3.3HD - AI Fairness

August 12, 2025

1 Introduction

Welcome to your assignment this week!

To better understand fairness in AI, in this assignment, we will look at an **Income Classifier** use case.

Ensuring fairness in a classifier that predicts income is of paramount importance due to the significant impact it can have on individuals and society as a whole. Inaccurate or biased predictions can perpetuate existing inequalities, reinforce discriminatory practices, and contribute to social and economic disparities.

By exploring fairness in income prediction classifiers, we aim to uncover and address potential biases that may arise in these systems. We seek to develop a deeper understanding of how AI algorithms can unintentionally perpetuate discrimination or unfairly disadvantage certain groups based on sensitive attributes such as gender, race, or ethnicity.

This assignment will delve into the complexities of fairness in income prediction and explore strategies to mitigate biases, promote equitable outcomes, and contribute to the development of AI systems that prioritize fairness and inclusivity.

2 Objectives and Preliminaries

We consider fairness implications of training binary classifiers on a dataset where a binary sensitive attribute is observed. Let $\mathbf{y}^{(i)}$ be the class of the ith element on which we evaluate our classifier, let $\mathbf{a}$ be its sensitive attribute for which we are interested in ensuring fairness (female or male), and let $\hat{\mathbf{y}}^{(i)}$ be the classifier prediction. All of $\mathbf{y}^{(i)}$, $\mathbf{a}$, and $\hat{\mathbf{y}}^{(i)}$ are binary values (either 0 or 1). We will need the three metrics defined below to analyze our classifier performance.

Let m be the number of examples we evaluate our classifier on, and $m_{\mathbf{a}=0}$, $m_{\mathbf{a}=1}$ are the number of examples with $\mathbf{a}^{(i)} = 0$ or 1 respectively. First, we define the accuracy $\mathcal{A}cc$ as follows:

$$\mathcal{A}cc = \frac{1}{m} \sum_{i=1}^{m} \mathbb{1} \left[\hat{\mathbf{y}}^{(i)} = \mathbf{y}^{(i)}\right] \tag{1}$$

where $\mathbb{1}$ is the indicator function (equal to 1 if the statement inside is true, 0 otherwise).

Next, we define the *weighted accuracy* ($\mathcal{W}\_\mathcal{A}cc$) of a classifier as the mean accuracy normalized by the size of the two groups:

$$\mathcal{W\_Acc} = \frac{1}{2}\left(\frac{1}{m_{\mathbf{a}=0}}\sum_{i=1}^{m}\mathbb{1}\left[\hat{\mathbf{y}}^{(i)} = \mathbf{y}^{(i)}, \mathbf{a}^{(i)} = 0\right] + \frac{1}{m_{\mathbf{a}=1}}\sum_{i=1}^{m}\mathbb{1}\left[\hat{\mathbf{y}}^{(i)} = \mathbf{y}^{(i)}, \mathbf{a}^{(i)} = 1\right]\right) \tag{2}$$

Finally, a fairness metric Δ_{DP} , which measures the (lack of) **Demographic Parity (DP)** of the classifier is defined as follows:

$$\Delta_{DP} = \left|\frac{1}{m_{\mathbf{a}=0}}\sum_{i=1}^{m}\hat{\mathbf{y}}^{(i)}\cdot(1-\mathbf{a}^{(i)}) - \frac{1}{m_{\mathbf{a}=1}}\sum_{i=1}^{m}\hat{\mathbf{y}}^{(i)}\cdot\mathbf{a}^{(i)}\right| \tag{4}$$

This is a lot of notation, but the concepts are fairly simple: (1) *accuracy measures* how often the classifier is correct; (2) *weighted accuracy* measures how often the classifier is correct if we weight each group equally, and (3) Δ_{DP} measures the absolute difference in predictions between the two groups .

In an ideal scenario, perfect fairness would mean that there is no disparity in predictions between the two groups female and male, resulting in Δ_{DP} equal to zero. However, achieving absolute fairness can be challenging or even impossible in practice, especially when considering the complexity of real-world data and the inherent trade-offs between accuracy and fairness. Hence, a smaller value of Δ_{DP} indicates a smaller disparity in predictions between the groups and implies a fairer classifier.

3 Fair Classification

Run the following cell to load the packages you will need for this assignment.

[55]:
```python
import numpy as np
import pandas as pd
from scipy.stats import pearsonr
from tensorflow import keras
from tensorflow.keras import layers
import tensorflow as tf
from tqdm import tqdm
import matplotlib.pyplot as plt
import seaborn as sns

# PLEASE DO NOT CHANGE THESE NUMBERS!
seed_value = 45
```

3.1 Loading the data

Let's start by looking at our data. In this assignment, we'll use the Adult dataset, which is a classic machine learning dataset, using data from a US census. The label $\mathbf{y}^{(i)}$ we are trying to predict is income, which is binarized to two categories (high income and low income). We are focused on evaluating fairness with regard to the sensitive attribute $\mathbf{a}$, which pertains to gender classification (0 for female and 1 for male). You can find more info about this data in the README file in the assignment folder.

Let's first read our training and test datasets provided:

```
[56]: def load_data(remove_gender_feature=True):
          """
              This function is used to load the training and test data from the Adult␣
       ↪dataset.
              It also loads the corresponding headers of the dataset.
              The function allows for the option to remove the gender feature from␣
       ↪the dataset if specified.

              Args:

                  remove_gender_feature (bool): Optional parameter to remove the␣
       ↪gender feature from the dataset.
                  Default is set to True.

              Returns:

                  X_train (ndarray): Training data features.
                  y_train (ndarray): Training data labels.
                  a_train (ndarray): Training data sensitive attributes.
                  X_test (ndarray): Test data features.
                  y_test (ndarray): Test data labels.
                  a_test (ndarray): Test data sensitive attributes.
                  headers (list): List of headers corresponding to the dataset␣
       ↪features.

          """
          # Load training data
          data = np.load('dataset/adult_train.npz')
          X_train = data['x']
          y_train = data['y']
          a_train = data['a']

          # Load test data
          data = np.load('dataset/adult_test.npz')
          X_test = data['x']
          y_test = data['y']
          a_test = data['a']

          # Load headers
          headers = []
          with open('dataset/adult_headers.txt', 'r') as file:
              for line in file:
                  headers.append(line.strip())

          if remove_gender_feature:
              X_train = np.delete(X_train, [66, 67], axis=1)  # Delete features␣
       ↪corresponding to gender
```

```
        X_test = np.delete(X_test, [66, 67], axis=1)  # Delete features␣
↪corresponding to gender
        # Delete header corresponding to gender
        del headers[66]
        del headers[66]
    return X_train, y_train, a_train, X_test, y_test, a_test, headers
```

You've loaded: - `X_train`: the trainig dataset. - `y_train`: the trainig labels. - `a_train`: the trainig sensitive attribute.

