, 3)  # Output layer (3 classes)

    def forward(self, x):
        x = torch.relu(self.hidden(x))  # ReLU activation
        x = self.output(x) # no softmax here since we'll use nn.CrossEntropyLoss which applies softmax
        z = nn.functional.softmax(x, dim=1)
        return x # logits

# Initialize the model
model = MLP()
print(model)
```

```
MLP(
  (hidden): Linear(in_features=4, out_features=16, bias=True)
  (output): Linear(in_features=16, out_features=3, bias=True)
)
```

Training :

```python
# Loss function and optimizer
import torch.optim as optim

# Training the model
model = MLP()

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training the model
n_epochs = 50
train_losses = []
val_losses = []

table = wandb.Table(columns=["training/epoch", "training/train Loss", "training/validation Loss"])

for epoch in range(n_epochs):
    model.train()
    epoch_train_loss = 0
    for batch_X, batch_y in train_loader:
        optimizer.zero_grad()
        outputs = model(batch_X)
        loss = criterion(outputs, batch_y)
        loss.backward()
        optimizer.step()
        epoch_train_loss += loss.item()

    # Compute validation loss
    model.eval()
    epoch_val_loss = 0
    with torch.no_grad():
        for val_X, val_y in val_loader:
            val_outputs = model(val_X)
            val_loss = criterion(val_outputs, val_y)
            epoch_val_loss += val_loss.item()

    # Store average losses
    train_loss = epoch_train_loss / len(train_loader)
    val_loss = epoch_val_loss / len(val_loader)
    train_losses.append(train_loss)
    val_losses.append(val_loss)

    # Log to W&B
    training_metrics = {"training/train Loss": train_loss, "training/validation Loss": val_loss}
    wandb.log(training_metrics)

    table.add_data(epoch+1, train_loss, val_loss)

    # Print progress every 10 epochs
    if (epoch+1) % 10 == 0:
        print(f"Epoch [{epoch+1}/{n_epochs}], Train Loss: {train_loss:.4f}, Validation Loss: {val_loss:.4f}")


wandb.log({"training": table})
```

```
Epoch [10/50], Train Loss: 0.9120, Validation Loss: 0.8459
Epoch [20/50], Train Loss: 0.7733, Validation Loss: 0.6917
Epoch [30/50], Train Loss: 0.5946, Validation Loss: 0.5701
Epoch [40/50], Train Loss: 0.5440, Validation Loss: 0.4880
Epoch [50/50], Train Loss: 0.4971, Validation Loss: 0.4299
```

Evaluating:

```python
# Compute metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

accuracy = accuracy_score(y_true, y_pred)
precision = precision_score(y_true, y_pred, average="macro")
recall = recall_score(y_true, y_pred, average="macro")
f1 = f1_score(y_true, y_pred, average="macro")

print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-score: {f1:.4f}")

testing_metrics = {"testing/accuracy": accuracy, "testing/precision": precision, "testing/recall": recall, "testing/f1": f1}
wandb.log(testing_metrics)

# Confusion Matrix
conf_matrix = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=iris.target_names, yticklabels=iris.target_names)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")

plt.savefig("confusion_matrix.png", bbox_inches="tight", dpi=300)  # High quality save
plt.show()


confusion_matrix_metrics = {"confusion_matrix": wandb.Image("confusion_matrix.png")}
wandb.log(confusion_matrix_metrics)

wandb.finish()
```

```
Accuracy: 0.7667
Precision: 0.8627

Recall: 0.7667
F1-score: 0.7341
```

Logs:

Run history:

testing/accuracy	
testing/f1	
testing/precision	
testing/recall	
training/train Loss	
training/validation Loss	

Run summary:

testing/accuracy	0.76667
testing/f1	0.73409
testing/precision	0.86275
testing/recall	0.76667
training/train Loss	0.49709
training/validation Loss	0.42991

View run **MLP_Run_25_142349** at: https://wandb.ai/23110314-indian-institute-of-technology-gandhinagar/iris-classification-stt/runs/5a5co1fk
View project at: https://wandb.ai/23110314-indian-institute-of-technology-gandhinagar/iris-classification-stt
Synced 5 W&B file(s), 2 media file(s), 2 artifact file(s) and 0 other file(s)

Find logs at: .\wandb\run-20250225_142349-5a5co1fk\logs

Section 2 :

Task 1:

Code:

```python
import autogluon.core as ag

def train_manual_mode(train_data,val_data, batch_size, learning_rate, num_epochs):
    """Train model using manual hyperparameters in AutoGluon."""

