Combined Structured Notes

Post 1 Notes

Post ID: 1 - Heart Stroke Prediction Model

Project Summary

- Developed a heart stroke prediction model using PyTorch and Logistic Regression.
- Addressed a highly imbalanced dataset using the SMOTE (Synthetic Minority Over-sampling Technique) method.
- Model shows average performance.
- Project focused on applying machine learning techniques to critical health prediction.

Technologies Used

- PyTorch
- Logistic Regression
- SMOTE

Challenges

- Highly imbalanced dataset with a vast difference between 'no stroke' and 'stroke' instances.

Scene 1: Code Editor

- Dark-themed code editor displaying Python code.
- Code appears to be a class definition related to training a machine learning model.
- Bottom lines of code are partially cut-off.
- No audio.

Extracted Code

```
python
device = torch.cuda if torch.cuda.is_available() else 'cpu'
except RuntimeError:
print("Can't be converted to the specified device so converted to (device)")

def set_loader(self, train_loader, val_loader=None):
self.train_loader = train_loader
self.val_loader = val_loader

def set_tensorboard(self, name, runs):
self.tensorboardWriter = f"{folder}/{runs}/{name}"

def make_train_fn(self):
```

...[cut-off]...

```
def perform_train_fn(x, y):
self.model.train() #Enabling to training mode...
y_hat = self.model(x)
loss = self.loss_fn(y_hat, y)
loss.backward()
self.optimizer.step()
self.optimizer.zero_grad()
return loss.item()

def perform_train_fn

def make_val_fn(self):
def perform_val_fn(x, y):
self.model.eval()
y_hat = self.model(x)
loss = self.loss_fn(y_hat, y)
return loss.item()

def mini_batch(self, validation=False):
if validation:
dataloader = self.val_loader
else:
dataloader = self.train_loader

mini_batch_losses = []
for x_batch, y_batch in dataloader:
x_batch = x_batch.to(self.device)
y_batch = y_batch.to(self.device)
mini_batch_loss = self.mini_batch_loss_fn(x_batch, y_batch)
mini_batch_losses.append(mini_batch_loss)
loss = np.mean(mini_batch_losses)
return loss

def train(self, n_epochs):
for epoch in range(n_epochs):
loss = self.mini_batch(validation=False)
self.train_losses.append(loss)
with torch.no_grad():
val_loss = self.mini_batch(validation=True)
val_val_losses.append(val_loss)

if self.writer:
scalars = {"training Loss":loss}
if val_loss is not None:
scalars.update({"Validation_val_loss":val_loss})
self.writer.add_scalars(main_tag="loss",tag_scalar_dict=scalars,global_step=epoch)


if self.writer:
self.writer.flush()

def save_checkpoint(self, file_name):
check_point = {"model_state_dict":self.model.state_dict(),
"total_epoch": epoch,
"optimizer_state_dict":self.optimizer.state_dict(),
"train_loss": self.train_losses,
"val_loss": self.val_losses}
```

#[cut-off]

Hashtags

- #PyTorch
- #LogisticRegression
- #MachineLearning
- #DataScience
- #Healthcare
- #StrokePrediction
- #SMOTE
- #DataImbalance
- #DeepLearning
- #AI

Other Notes

- No diagrams, graphs, charts, tables, equations, audio, or subtitles are present.
- Only object visible is a computer screen displaying code.
- Setting appears to be a digital environment. No people visible.
- Last few lines of code are partially cut off.