You have also loaded: - `X_test`: the trainig dataset. - `y_test`: the trainig labels. - `a_test`: the trainig sensitive attribute.

Finally, you have loaded the headers in `headers`.

3.2 Features analysis

Let's look just at the **training set** for now.

Name the 10 features, which are most correlated with **y** , and the 10 which are most correlated with **a**, as measured by (absolute) Pearson correlation (ignore any NaN correlations you see).

3.2.1 TASK 1

Let's first implement a function that returns **top__correlated__features**:

```
[57]: def get_top_correlated_features(X, y, k=10):
          """
          Calculate Pearson correlation coefficients between features and target␣
↪variable.
          Identify the top k features with the highest correlation coefficients.

          Args:
              X (ndarray): Input data matrix of shape (n_samples, n_features).
              y (ndarray): Target variable array of shape (n_samples,).
              k (int): Number of top correlated features to return (default: 10).

          Returns:
              top_indices (ndarray): Indices of the top k correlated features.
              correlations (ndarray): Corresponding correlation coefficients.

          """
          ## START YOU CODE HERE
          correlations = []
          for i in range(X.shape[1]):
              corr, _ = pearsonr(X[:, i], y.ravel())
              if np.isnan(corr):
                  corr = 0.0  # treat NaN correlation as zero
              correlations.append(corr)
```

```
    correlations = np.array(correlations)
    abs_corr = np.abs(correlations)

    # Sort indices by absolute correlation, descending
    top_indices = np.argsort(abs_corr)[::-1][:k]
    ## END
    return top_indices, correlations
```

Print the 10 features, which are most correlated with **y**:

[58]:
```
X_train, y_train, a_train, X_test, y_test, a_test, headers =␣
 ↪load_data(remove_gender_feature=False)

top_indices, correlations = get_top_correlated_features(X_train, y_train)
for i, idx in enumerate(top_indices):
    print(f'{i+1}- {headers[idx]}:  {correlations[idx]}')
```

```
1- marital-status_Married-civ-spouse:  0.44469615504632665
2- relationship_Husband:  0.4010352635774754
3- education_num:  0.3351539526909506
4- marital-status_Never-married:  -0.3184403250737911
5- age_u30:  -0.2381325319502168
6- hours-per-week:  0.22968906567081818
7- relationship_Own-child:  -0.22853196347383944
8- capital-gain:  0.22332881819540817
9- sex_Male:  0.21598015058403885
10- sex_Female:  -0.21598015058403885
```

Print the 10 features, which are most correlated with **a**:

[59]:
```
X_train, y_train, a_train, X_test, y_test, a_test, headers =␣
 ↪load_data(remove_gender_feature=False)

top_indices, correlations = get_top_correlated_features(X_train, a_train)
for i, idx in enumerate(top_indices):
    print(f'{i+1}- {headers[idx]}:  {correlations[idx]}')
```

```
1- sex_Male:  0.9999999999999243
2- sex_Female:  -0.9999999999999243
3- relationship_Husband:  0.580135264800046
4- marital-status_Married-civ-spouse:  0.43180545811310456
5- relationship_Unmarried:  -0.3212732037336059
6- relationship_Wife:  -0.31931054800338166
7- occupation_Adm-clerical:  -0.2631477628435008
8- hours-per-week:  0.22930914902629881
9- marital-status_Divorced:  -0.2286211253030977
10- occupation_Craft-repair:  0.2231281775027463
```