    nn_options = {
        'num_epochs': num_epochs,
        'learning_rate': learning_rate,
        'batch_size': batch_size,
        'hidden_size': 16,
        'num_layers': 2,
        'activation': 'relu'
    }

    hyperparameters = {
        'NN_TORCH': nn_options
    }

    model_path = f'models/manual_training/bs{batch_size}_lr{learning_rate}_epochs{num_epochs}'

    predictor = TabularPredictor(
        label='target',
        problem_type='multiclass',
        verbosity=1,
        eval_metric='accuracy',
        path=model_path,
    ).fit(
        train_data=train_data,
        tuning_data=val_data,
        use_bag_holdout=True,  # Use validation data for tuning
        # disable bagging
        num_bag_folds=0,
        # disable weighted ensembling
        num_stack_levels=0,
        fit_weighted_ensemble =False,
        fit_full_last_level_weighted_ensemble= False,

        verbosity=2,
        hyperparameters=hyperparameters,
        time_limit=20*60,
        presets='best_quality',
    )

    return predictor
```

0.0s

Evaluating Model:

```python
from autogluon.tabular import TabularPredictor
from sklearn.metrics import confusion_matrix, classification_report, f1_score
from autogluon.common import space

def evaluate_model(predictor, test_data):
    """Evaluate the trained model on test data."""
    print("Evaluating model...")
    y_pred = predictor.predict(test_data.drop(columns=['target']))
    y_true = test_data['target']

    accuracy = (y_pred == y_true).mean()
    f1 = f1_score(y_true, y_pred, average='weighted')

    print(f"Accuracy: {accuracy:.4f}, F1 Score: {f1:.4f}\n")

    # Confusion Matrix Plot
    plt.figure(figsize=(6, 5))
    cm = confusion_matrix(y_true, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=predictor.class_labels, yticklabels=predictor.class_labels)
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.title("Confusion Matrix")
    plt.show()

    # Print classification report
    print(classification_report(y_true, y_pred))

    # Show 5 sample predictions
    sample_indices = [1, 5, 10, 15, 20]
    for i in sample_indices:
        print(f"Sample {i}: Truth: {y_true.iloc[i]}, Predicted: {y_pred.iloc[i]}")

    print('-'*50)

    return accuracy, f1
```

0.0s

Iterating over all possible configurations:

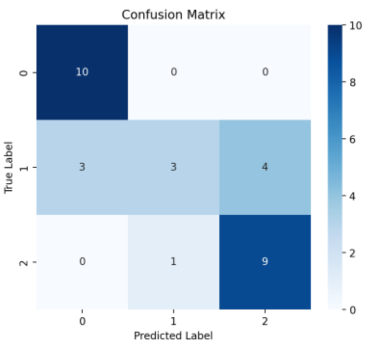
```python
import os
import warnings
warnings.filterwarnings('ignore')

batch_sizes = [2, 4]
learning_rates = [1e-3, 1e-5]
epochs_list = [1, 3, 5]
# batch_sizes = [2]
# learning_rates = [1e-3]
# epochs_list = [3]

predictors = []
retrain = True
predictor_params = {}

results_table = pd.DataFrame(columns=['Batch Size', 'Learning Rate', 'Epochs', 'Test Accuracy','Validation Accuracy', 'F1 Score'])

for batch_size in batch_sizes:
    for lr in learning_rates:
        for epochs in epochs_list:
            if os.path.exists(f'models/manual_training/bs{batch_size}_lr{lr}_epochs{epochs}/predictor.pkl') and not retrain:
                print(f"Model already trained with Batch Size: {batch_size}, Learning Rate: {lr}, Epochs: {epochs}")
                predictor = TabularPredictor.load(f'models/manual_training/bs{batch_size}_lr{lr}_epochs{epochs}')
            else:
                print(f"Training with Batch Size: {batch_size}, Learning Rate: {lr}, Epochs: {epochs}")
                predictor = train_manual_mode(train_df,val_df,batch_size, lr, epochs)

            predictor.fit_summary()

