Πρόβλεψη Κατανάλωσης Νοικοκυριού

Για το πρώτο κομμάτι του διαγωνισμού χρειάστηκε μέσω του συνόλου των δεδομένων που μας προμηθεύτηκε, να προβλέψουμε τη κατανάλωση κάθε νοικοκυριού και τα ποσοστά φτώχειας, ανάλογα με κάποια χαρακτηριστικά που δόθηκαν μέσω ερευνών.

Καθαρισμός Δεδομένων

Αρχικά, αφαίρεσα τα χαρακτηριστικά `***survey_id***` και `***hhid***` που αφορούσαν μοναδικά αναγνωριστικά έρευνας και νοικοκυριού αντοίστιχα, διότι δεν παρείχαν χρήσιμη πληροφορία για τα μοντέλα μας.

```
x_train = household_train.drop(columns=['hhid', 'survey_id'])
x_test = household_test.drop(columns=['hhid', 'survey_id'])

y_train = target_train.drop(columns=['survey_id','hhid'])
```

Στη συνέχεια, χρειάστηκε να αντιστοιχίσω όλες τις τιμές κειμένου των κατηγορικών χαρακτηριστικών, με αντίστοιχες ακέραιες τιμές όπως δόθηκαν από το αρχείο **feature_value_descriptions.csv**.

```
consumed600,Sugar (white and brown),0,No
consumed600,Sugar (white and brown),1,Yes
consumed700,Egg,0,No
consumed700,Egg,1,Yes
consumed800,Beef and other red meats,0,No
consumed800,Beef and other red meats,1,Yes
consumed900,Chicken meat and other poultry,0,No
consumed900,Chicken meat and other poultry,1,Yes
dweltyp,Types of Dwelling,1,Detached house
dweltyp,Types of Dwelling,2,Multi-family house
dweltyp,Types of Dwelling,3,Separate apartment
dweltyp,Types of Dwelling,4,Communal apartment
dweltyp,Types of Dwelling,5,Room in a larger dwelling
dweltyp,Types of Dwelling,6,Several buildings connected
dweltyp,Types of Dwelling,7,Several separate buildings
dweltyp,Types of Dwelling,8,Improvised housing unit
dweltyp,Types of Dwelling,99,Other
educ_max,Highest education level in household,0,Never attended
educ_max,Highest education level in household,1,Incomplete Primary Education
```

Επίσης, απομόνωσα την στήλη `***weight***`, για να την χρησιμοποιήσω κατά την διαδικασία της εκπαίδευσης των μοντέλων μηχανικής μάθησης.

Τέλος, επειδή δεν προμηθεύτηκαν δεδομένα ελέγχου, χώρισα τα δεδομένα εκπαίδευσης με ποσοστό 70-30.

Επιλογή Μοντέλων

Για το συγκεκριμένο πρόβλημα παλινδρόμησης επιλέχθηκαν τα εξής μοντέλα:

1. LinearRegressor
2. RandomForestRegressor
3. XGBRegressor
4. Neural Network with 3-layers

Διάλεξα την συγκεκριμένη σειρά μοντέλων, ώστε να υπάρχει σύγκριση τιμών από το πιο απλό μοντέλο μηχανικής μάθησης ως το πιο πολύπλοκο XGBRegressor, και τέλος νευρωνικό δίκτυο όπου αυξάνεται αρκετά η πολυπλοκότητα.

ΑΠΟΤΕΛΕΣΜΑΤΑ

Ακολουθεί πίνακας αποτελεσμάτων των μοντέλων μηχανικής μάθησης, με βάση το Root Mean Squared Error, Mean Absolute Error και Mean Absolute Percentage.