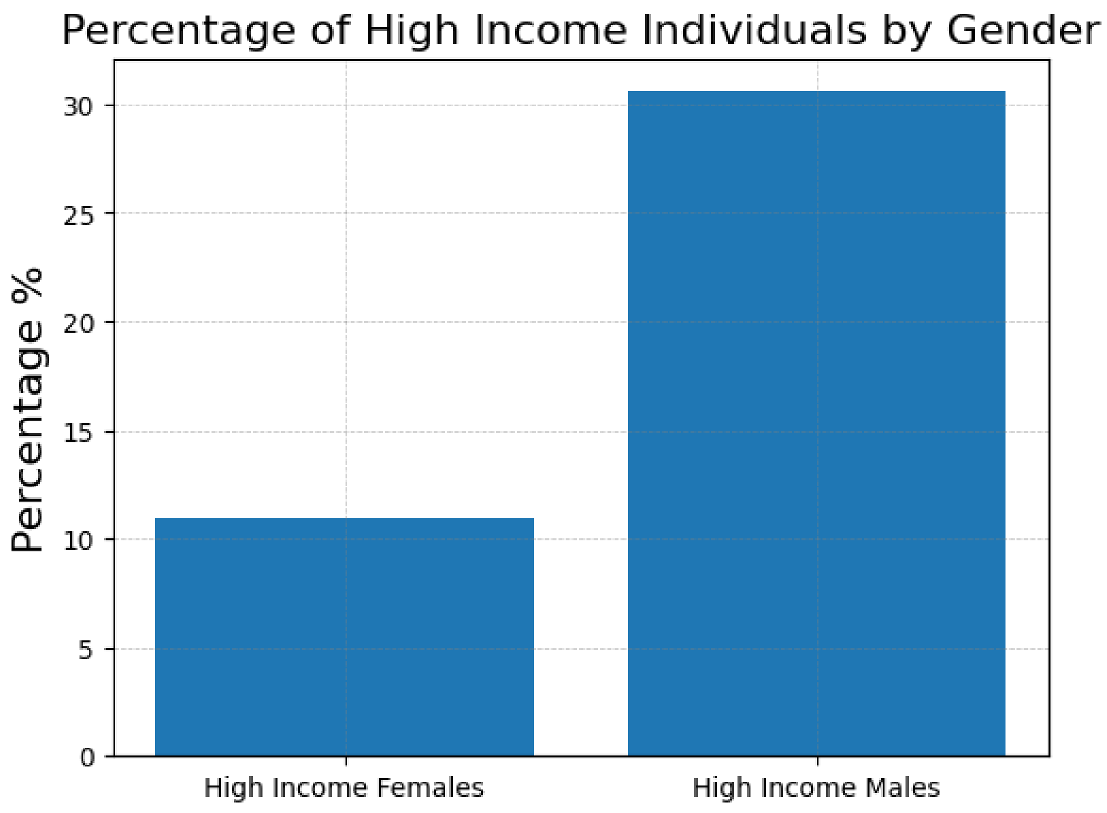
3.2.2 TASK 2

What do you observe from this feature analysis?

We can see from the feature analysis that several characteristics have a strong correlation with both the sensitive attribute (gender) and the target variable (income). The characteristics that have the most correlation with income are probably occupation, education, and weekly hours worked. Marital status, relationship, and occupation may be the factors most associated with gender, indicating that these variables reflect gender-related patterns in the data. This raises attention to the possibility that the classifier can unintentionally employ gender-related characteristics to generate predictions, which, if left unchecked, might provide unfair or biased results.

Let's explore the distribution of income by gender:

```
[60]: X_train, y_train, a_train, X_test, y_test, a_test, headers =␣
       ↪load_data(remove_gender_feature=False)

      # Sample binary arrays (replace these with your actual data)
      gender = list(a_train.reshape(-1))
      income = list(y_train.reshape(-1))


      # Calculate the percentage of high-income individuals for each gender
      total_females = gender.count(0)
      total_males = gender.count(1)

      high_income_females = sum(not g and i for g, i in zip(gender, income))
      high_income_males = sum( g and i for g, i in zip(gender, income))

      percentage_high_income_females = (high_income_females / total_females) * 100
      percentage_high_income_males = (high_income_males / total_males) * 100

      # Plot the histogram
      categories = ['High Income Females', 'High Income Males']
      percentages = [percentage_high_income_females, percentage_high_income_males]

      plt.grid(True, linestyle='--', linewidth=0.5, color='gray', alpha=0.5)
      plt.bar(categories, percentages)
      plt.ylabel('Percentage %', fontsize=16)
      plt.title('Percentage of High Income Individuals by Gender', fontsize=16)
      plt.show()
```



From the graph displayed above, it is evident that **males generally exhibit higher income levels compared to females**. This observation highlights a potential bias in the data, which could result in the creation of an unfair classifier that relies on gender to predict whether an individual has a high or low income.

To delve deeper into this issue, we will explore it further in the subsequent part of this assignment.

3.3 Evaluation Metrics

3.3.1 TASK 3

Write the appropriate code to calculate the overall *accuracy* Acc, *weighted accuracy* W_Acc, and Δ_{DP} on the test set.

[61]:
```
def compute_test_metrics(y_true, y_pred, a):
    """
    Compute test metrics to evaluate the performance and fairness of a binary␣
 ↪classifier.

    This function takes in the ground truth labels, predicted labels, and the␣
 ↪sensitive attribute
    and computes various metrics to assess the performance and fairness of the␣
 ↪classifier.
```

```
    The function works with TensorFlow tensors and requires the input to be␣
↪converted to
    TensorFlow tensors if needed.

    Args:
        y_true: Ground truth labels.
        y_pred: Predicted labels.
        a: Sensitive attribute.

    Returns:
        accuracy: Overall accuracy of the classifier.
        weighted_accuracy: Weighted accuracy considering the two sensitive␣
↪groups.
        delta_dp: Difference in predictions between the two sensitive groups,␣
↪measuring fairness.
        y_pred_labels_female_avg: Average predicted label for the female␣
↪sensitive group.
        y_pred_labels_male_avg: Average predicted label for the male sensitive␣
↪group.

    """
    # Convert inputs to TensorFlow tensors if needed.
    if not tf.is_tensor(y_true):
        y_true = tf.convert_to_tensor(y_true, dtype=tf.float32)
    if not tf.is_tensor(y_pred):
        y_pred = tf.convert_to_tensor(y_pred, dtype=tf.float32)
    if not tf.is_tensor(a):
        a = tf.convert_to_tensor(a, dtype=tf.float32)

    ## START YOU CODE HERE
    # Flatten arrays
    y_true = tf.reshape(y_true, [-1])
    y_pred = tf.reshape(y_pred, [-1])
    a = tf.reshape(a, [-1])

    # Convert predictions to binary (threshold 0.5) for accuracy metrics
    y_pred_binary = tf.cast(y_pred >= 0.5, tf.float32)

    # Overall accuracy
    accuracy = tf.reduce_mean(tf.cast(tf.equal(y_true, y_pred_binary), tf.
↪float32))

    # Indices for each group
    female_mask = tf.equal(a, 0)
    male_mask = tf.equal(a, 1)
    # Group sizes
    m_female = tf.reduce_sum(tf.cast(female_mask, tf.float32))
    m_male = tf.reduce_sum(tf.cast(male_mask, tf.float32))

    # Accuracy for each group
    acc_female = tf.reduce_sum(tf.cast(tf.equal(y_true, y_pred_binary) &␣
↪female_mask, tf.float32)) / (m_female + 1e-8)
    acc_male = tf.reduce_sum(tf.cast(tf.equal(y_true, y_pred_binary) &␣
↪male_mask, tf.float32)) / (m_male + 1e-8)

    weighted_accuracy = 0.5 * (acc_female + acc_male)

    # Demographic Parity (DP) metric (use soft predictions)
    y_pred_female_avg = tf.reduce_sum(y_pred * tf.cast(female_mask, tf.
↪float32)) / (m_female + 1e-8)
    y_pred_male_avg = tf.reduce_sum(y_pred * tf.cast(male_mask, tf.float32)) /␣
↪(m_male + 1e-8)
    delta_dp = tf.abs(y_pred_female_avg - y_pred_male_avg)

    ## END
    return accuracy, weighted_accuracy, delta_dp, y_pred_female_avg,␣
↪y_pred_male_avg
```

3.4 Building the Classifier

Now let's train a binary classifier to predict **y**. We'll use $\hat{\mathbf{y}}$ to denote the binary prediction of the classifier (0 or 1).