            accuracy,f1 = evaluate_model(predictor,test_df)
            results_table.loc[len(results_table)] = {
                'Batch Size': batch_size,
                'Learning Rate': lr,
                'Epochs': epochs,
                'Test Accuracy': accuracy,
                'Validation Accuracy':predictor.leaderboard().loc[0, 'score_val'],
                'F1 Score': f1
            }
            print('Validation Accuracy:',predictor.leaderboard().loc[0, 'score_val'])
            predictors.append(predictor)
```
✓ 9.7s

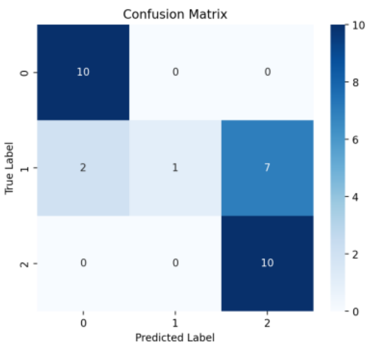
Results Table:

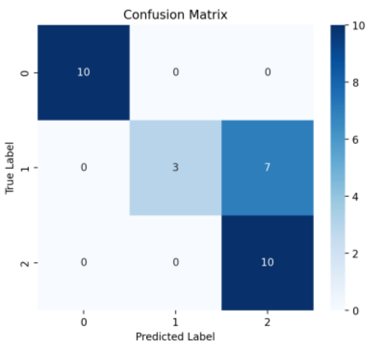
	# Batch Size	# Learning Rate	# Epochs	# Train Accuracy	# Test Accuracy	# Validation Accuracy	# F1 Score	Model
0	2	0.001	1	0.7238095238095238	0.7333333333333333	0.8	0.3292753623188406	NeuralNetTorch
1	2	0.001	3	0.8095238095238095	0.7	0.8	0.3292753623188406	NeuralNetTorch
2	2	0.001	5	0.8666666666666667	0.7666666666666667	0.8666666666666667	0.3292753623188406	NeuralNetTorch
3	2	1e-05	1	0.3904761904761905	0.4	0.26666666666666666	0.3292753623188406	NeuralNetTorch
4	2	1e-05	3	0.3904761904761905	0.4	0.3333333333333333	0.3292753623188406	NeuralNetTorch
5	2	1e-05	5	0.3904761904761905	0.4	0.3333333333333333	0.3292753623188406	NeuralNetTorch