	LinearRegressor	**RandomForestRegressor**	**XGBRegressor**
RMSE	7.189	7.008	6.050
MAE	4.039	4.014	3.410
MAPE	0.433	0.411	0.318

Αφού παρατηρήθηκε ότι ο XGBRegressor, με τις default παραμέτρους, είχε το μικρότερο ποσοστό λάθους σε σχέση με τα υπόλοιπα μοντέλα, έκανα fine-tuning του μοντέλου χρησιμοποιόντας την βιλιοθήκη *Optuna*.

```python
def objective(trial):
    n_estimators = trial.suggest_int("n_estimators", 50, 500)
    learning_rate = trial.suggest_float("learning_rate", 1e-4, 0.3, log=True)
    gamma = trial.suggest_float("gamma", 1e-3, 5, log=True)
    booster = trial.suggest_categorical("booster", ['gbtree','gblinear','dart'])
    max_depth = trial.suggest_int("max_depth", 2, 20)

    xgbReg = XGBRegressor(objective='reg:squarederror',
                          booster=booster,
                          n_estimators=n_estimators,
                          learning_rate=learning_rate,
                          gamma=gamma,
                          max_depth=max_depth,
                          device=device)

    xgbReg.fit(fx_train, y_train, sample_weight=train_weights)

    ypred = xgbReg.predict(fx_val)
    mae_score_test = mae(y_val, ypred, test_weights)

    return mae_score_test


study = optuna.create_study(direction='minimize')
study.optimize(objective, n_trials=100)

print(f"Best parameters: {study.best_params}")
print(f"Best MAE: {study.best_value}")
```

Οι καλύτερες τιμές των παραμέτρων που βρέθηκαν ήταν οι εξής:

Best parameters: {'n_estimators': 316, 'learning_rate': 0.023087045809130017, 'gamma': 2.4911737970405934, 'booster': 'dart', 'max_depth': 9}

```python
tuned_xgbReg = XGBRegressor(objective='reg:squarederror',
                            booster='dart',
                            n_estimators=316,
                            learning_rate=0.023087045809130017,
                            gamma=2.4911737970405934,
                            max_depth=9,
                            device=device)

tuned_xgbReg.fit(fx_train, y_train, sample_weight=train_weights)
```

	XGBRegressor
RMSE	5.899
MAE	3.306
MAPE	0.309

Νευρωνικό Δίκτυο

Τέλος, εκπαιδεύτηκε νευρωνικό δίκτυο τριών επιπέδων για 91 εποχές, και κατάφερε να πετύχει MAPE = ~0.124. Παρόλα αυτά, όταν χρησιμοποιήθηκε για την πρόβλεψη των τελικών δεδομένων και υποβλήθηκε για εξέταση, έλαβε μικρότερη βαθμολογία από τον XGBRegressor.

```
class NeuralNet(nn.Module):
    def __init__(self, input_dim, output_dim):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(input_dim, 128),
            nn.BatchNorm1d(128),
            nn.ReLU(),
            nn.Dropout(0.1),

            nn.Linear(128, 64),
            nn.BatchNorm1d(64),
            nn.ReLU(),
            nn.Dropout(0.1),

            nn.Linear(64, 32),
            nn.BatchNorm1d(32),
            nn.ReLU(),

            nn.Linear(32, output_dim)
        )

    def forward(self, x):
        return self.net(x)
```

```
model = NeuralNet(input_size, n_outputs).to(device)

criterion = nn.SmoothL1Loss(beta=1.0)
optimizer = optim.Adam(
    model.parameters(),
    lr=3e-4,
    weight_decay=1e-5
)

scheduler = optim.lr_scheduler.ReduceLROnPlateau(
    optimizer,
    mode='min',
    factor=0.5,
    patience=50,
    min_lr=1e-5
)
```

Πρόβλεψη Ποσοστών Φτιώχας

Για το δεύτερο κομμάτι του διαγωνισμού, χρειάστηκε να προβλέψουμε τα ποσοστά φτώχειας για τα τρία διαφορετικά group ερευνών. Επειδή για κάθε ~32000 δεδομένα εκπαίδευσης έπρεπε να εξάγουμε 19 ξεχωριστά ποσοστά φτώχιας, μας περιόριζε στην επιλογή μοντέλων μηχανικής μάθησης και η επιλογή του νευρωνικού δικτύου ήταν μονόδρομος.

Επεξεργασία Δεδομένων

Αρχικά, χρειάστηκε να χωρίσω τα δεδομένα εκπαίδευσης και ελέγχου σε έξι υποομάδες ανάλογα με το *survey_id* , διότι κάθε υποομάδα των 30000 περίπου δεδομένων εκπαιδεύονταν για να παράξει τις 19 τελικές τιμές για τη πρόβλεψη φτώχιας.