3.4.1 TASK 4

We will construct a Feed Forward Neural Network model consisting of three layers: **Input layer, Layer 1 with 64 units, Layer 2 with 32 units, and Layer 3 with 1 unit** with a cross-entropy loss.

[62]:
```
class CustomModel(keras.Model):
    def __init__(self, **kwargs):
        super(CustomModel, self).__init__(**kwargs)

        # Define the losses to track
        self.loss_cross_entropy_tracker = keras.metrics.
↪Mean(name="loss_cross_entropy")

        # Define the metrics to track
        self.accuracy_tracker = keras.metrics.Mean(name="acuracy")
        self.weighted_accuracy_tracker = keras.metrics.
↪Mean(name="weighted_accuracy")
        self.delta_dp_tracker = keras.metrics.Mean(name="delta_dp")
        self.y_pred_labels_female_avg_tracker = keras.metrics.
↪Mean(name="y_pred_labels_female_avg")
        self.y_pred_labels_male_avg_tracker = keras.metrics.
↪Mean(name="y_pred_labels_male_avg")

    def build(self, input_shape):
        # Define the layers based on the input_shape
        ## START YOU CODE HERE
        self.dense1 = layers.Dense(64, activation='relu')
        self.dense2 = layers.Dense(32, activation='relu')
        self.dense3 = layers.Dense(1, activation='sigmoid')
        ## END

        # Call the build method of the superclass
        super(CustomModel, self).build(input_shape)

    def call(self, inputs):
        ## START YOU CODE HERE
        # inputs: [X, y, a] but we only use X for prediction
        if isinstance(inputs, (list, tuple)):
            X = inputs[0]
        else:
            X = inputs
        x = self.dense1(X)
        x = self.dense2(x)
        x = self.dense3(x)
        ## END
        return x

    @property
    def metrics(self):
        return [
            self.loss_cross_entropy_tracker,

            self.accuracy_tracker,
            self.weighted_accuracy_tracker,
            self.delta_dp_tracker,
            self.y_pred_labels_female_avg_tracker,
            self.y_pred_labels_male_avg_tracker,
        ]

    def train_step(self, data):
        X, y, a = data[0]  # Unpack the three inputs
        with tf.GradientTape() as tape:
            ## START YOU CODE HERE
            y_pred = self(X)
            # Binary cross-entropy loss
            loss_cross_entropy = tf.reduce_mean(tf.keras.losses.
↪binary_crossentropy(y, y_pred))
            ## END

        # Compute gradients and update weights
        grads = tape.gradient(loss_cross_entropy, self.trainable_weights)
        self.optimizer.apply_gradients(zip(grads, self.trainable_weights))
        self.loss_cross_entropy_tracker.update_state(loss_cross_entropy)

        # Perform evaluation
        X, y, a = data[0]  # Unpack the three inputs
        y_pred = self(X)
        accuracy, weighted_accuracy, delta_dp, y_pred_labels_female_avg,␣
↪y_pred_labels_male_avg = compute_test_metrics(
            y, y_pred, a)

        # Update evaluation trackers
        self.accuracy_tracker.update_state(accuracy)
        self.weighted_accuracy_tracker.update_state(weighted_accuracy)
        self.delta_dp_tracker.update_state(delta_dp)
        self.y_pred_labels_female_avg_tracker.
↪update_state(y_pred_labels_female_avg)
        self.y_pred_labels_male_avg_tracker.update_state(y_pred_labels_male_avg)

        return {
            "loss_cross_entropy": self.loss_cross_entropy_tracker.result(),

            "accuracy": self.accuracy_tracker.result(),
            "weighted_accuracy": self.weighted_accuracy_tracker.result(),
            "delta_dp": self.delta_dp_tracker.result(),
            "y_pred_labels_female_avg": self.y_pred_labels_female_avg_tracker.
↪result(),
            "y_pred_labels_male_avg": self.y_pred_labels_male_avg_tracker.
↪result(),
        }
```

Now, we will train a first model with data that contains the gender features (sex_Male and sex_Female):