6	4	0.001	1	0.6095238095238096	0.5666666666666667	0.6666666666666666	0.3292753623188406	NeuralNetTorch
7	4	0.001	3	0.8095238095238095	0.6666666666666666	0.7333333333333333	0.3292753623188406	NeuralNetTorch
8	4	0.001	5	0.8095238095238095	0.7	0.8	0.3292753623188406	NeuralNetTorch
9	4	1e-05	1	0.3904761904761905	0.4	0.26666666666666666	0.3292753623188406	NeuralNetTorch
10	4	1e-05	3	0.3904761904761905	0.4	0.26666666666666666	0.3292753623188406	NeuralNetTorch
11	4	1e-05	5	0.3904761904761905	0.4	0.3333333333333333	0.3292753623188406	NeuralNetTorch

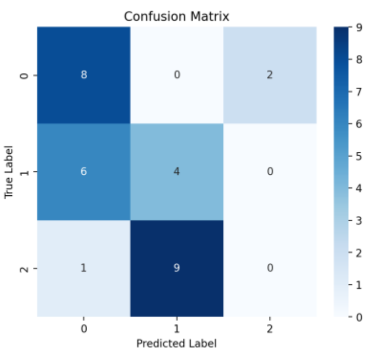
12 rows x 8 cols 25 per page Page 1 of 1

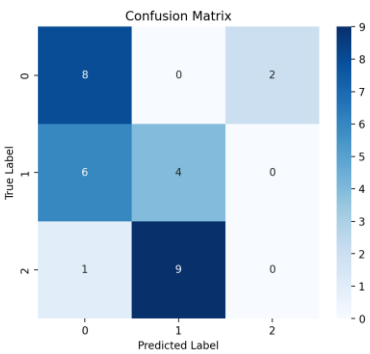
Confusion matrices:

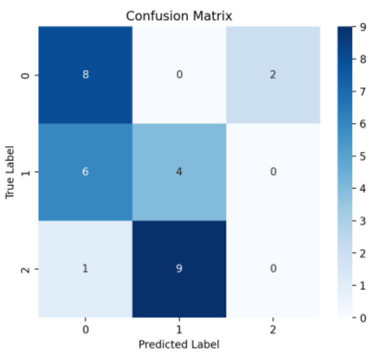


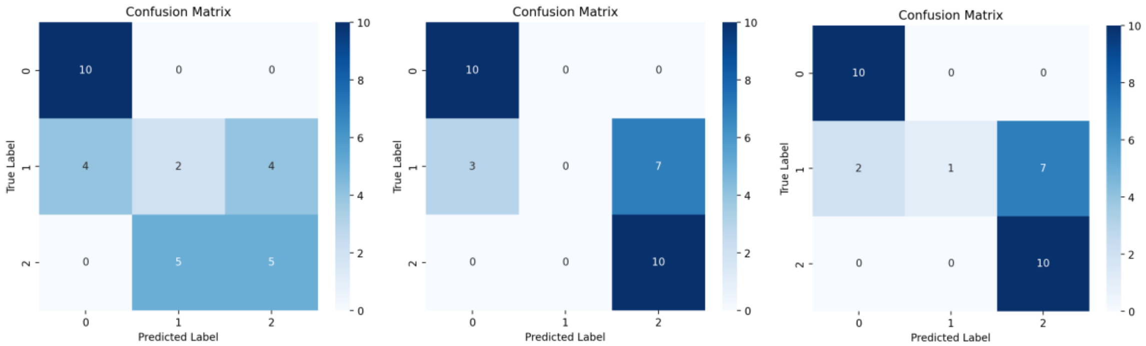








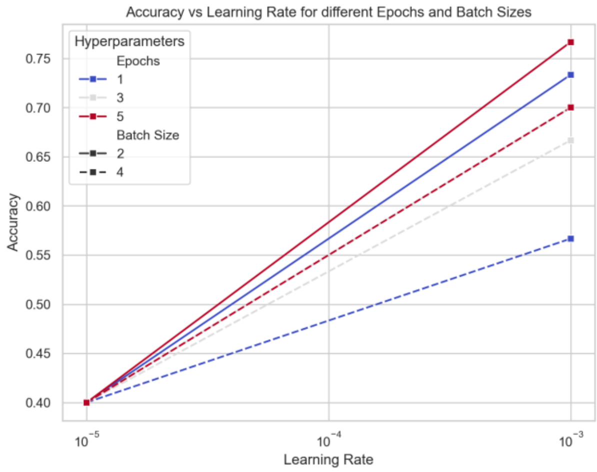




Overall Results Comparison:

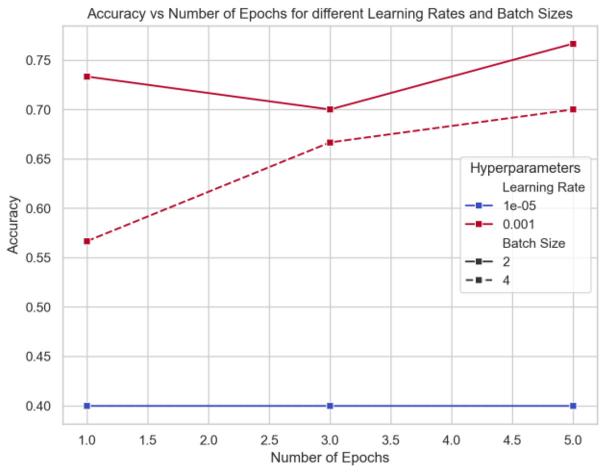
Accuracy vs Learning Rate:

As learning rate increases (for smaller values) , accuracy seems to rise.(direct)



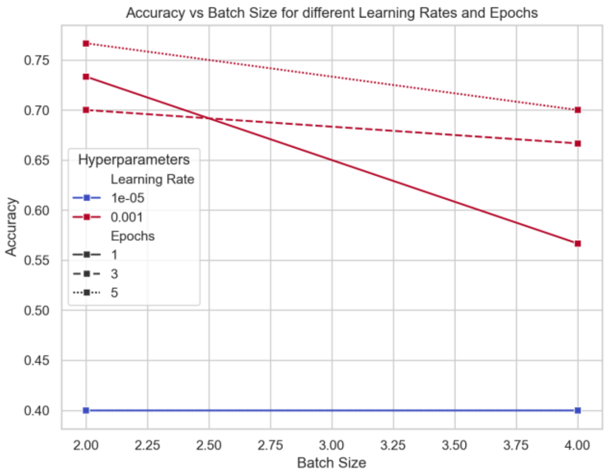
Accuracy vs Number of Epochs:

As number of epochs increases , for a good learning rate , accuracy increases.(direct)



Accuracy vs Batch Size:

With increase in batch size, accuracy decreases. (inverse)



Task 2:

Code:

Training:

```python
def train_auto_mode(train_data,val_data,search_strategy,num_trials,scheduler):

    model_path = f'models/auto_training/{search_strategy}'
    if(search_strategy =='grid'):
        search_strategy = 'random'
        nn_options = {
        'num_epochs': ag.space.Categorical(2,3,4),
        'learning_rate': ag.space.Categorical(1e-4,1e-2),
        'batch_size': ag.space.Categorical(2,3,4),
        'activation': 'relu',
        'hidden_size': 16,
        'num_layers': 2,
        }
        num_trials = 3*3*2
    else:
        nn_options = {
            'num_epochs': ag.space.Int(lower=2, upper=4),
            'learning_rate': ag.space.Real(lower=1e-4,upper=1e-2),
            'batch_size': ag.space.Int(lower=2,upper=4),
            'activation': 'relu',
            'hidden_size': 16,
            'num_layers': 2,
        }

    hyperparameters = {
                    'NN_TORCH': nn_options,
                    }

    time_limit = 120*60 # 2 mins

    hyperparameter_tune_kwargs = {
        'num_trials': num_trials,
        'scheduler' : scheduler,
        'searcher': search_strategy,
    }

    predictor = TabularPredictor(
        label='target',
        eval_metric='accuracy',
        problem_type='multiclass',
        path = model_path).fit(
        train_data=train_data,

        tuning_data=val_data,
        use_bag_holdout=True,  # Use validation data for tuning

        # disable bagging
        num_bag_folds=0,

        # disable weighted ensembling
        num_stack_levels=0,
        fit_weighted_ensemble =False,
        fit_full_last_level_weighted_ensemble= False,

        verbosity=2,
        hyperparameters=hyperparameters,
        hyperparameter_tune_kwargs=hyperparameter_tune_kwargs,
        time_limit=time_limit,
        )

    return predictor
```

✓ 0.0s

Calling function:

```python
results_table_auto = pd.DataFrame(columns=['Batch Size', 'Learning Rate', 'Epochs', 'Test Accuracy','Validation Accuracy', 'F1 Score'])
predictors_auto = []

def train_eval_auto_model(train_data,val_data,search_strategy,num_trials,scheduler):
    retrain = True
    if os.path.exists(f'models/auto_training/{search_strategy}/predictor.pkl') and not retrain:
        print(f"Model already trained with {search_strategy}")
        predictor = TabularPredictor.load(f'models/auto_training/{search_strategy}')
    else:
        print(f"Training with {search_strategy}")
        predictor = train_auto_mode(train_data,val_data,search_strategy,num_trials,scheduler)

    predictor.fit_summary()

    accuracy,f1 = evaluate_model(predictor,test_df)

    model_name = predictor._trainer.load_model(predictor.leaderboard().loc[0, 'model'])
    val_score = predictor.leaderboard().loc[0, 'score_val']