Split train data based on survey_id

```
unique_traingroups = household_train['survey_id'].unique()
unique_traingroups
```

```
array([100000, 200000, 300000])
```

```
Xsurvey1 = household_train[household_train['survey_id'] ==  unique_traingroups[0]]
Xsurvey2 = household_train[household_train['survey_id'] ==  unique_traingroups[1]]
Xsurvey3 = household_train[household_train['survey_id'] ==  unique_traingroups[2]]

Ysurvey1 = target_train_rates[target_train_rates['survey_id'] == unique_traingroups[0]]
Ysurvey2 = target_train_rates[target_train_rates['survey_id'] == unique_traingroups[1]]
Ysurvey3 = target_train_rates[target_train_rates['survey_id'] == unique_traingroups[2]]

Xsurvey1.shape, Xsurvey2.shape, Xsurvey3.shape
```

```
((32188, 88), (34584, 88), (37462, 88))
```

Νευρωνικό Δίκτυο

Το νευρωνικό δίκτυο αποτελείται μόνο από δύο επίπεδα με λίγους κόμβους, διότι είχαμε μόνο 3 δεδομένα για έλεγχο (y_train) και ήταν πολύ εύκολο να υπερπροσαρμοστεί το νευρωνικό. Επίσης, για τον ίδιο λόγο τα δεδομένα χωρίστηκαν σε μικρότερα κομμάτια (chunks) των 128 δεδομένων και έπειτα τροφοδοτήθηκαν στο νευρωνικό.

```python
class SetModel(nn.Module):
    def __init__(self, input_dim, output_dim):
        super().__init__()
        self.encoder = nn.Sequential(
            nn.Linear(input_dim, 64),
            nn.ReLU(),
            nn.Linear(64, 32),
            nn.ReLU(),
        )
        self.head = nn.Linear(32*2, output_dim)

    def forward(self, x):
        h = self.encoder(x)
        h_mean = h.mean(dim=0)
        h_max = h.max(dim=0).values
        h_combined = torch.cat([h_mean, h_max]).unsqueeze(0)
        out = self.head(h_combined)
        return out.squeeze(0)
```

```python
model = SetModel(input_size, n_outputs)
criterion = nn.MSELoss()
optimizer = optim.Adam(
    model.parameters(),
    lr=1e-5,
    weight_decay=1e-5
)
```

```python
epochs = 201

for epoch in range(epochs):
    total_loss = 0.0
    for X_chunk, y_chunk in train_chunks:
        optimizer.zero_grad()

        pred = model(X_chunk)
        loss = criterion(pred, y_chunk)
        loss.backward()

        torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
        optimizer.step()
        total_loss += loss.item()
    if epoch % 20 == 0:
        print(f"Epoch {epoch} | Loss: {total_loss:.6f}")
```

```
Epoch 0 | Loss: 256.963931
Epoch 20 | Loss: 0.290657
Epoch 40 | Loss: 0.133592
Epoch 60 | Loss: 0.081992
Epoch 80 | Loss: 0.056061
Epoch 100 | Loss: 0.040542
Epoch 120 | Loss: 0.030626
Epoch 140 | Loss: 0.023956
Epoch 160 | Loss: 0.019114
Epoch 180 | Loss: 0.015570
Epoch 200 | Loss: 0.012952
```

ΚΑΤΑΤΑΞΗ

Μετά από 2 υποβολές κατατάχθηκα στην 83 θέση από τις 351.

#80	**oknaitik** 2w 4d ago · 12 submissions	5.409	2.587	
#81	**somnathab3** 3d 22h ago · 2 submissions	5.414	2.625	
#82	**kecsap** 2d 19h ago · 6 submissions	5.424	2.463	
#83	**PanagiotisPapastathis** 2h 7min ago · 2 submissions	5.450	2.313	Share your work?
#84	**xvii** 6h 10min ago · 15 submissions	5.457	2.613	
#85	**No Comeback Crew** 1d 22h ago · 15 submissions	5.458	2.389	
#86	**_0037** 3w 4d ago · 2 submissions	5.474	2.080	
#87	**tabumis** 4d 2h ago · 10 submissions	5.475	2.674	