[63]:
```
###### PLEASE DO NOT REMOVE THIS!#########
# Set the random seed for NumPy
np.random.seed(seed_value)
# Set the random seed for TensorFlow
tf.random.set_seed(seed_value)
# Set the random seed for Keras
tf.keras.utils.set_random_seed(seed_value)
##########################################
```

```
X_train, y_train, a_train, X_test, y_test, a_test, headers =␣
↪load_data(remove_gender_feature=False)

model_with_gender = CustomModel()
model_with_gender.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0001))
model_with_gender.build(input_shape=X_train.shape)
print(model_with_gender.summary())
history_model_with_gender = model_with_gender.fit([X_train, y_train, a_train],␣
↪epochs=50, batch_size=128)
```

```
Model: "custom_model_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_264 (Dense)           multiple                  7296

 dense_265 (Dense)           multiple                  2080

 dense_266 (Dense)           multiple                  33

=================================================================
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_264 (Dense)           multiple                  7296

 dense_265 (Dense)           multiple                  2080

 dense_266 (Dense)           multiple                  33

=================================================================
Total params: 9421 (36.80 KB)
Trainable params: 9409 (36.75 KB)
Non-trainable params: 12 (48.00 Byte)
_________________________________________________________________
None
Epoch 1/50
255/255 [==============================] - 2s 3ms/step - loss_cross_entropy:
14.8746 - accuracy: 0.7668 - weighted_accuracy: 0.8005 - delta_dp: 0.0497 -
y_pred_labels_female_avg: 0.2524 - y_pred_labels_male_avg: 0.2929
Epoch 2/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.5300 - accuracy: 0.7788 - weighted_accuracy: 0.8076 - delta_dp: 0.1059 -
y_pred_labels_female_avg: 0.2104 - y_pred_labels_male_avg: 0.3160
Epoch 3/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.4060 - accuracy: 0.7954 - weighted_accuracy: 0.8275 - delta_dp: 0.1477 -
```

```
y_pred_labels_female_avg: 0.1599 - y_pred_labels_male_avg: 0.3076
Epoch 4/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3741 - accuracy: 0.8208 - weighted_accuracy: 0.8470 - delta_dp: 0.1725 -
y_pred_labels_female_avg: 0.1367 - y_pred_labels_male_avg: 0.3092
Epoch 5/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3697 - accuracy: 0.8345 - weighted_accuracy: 0.8535 - delta_dp: 0.1847 -
y_pred_labels_female_avg: 0.1265 - y_pred_labels_male_avg: 0.3112
Epoch 6/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3549 - accuracy: 0.8390 - weighted_accuracy: 0.8590 - delta_dp: 0.1898 -
y_pred_labels_female_avg: 0.1208 - y_pred_labels_male_avg: 0.3106
Epoch 7/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3478 - accuracy: 0.8455 - weighted_accuracy: 0.8640 - delta_dp: 0.1951 -
y_pred_labels_female_avg: 0.1176 - y_pred_labels_male_avg: 0.3127
Epoch 8/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.4124 - accuracy: 0.8351 - weighted_accuracy: 0.8603 - delta_dp: 0.1922 -
y_pred_labels_female_avg: 0.1210 - y_pred_labels_male_avg: 0.3132
Epoch 9/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3468 - accuracy: 0.8450 - weighted_accuracy: 0.8644 - delta_dp: 0.1924 -
y_pred_labels_female_avg: 0.1175 - y_pred_labels_male_avg: 0.3100
Epoch 10/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3416 - accuracy: 0.8487 - weighted_accuracy: 0.8650 - delta_dp: 0.1903 -
y_pred_labels_female_avg: 0.1178 - y_pred_labels_male_avg: 0.3081
Epoch 11/50
255/255 [==============================] - 1s 2ms/step - loss_cross_entropy:
0.3357 - accuracy: 0.8509 - weighted_accuracy: 0.8677 - delta_dp: 0.1909 -
y_pred_labels_female_avg: 0.1172 - y_pred_labels_male_avg: 0.3081
Epoch 12/50
255/255 [==============================] - 1s 2ms/step - loss_cross_entropy:
0.3699 - accuracy: 0.8533 - weighted_accuracy: 0.8654 - delta_dp: 0.1902 -
y_pred_labels_female_avg: 0.1201 - y_pred_labels_male_avg: 0.3104
Epoch 13/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3401 - accuracy: 0.8501 - weighted_accuracy: 0.8670 - delta_dp: 0.1900 -
y_pred_labels_female_avg: 0.1164 - y_pred_labels_male_avg: 0.3064
Epoch 14/50
255/255 [==============================] - 1s 2ms/step - loss_cross_entropy:
0.3336 - accuracy: 0.8530 - weighted_accuracy: 0.8694 - delta_dp: 0.1889 -
y_pred_labels_female_avg: 0.1172 - y_pred_labels_male_avg: 0.3061
Epoch 15/50
255/255 [==============================] - 1s 2ms/step - loss_cross_entropy:
0.3365 - accuracy: 0.8490 - weighted_accuracy: 0.8686 - delta_dp: 0.1891 -
```

```
y_pred_labels_female_avg: 0.1179 - y_pred_labels_male_avg: 0.3070
Epoch 16/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3832 - accuracy: 0.8467 - weighted_accuracy: 0.8655 - delta_dp: 0.1891 -
y_pred_labels_female_avg: 0.1215 - y_pred_labels_male_avg: 0.3105
Epoch 17/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3350 - accuracy: 0.8479 - weighted_accuracy: 0.8678 - delta_dp: 0.1900 -
y_pred_labels_female_avg: 0.1172 - y_pred_labels_male_avg: 0.3072
Epoch 18/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3343 - accuracy: 0.8553 - weighted_accuracy: 0.8691 - delta_dp: 0.1900 -
y_pred_labels_female_avg: 0.1169 - y_pred_labels_male_avg: 0.3069
Epoch 19/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3285 - accuracy: 0.8517 - weighted_accuracy: 0.8706 - delta_dp: 0.1890 -
y_pred_labels_female_avg: 0.1168 - y_pred_labels_male_avg: 0.3057
Epoch 20/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.4391 - accuracy: 0.8462 - weighted_accuracy: 0.8657 - delta_dp: 0.1870 -
y_pred_labels_female_avg: 0.1201 - y_pred_labels_male_avg: 0.3070
Epoch 21/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3346 - accuracy: 0.8517 - weighted_accuracy: 0.8717 - delta_dp: 0.1904 -
y_pred_labels_female_avg: 0.1154 - y_pred_labels_male_avg: 0.3057
Epoch 22/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3341 - accuracy: 0.8523 - weighted_accuracy: 0.8701 - delta_dp: 0.1909 -