    hp = model_name.params

    results_table_auto.loc[len(results_table_auto)] = {
                'Batch Size': hp['batch_size'],
                'Learning Rate': hp['learning_rate'],
                'Epochs': hp['num_epochs'],
                'Test Accuracy': accuracy,
                'Validation Accuracy':val_score,
                'F1 Score': f1,
                'Search Strategy': search_strategy,
                'Scheduler': scheduler
            }
    predictors_auto.append(predictor)
    return predictor
```
✓ 0.0s

Confusion matrices:

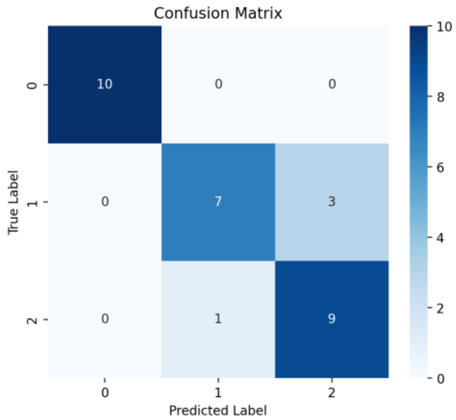
Random Search:

Random search is a hyperparameter optimization technique that randomly selects a combination of hyperparameters from a given range and evaluates them using a validation set.

5

	# Batch Size	# Learning Rate	# Epochs	# Train Accuracy	# Test ...	# Validation Accuracy	# F1 Score	Search Strategy	Sche...	Model
0	3	0.0036524217928226746	3	0.9142857142857143	0.8666666666	1.0	0.3292753623188406	random	FIFO	NeuralNetTorch\1426b_00002
1	3	0.007501332863744887	4	0.8857142857142857	0.8	0.8666666666666667	0.3292753623188406	random	FIFO	NeuralNetTorch\1426b_00003
2	4	0.005474208175072749	2	0.8666666666666667	0.8	0.8666666666666667	0.3292753623188406	random	FIFO	NeuralNetTorch\1426b_00004
3	4	0.007973146387412913	2	0.7904761904761904	0.7	0.8666666666666667	0.3292753623188406	random	FIFO	NeuralNetTorch\1426b_00001
4	2	0.0001	2	0.4380952380952381	0.4333333333	0.4	0.3292753623188406	random	FIFO	NeuralNetTorch\1426b_00000

5 rows x 10 cols 10 per page Page 1 of 1



Grid Search:

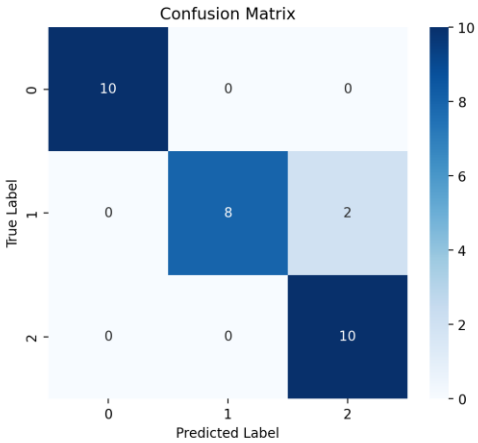
Grid search is a hyperparameter optimization technique that exhaustively searches through a grid of hyperparameters to find the best combination.

18

	# Batch Size	# Learning Rate	# Epochs	# Train Accuracy	# Test Accuracy	# Validation Accuracy	# F1 Score	Searc...	Sched...	Model
0	4	0.01	3	0.9809523809523809	0.9666666666666667	1.0	0.3292753623188406	grid	FIFO	NeuralNetTorch\21403_00017
1	4	0.01	2	0.9714285714285714	0.9333333333333333	1.0	0.3292753623188406	grid	FIFO	NeuralNetTorch\21403_00013
2	3	0.01	4	0.9714285714285714	0.9333333333333333	1.0	0.3292753623188406	grid	FIFO	NeuralNetTorch\21403_00004
3	2	0.01	4	0.9619047619047619	0.9333333333333333	1.0	0.3292753623188406	grid	FIFO	NeuralNetTorch\21403_00001
4	4	0.01	2	0.9428571428571428	0.9333333333333333	1.0	0.3292753623188406	grid	FIFO	NeuralNetTorch\21403_00016
5	2	0.01	3	0.9238095238095239	0.9333333333333333	1.0	0.3292753623188406	grid	FIFO	NeuralNetTorch\21403_00015
6	4	0.01	3	0.9142857142857143	0.9	1.0	0.3292753623188406	grid	FIFO	NeuralNetTorch\21403_00003
7	2	0.01	3	0.9142857142857143	0.9	1.0	0.3292753623188406	grid	FIFO	NeuralNetTorch\21403_00005
8	3	0.01	2	0.8857142857142857	0.8666666666666667	0.9333333333333333	0.3292753623188406	grid	FIFO	NeuralNetTorch\21403_00002