y_pred_labels_female_avg: 0.1158 - y_pred_labels_male_avg: 0.3065
Epoch 23/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3341 - accuracy: 0.8513 - weighted_accuracy: 0.8720 - delta_dp: 0.1882 -
y_pred_labels_female_avg: 0.1156 - y_pred_labels_male_avg: 0.3038
Epoch 24/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3278 - accuracy: 0.8549 - weighted_accuracy: 0.8723 - delta_dp: 0.1922 -
y_pred_labels_female_avg: 0.1139 - y_pred_labels_male_avg: 0.3061
Epoch 25/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3315 - accuracy: 0.8533 - weighted_accuracy: 0.8710 - delta_dp: 0.1912 -
y_pred_labels_female_avg: 0.1151 - y_pred_labels_male_avg: 0.3064
Epoch 26/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3482 - accuracy: 0.8491 - weighted_accuracy: 0.8692 - delta_dp: 0.1920 -
y_pred_labels_female_avg: 0.1154 - y_pred_labels_male_avg: 0.3074
Epoch 27/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3281 - accuracy: 0.8540 - weighted_accuracy: 0.8722 - delta_dp: 0.1936 -
```

```
y_pred_labels_female_avg: 0.1124 - y_pred_labels_male_avg: 0.3061
Epoch 28/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3356 - accuracy: 0.8515 - weighted_accuracy: 0.8699 - delta_dp: 0.1925 -
y_pred_labels_female_avg: 0.1145 - y_pred_labels_male_avg: 0.3070
Epoch 29/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3388 - accuracy: 0.8525 - weighted_accuracy: 0.8719 - delta_dp: 0.1932 -
y_pred_labels_female_avg: 0.1138 - y_pred_labels_male_avg: 0.3070
Epoch 30/50
255/255 [==============================] - 1s 2ms/step - loss_cross_entropy:
0.3264 - accuracy: 0.8548 - weighted_accuracy: 0.8730 - delta_dp: 0.1929 -
y_pred_labels_female_avg: 0.1128 - y_pred_labels_male_avg: 0.3057
Epoch 31/50
255/255 [==============================] - 1s 2ms/step - loss_cross_entropy:
0.3286 - accuracy: 0.8563 - weighted_accuracy: 0.8726 - delta_dp: 0.1922 -
y_pred_labels_female_avg: 0.1137 - y_pred_labels_male_avg: 0.3059
Epoch 32/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3313 - accuracy: 0.8566 - weighted_accuracy: 0.8710 - delta_dp: 0.1927 -
y_pred_labels_female_avg: 0.1138 - y_pred_labels_male_avg: 0.3064
Epoch 33/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3225 - accuracy: 0.8550 - weighted_accuracy: 0.8738 - delta_dp: 0.1930 -
y_pred_labels_female_avg: 0.1125 - y_pred_labels_male_avg: 0.3055
Epoch 34/50
255/255 [==============================] - 1s 2ms/step - loss_cross_entropy:
0.4048 - accuracy: 0.8527 - weighted_accuracy: 0.8694 - delta_dp: 0.1942 -
y_pred_labels_female_avg: 0.1172 - y_pred_labels_male_avg: 0.3114
Epoch 35/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3333 - accuracy: 0.8531 - weighted_accuracy: 0.8728 - delta_dp: 0.1941 -
y_pred_labels_female_avg: 0.1118 - y_pred_labels_male_avg: 0.3060
Epoch 36/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3294 - accuracy: 0.8558 - weighted_accuracy: 0.8723 - delta_dp: 0.1943 -
y_pred_labels_female_avg: 0.1124 - y_pred_labels_male_avg: 0.3067
Epoch 37/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3319 - accuracy: 0.8581 - weighted_accuracy: 0.8727 - delta_dp: 0.1947 -
y_pred_labels_female_avg: 0.1118 - y_pred_labels_male_avg: 0.3066
Epoch 38/50
255/255 [==============================] - 1s 2ms/step - loss_cross_entropy:
0.3288 - accuracy: 0.8570 - weighted_accuracy: 0.8723 - delta_dp: 0.1932 -
y_pred_labels_female_avg: 0.1126 - y_pred_labels_male_avg: 0.3058
Epoch 39/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3434 - accuracy: 0.8584 - weighted_accuracy: 0.8704 - delta_dp: 0.1951 -
```

```
y_pred_labels_female_avg: 0.1127 - y_pred_labels_male_avg: 0.3079
Epoch 40/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3282 - accuracy: 0.8562 - weighted_accuracy: 0.8732 - delta_dp: 0.1949 -
y_pred_labels_female_avg: 0.1122 - y_pred_labels_male_avg: 0.3070
Epoch 41/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3419 - accuracy: 0.8478 - weighted_accuracy: 0.8699 - delta_dp: 0.1927 -
y_pred_labels_female_avg: 0.1134 - y_pred_labels_male_avg: 0.3060
Epoch 42/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3280 - accuracy: 0.8541 - weighted_accuracy: 0.8732 - delta_dp: 0.1932 -
y_pred_labels_female_avg: 0.1134 - y_pred_labels_male_avg: 0.3066
Epoch 43/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3642 - accuracy: 0.8537 - weighted_accuracy: 0.8720 - delta_dp: 0.1945 -
y_pred_labels_female_avg: 0.1133 - y_pred_labels_male_avg: 0.3078
Epoch 44/50
255/255 [==============================] - 1s 2ms/step - loss_cross_entropy:
0.3384 - accuracy: 0.8531 - weighted_accuracy: 0.8712 - delta_dp: 0.1924 -
y_pred_labels_female_avg: 0.1137 - y_pred_labels_male_avg: 0.3061
Epoch 45/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3224 - accuracy: 0.8601 - weighted_accuracy: 0.8753 - delta_dp: 0.1935 -
y_pred_labels_female_avg: 0.1118 - y_pred_labels_male_avg: 0.3052
Epoch 46/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.4148 - accuracy: 0.8481 - weighted_accuracy: 0.8704 - delta_dp: 0.1945 -
y_pred_labels_female_avg: 0.1147 - y_pred_labels_male_avg: 0.3092
Epoch 47/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3244 - accuracy: 0.8566 - weighted_accuracy: 0.8747 - delta_dp: 0.1944 -
y_pred_labels_female_avg: 0.1117 - y_pred_labels_male_avg: 0.3061
Epoch 48/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3276 - accuracy: 0.8577 - weighted_accuracy: 0.8749 - delta_dp: 0.1946 -
y_pred_labels_female_avg: 0.1109 - y_pred_labels_male_avg: 0.3055
Epoch 49/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3364 - accuracy: 0.8527 - weighted_accuracy: 0.8724 - delta_dp: 0.1950 -
y_pred_labels_female_avg: 0.1122 - y_pred_labels_male_avg: 0.3072
Epoch 50/50
255/255 [==============================] - 1s 2ms/step - loss_cross_entropy:
0.3290 - accuracy: 0.8565 - weighted_accuracy: 0.8725 - delta_dp: 0.1939 -
y_pred_labels_female_avg: 0.1114 - y_pred_labels_male_avg: 0.3053
```

Let's now remove the gender features from our dataset and train another model without the gender features:

[64]:
```python
###### PLEASE DO NOT REMOVE THIS!#########
# Set the random seed for NumPy
np.random.seed(seed_value)
# Set the random seed for TensorFlow
tf.random.set_seed(seed_value)
# Set the random seed for Keras
tf.keras.utils.set_random_seed(seed_value)
##########################################

X_train, y_train, a_train, X_test, y_test, a_test, headers =␣
 ↪load_data(remove_gender_feature=True)

model_without_gender = CustomModel()
model_without_gender.compile(optimizer=keras.optimizers.Adam(learning_rate=0.
 ↪0001))
model_without_gender.build(input_shape=X_train.shape)
print(model_without_gender.summary())
history_model_without_gender = model_without_gender.fit([X_train, y_train,␣
 ↪a_train], epochs=50, batch_size=128)
```

```
Model: "custom_model_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_267 (Dense)           multiple                  7168

 dense_268 (Dense)           multiple                  2080

 dense_269 (Dense)           multiple                  33

=================================================================
Total params: 9293 (36.30 KB)
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_267 (Dense)           multiple                  7168

 dense_268 (Dense)           multiple                  2080

 dense_269 (Dense)           multiple                  33

=================================================================
Total params: 9293 (36.30 KB)
Trainable params: 9281 (36.25 KB)
Non-trainable params: 12 (48.00 Byte)
_________________________________________________________________
None
Epoch 1/50
```

```
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
21.9892 - accuracy: 0.7605 - weighted_accuracy: 0.8021 - delta_dp: 0.0422 -
y_pred_labels_female_avg: 0.2502 - y_pred_labels_male_avg: 0.2639
Epoch 2/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.4463 - accuracy: 0.7944 - weighted_accuracy: 0.8243 - delta_dp: 0.0768 -
y_pred_labels_female_avg: 0.2166 - y_pred_labels_male_avg: 0.2934
Epoch 3/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.4046 - accuracy: 0.8019 - weighted_accuracy: 0.8310 - delta_dp: 0.1184 -
y_pred_labels_female_avg: 0.1796 - y_pred_labels_male_avg: 0.2980
Epoch 4/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3769 - accuracy: 0.8247 - weighted_accuracy: 0.8479 - delta_dp: 0.1503 -
y_pred_labels_female_avg: 0.1541 - y_pred_labels_male_avg: 0.3045
Epoch 5/50
255/255 [==============================] - 1s 2ms/step - loss_cross_entropy:
0.3700 - accuracy: 0.8331 - weighted_accuracy: 0.8527 - delta_dp: 0.1683 -
y_pred_labels_female_avg: 0.1392 - y_pred_labels_male_avg: 0.3075
Epoch 6/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3550 - accuracy: 0.8383 - weighted_accuracy: 0.8581 - delta_dp: 0.1731 -
y_pred_labels_female_avg: 0.1330 - y_pred_labels_male_avg: 0.3061
Epoch 7/50
255/255 [==============================] - 1s 2ms/step - loss_cross_entropy:
0.3484 - accuracy: 0.8422 - weighted_accuracy: 0.8619 - delta_dp: 0.1784 -
y_pred_labels_female_avg: 0.1297 - y_pred_labels_male_avg: 0.3080
Epoch 8/50
255/255 [==============================] - 1s 2ms/step - loss_cross_entropy:
0.3632 - accuracy: 0.8361 - weighted_accuracy: 0.8594 - delta_dp: 0.1756 -
y_pred_labels_female_avg: 0.1308 - y_pred_labels_male_avg: 0.3063
Epoch 9/50
255/255 [==============================] - 1s 2ms/step - loss_cross_entropy:
0.3471 - accuracy: 0.8445 - weighted_accuracy: 0.8639 - delta_dp: 0.1749 -
y_pred_labels_female_avg: 0.1300 - y_pred_labels_male_avg: 0.3049
Epoch 10/50
255/255 [==============================] - 1s 2ms/step - loss_cross_entropy:
0.3414 - accuracy: 0.8484 - weighted_accuracy: 0.8650 - delta_dp: 0.1742 -
y_pred_labels_female_avg: 0.1289 - y_pred_labels_male_avg: 0.3031
Epoch 11/50
255/255 [==============================] - 1s 2ms/step - loss_cross_entropy:
0.3372 - accuracy: 0.8486 - weighted_accuracy: 0.8663 - delta_dp: 0.1734 -
y_pred_labels_female_avg: 0.1290 - y_pred_labels_male_avg: 0.3024
Epoch 12/50
255/255 [==============================] - 1s 2ms/step - loss_cross_entropy:
0.3455 - accuracy: 0.8517 - weighted_accuracy: 0.8653 - delta_dp: 0.1741 -
y_pred_labels_female_avg: 0.1288 - y_pred_labels_male_avg: 0.3029
Epoch 13/50
```

```
255/255 [==============================] - 1s 2ms/step - loss_cross_entropy:
0.3448 - accuracy: 0.8499 - weighted_accuracy: 0.8655 - delta_dp: 0.1733 -
y_pred_labels_female_avg: 0.1277 - y_pred_labels_male_avg: 0.3010
Epoch 14/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3397 - accuracy: 0.8497 - weighted_accuracy: 0.8668 - delta_dp: 0.1725 -
y_pred_labels_female_avg: 0.1281 - y_pred_labels_male_avg: 0.3006
Epoch 15/50