9	2	0.01	3	0.7142857142857143	0.7333333333333333	0.8	0.3292753623188406	grid	FIFO	NeuralNetTorch\21403_00009
10	2	0.0001	2	0.6	0.6666666666666666	0.6	0.3292753623188406	grid	FIFO	NeuralNetTorch\21403_00014
11	3	0.0001	2	0.5904761904761905	0.6333333333333333	0.5333333333333333	0.3292753623188406	grid	FIFO	NeuralNetTorch\21403_00010
12	3	0.0001	4	0.580952380952381	0.6333333333333333	0.4666666666666667	0.3292753623188406	grid	FIFO	NeuralNetTorch\21403_00012
13	4	0.0001	4	0.5619047619047619	0.5666666666666667	0.4666666666666667	0.3292753623188406	grid	FIFO	NeuralNetTorch\21403_00006
14	2	0.0001	4	0.5523809523809524	0.5666666666666667	0.4666666666666667	0.3292753623188406	grid	FIFO	NeuralNetTorch\21403_00011
15	4	0.0001	3	0.5333333333333333	0.5	0.4666666666666667	0.3292753623188406	grid	FIFO	NeuralNetTorch\21403_00007
16	4	0.0001	4	0.4380952380952381	0.43333333333333335	0.4	0.3292753623188406	grid	FIFO	NeuralNetTorch\21403_00008
17	2	0.0001	2	0.4	0.43333333333333335	0.4	0.3292753623188406	grid	FIFO	NeuralNetTorch\21403_00000

18 rows x 10 cols 25 per page Page 1 of 1

(with replacement – closest to grid search)

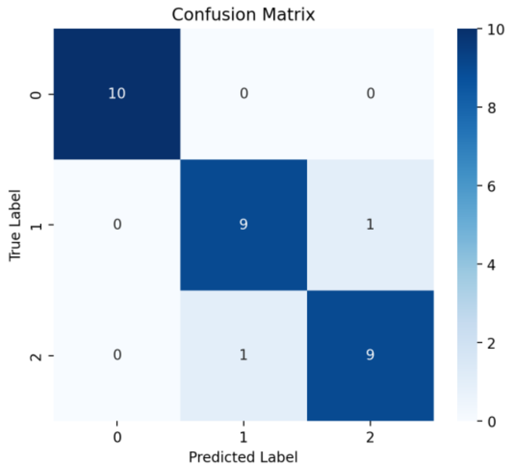


Bayes Optimization:

Bayesian optimization is a probabilistic model-based optimization technique that uses the posterior distribution of the loss function to select the most promising hyperparameters to evaluate in the next iteration.

5

	# Batch Size	# Learning Rate	# Epochs	# Train Accuracy	# Test Accuracy	# Validation Accuracy	# F1 Score	Search Strategy	Sched...	Model
0	4	0.006827674900436265	3	0.9714285714285714	0.9333333333333333	1.0	0.3292753623188406	bayes	FIFO	NeuralNetTorch\58a2bca0
1	3	0.008440544061688083	4	0.9714285714285714	0.9333333333333333	1.0	0.3292753623188406	bayes	FIFO	NeuralNetTorch\ad471457
2	2	0.009400030041468716	3	0.9523809523809523	0.9	1.0	0.3292753623188406	bayes	FIFO	NeuralNetTorch\b47f8f9e
3	3	0.004101488492199611	2	0.819047619047619	0.7333333333333333	0.8666666666666667	0.3292753623188406	bayes	FIFO	NeuralNetTorch\98ad48d0
4	2	0.0001	2	0.4380952380952381	0.43333333333333335	0.4	0.3292753623188406	bayes	FIFO	NeuralNetTorch\38e5c39c

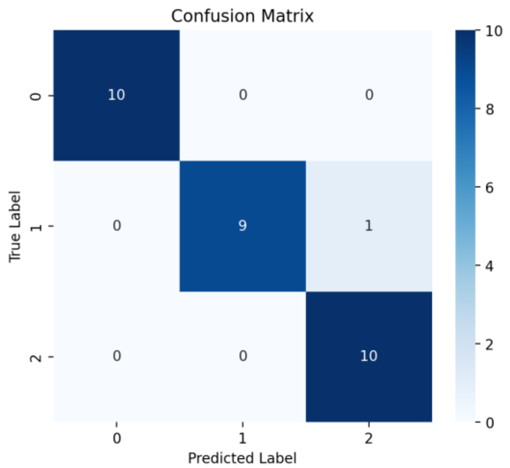


Bayes Optimization and hyperband:

Bayes with Hyperband essentially refers to a hybrid approach where the exploration strategy of Bayesian Optimization is combined with the efficient resource allocation mechanism of the Hyperband algorithm, and this combined method is often called ASHA (Asynchronous Successive Halving Algorithm); meaning, when you use Bayesian optimization to guide the search within the framework of Hyperband's parallel exploration and early stopping, you are effectively using ASHA.

5

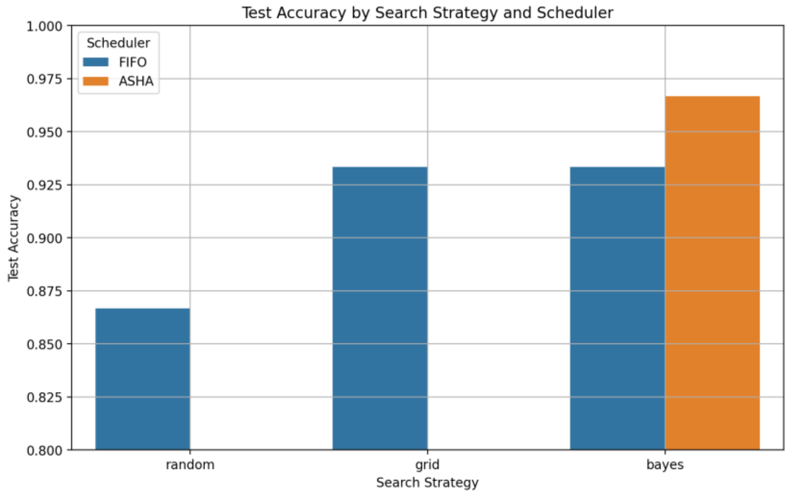
	# Batch Size	# Learning Rate	# Epochs	# Train Accuracy	# Test Accuracy	# Validation Accuracy	# F1 Score	Search Strategy	Sche...	Model
0	2	0.00967035452755208	3	0.9619047619047619	0.9666666666666667	1.0	0.3292753623188406	bayes_hyperband	ASHA	NeuralNetTorch\2a0048b3
1	4	0.008037556294293868	4	0.9333333333333333	0.8333333333333334	0.9333333333333333	0.3292753623188406	bayes_hyperband	ASHA	NeuralNetTorch\875bb9fa
2	3	0.001363854353460738	4	0.780952380952381	0.7	0.8666666666666667	0.3292753623188406	bayes_hyperband	ASHA	NeuralNetTorch\96751250
3	3	0.0016762927498813619	2	0.7428571428571429	0.6666666666666666	0.7333333333333333	0.3292753623188406	bayes_hyperband	ASHA	NeuralNetTorch\a37759a3
4	2	0.0001	2	0.4380952380952381	0.43333333333333335	0.4	0.3292753623188406	bayes_hyperband	ASHA	NeuralNetTorch\970d5f7b

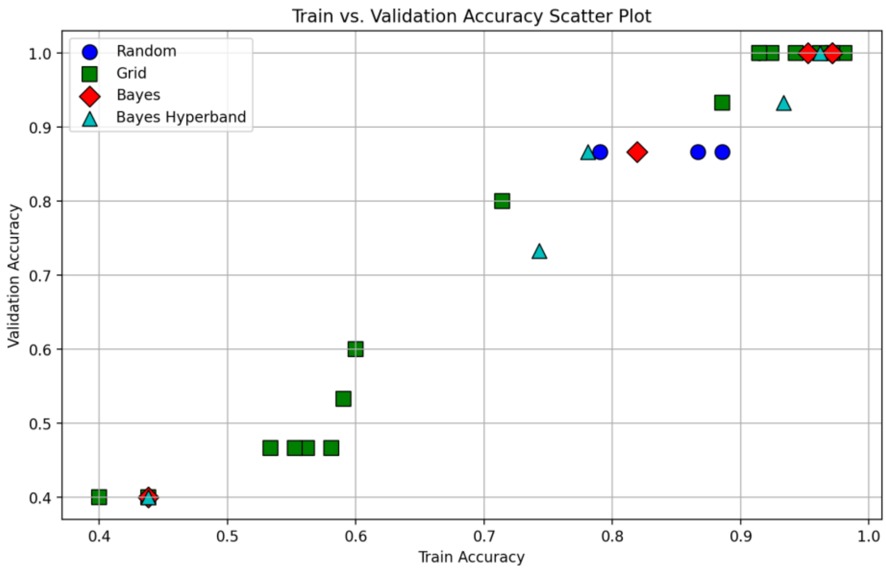


Results table Overall:

	# Batch Size	# Learning Rate	# Epochs	# Test Accuracy	# Validation Accuracy	# F1 Score	Search Strat...	Scheduler	# Train Accuracy
0	3	0.0036524217928226746	3	0.8666666666666667	1.0	0.8653198653198653	random	FIFO	0.9142857142857143
1	4	0.01	3	0.9333333333333333	1.0	0.9326599326599326	grid	FIFO	0.9809523809523809
2	4	0.006827674900436265	3	0.9333333333333333	1.0	0.9333333333333333	bayes	FIFO	0.9714285714285714
3	2	0.00967035452755208	3	0.9666666666666667	1.0	0.9665831244778613	bayes	ASHA	0.9619047619047619

4 rows x 9 cols 10 per page Page 1 of 1





Relationship Between Hyperparameters and Performance

Epochs vs. Performance (Direct Relationship)

More epochs generally allow the model to learn better, reducing loss and improving accuracy & F1 score.

Since all trials have only 3 epochs, we can't see a strong effect, but usually, more epochs the better performance (up to a limit).Overfitting only happens after a large number of epochs (around 20 or more)

Batch Size vs. Performance (Inverse Relationship)

Smaller batch sizes (e.g., 2, 3) tend to perform better (higher test accuracy & F1).

Larger batch sizes (e.g., 4) can lead to faster convergence but may not generalize as well.

Learning Rate vs. Performance (Optimal Range Effect)

Too high (0.01, Row 1): Can converge quickly but risk overshooting the optimal solution.

Too low (0.0036, Row 0): May learn too slowly and not reach optimal performance in given epochs.

Balanced (0.0097, Row 3): Achieves the best accuracy, meaning there's an optimal learning rate range.

Conclusion :

Best Approach: The Bayesian search strategy with ASHA (Row 3) performs the best.

It finds the best hyperparameters efficiently, leading to highest test accuracy (96.67%) and best F1 score (0.9666).

Grid Search (Row 1) is okay but can be slow since it tests all possible values systematically.

Random Search (Row 0) is the least effective because it selects values randomly without learning from past results.

Overall: Bayesian search is smarter—it picks better values by learning from past tests, making it more efficient.