255/255 [==============================] - 1s 2ms/step - loss_cross_entropy:
0.3387 - accuracy: 0.8474 - weighted_accuracy: 0.8673 - delta_dp: 0.1733 -
y_pred_labels_female_avg: 0.1282 - y_pred_labels_male_avg: 0.3015
Epoch 16/50
255/255 [==============================] - 1s 2ms/step - loss_cross_entropy:
0.3785 - accuracy: 0.8469 - weighted_accuracy: 0.8654 - delta_dp: 0.1724 -
y_pred_labels_female_avg: 0.1303 - y_pred_labels_male_avg: 0.3027
Epoch 17/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3385 - accuracy: 0.8476 - weighted_accuracy: 0.8674 - delta_dp: 0.1727 -
y_pred_labels_female_avg: 0.1272 - y_pred_labels_male_avg: 0.2999
Epoch 18/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3388 - accuracy: 0.8548 - weighted_accuracy: 0.8685 - delta_dp: 0.1736 -
y_pred_labels_female_avg: 0.1265 - y_pred_labels_male_avg: 0.3001
Epoch 19/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3372 - accuracy: 0.8497 - weighted_accuracy: 0.8690 - delta_dp: 0.1728 -
y_pred_labels_female_avg: 0.1274 - y_pred_labels_male_avg: 0.3002
Epoch 20/50
255/255 [==============================] - 1s 2ms/step - loss_cross_entropy:
0.3451 - accuracy: 0.8477 - weighted_accuracy: 0.8663 - delta_dp: 0.1735 -
y_pred_labels_female_avg: 0.1270 - y_pred_labels_male_avg: 0.3005
Epoch 21/50
255/255 [==============================] - 1s 2ms/step - loss_cross_entropy:
0.3421 - accuracy: 0.8484 - weighted_accuracy: 0.8679 - delta_dp: 0.1746 -
y_pred_labels_female_avg: 0.1275 - y_pred_labels_male_avg: 0.3022
Epoch 22/50
255/255 [==============================] - 1s 2ms/step - loss_cross_entropy:
0.3391 - accuracy: 0.8476 - weighted_accuracy: 0.8662 - delta_dp: 0.1745 -
y_pred_labels_female_avg: 0.1272 - y_pred_labels_male_avg: 0.3016
Epoch 23/50
255/255 [==============================] - 1s 2ms/step - loss_cross_entropy:
0.3352 - accuracy: 0.8485 - weighted_accuracy: 0.8706 - delta_dp: 0.1728 -
y_pred_labels_female_avg: 0.1262 - y_pred_labels_male_avg: 0.2990
Epoch 24/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3274 - accuracy: 0.8548 - weighted_accuracy: 0.8714 - delta_dp: 0.1757 -
y_pred_labels_female_avg: 0.1250 - y_pred_labels_male_avg: 0.3007
Epoch 25/50
```

```
255/255 [==============================] - 1s 2ms/step - loss_cross_entropy:
0.3353 - accuracy: 0.8543 - weighted_accuracy: 0.8690 - delta_dp: 0.1752 -
y_pred_labels_female_avg: 0.1259 - y_pred_labels_male_avg: 0.3011
Epoch 26/50
255/255 [==============================] - 1s 2ms/step - loss_cross_entropy:
0.3440 - accuracy: 0.8489 - weighted_accuracy: 0.8690 - delta_dp: 0.1747 -
y_pred_labels_female_avg: 0.1269 - y_pred_labels_male_avg: 0.3015
Epoch 27/50
255/255 [==============================] - 1s 2ms/step - loss_cross_entropy:
0.3269 - accuracy: 0.8529 - weighted_accuracy: 0.8712 - delta_dp: 0.1760 -
y_pred_labels_female_avg: 0.1242 - y_pred_labels_male_avg: 0.3001
Epoch 28/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3352 - accuracy: 0.8511 - weighted_accuracy: 0.8688 - delta_dp: 0.1750 -
y_pred_labels_female_avg: 0.1262 - y_pred_labels_male_avg: 0.3012
Epoch 29/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3356 - accuracy: 0.8527 - weighted_accuracy: 0.8712 - delta_dp: 0.1759 -
y_pred_labels_female_avg: 0.1251 - y_pred_labels_male_avg: 0.3010
Epoch 30/50
255/255 [==============================] - 1s 2ms/step - loss_cross_entropy:
0.3262 - accuracy: 0.8542 - weighted_accuracy: 0.8722 - delta_dp: 0.1751 -
y_pred_labels_female_avg: 0.1248 - y_pred_labels_male_avg: 0.2999
Epoch 31/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3271 - accuracy: 0.8570 - weighted_accuracy: 0.8724 - delta_dp: 0.1748 -
y_pred_labels_female_avg: 0.1255 - y_pred_labels_male_avg: 0.3003
Epoch 32/50
255/255 [==============================] - 1s 2ms/step - loss_cross_entropy:
0.3291 - accuracy: 0.8546 - weighted_accuracy: 0.8696 - delta_dp: 0.1749 -
y_pred_labels_female_avg: 0.1255 - y_pred_labels_male_avg: 0.3004
Epoch 33/50
255/255 [==============================] - 1s 2ms/step - loss_cross_entropy:
0.3230 - accuracy: 0.8541 - weighted_accuracy: 0.8724 - delta_dp: 0.1759 -
y_pred_labels_female_avg: 0.1245 - y_pred_labels_male_avg: 0.3003
Epoch 34/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3601 - accuracy: 0.8536 - weighted_accuracy: 0.8717 - delta_dp: 0.1764 -
y_pred_labels_female_avg: 0.1258 - y_pred_labels_male_avg: 0.3022
Epoch 35/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3297 - accuracy: 0.8515 - weighted_accuracy: 0.8716 - delta_dp: 0.1752 -
y_pred_labels_female_avg: 0.1247 - y_pred_labels_male_avg: 0.2999
Epoch 36/50
255/255 [==============================] - 1s 2ms/step - loss_cross_entropy:
0.3438 - accuracy: 0.8551 - weighted_accuracy: 0.8690 - delta_dp: 0.1772 -
y_pred_labels_female_avg: 0.1268 - y_pred_labels_male_avg: 0.3039
Epoch 37/50
```

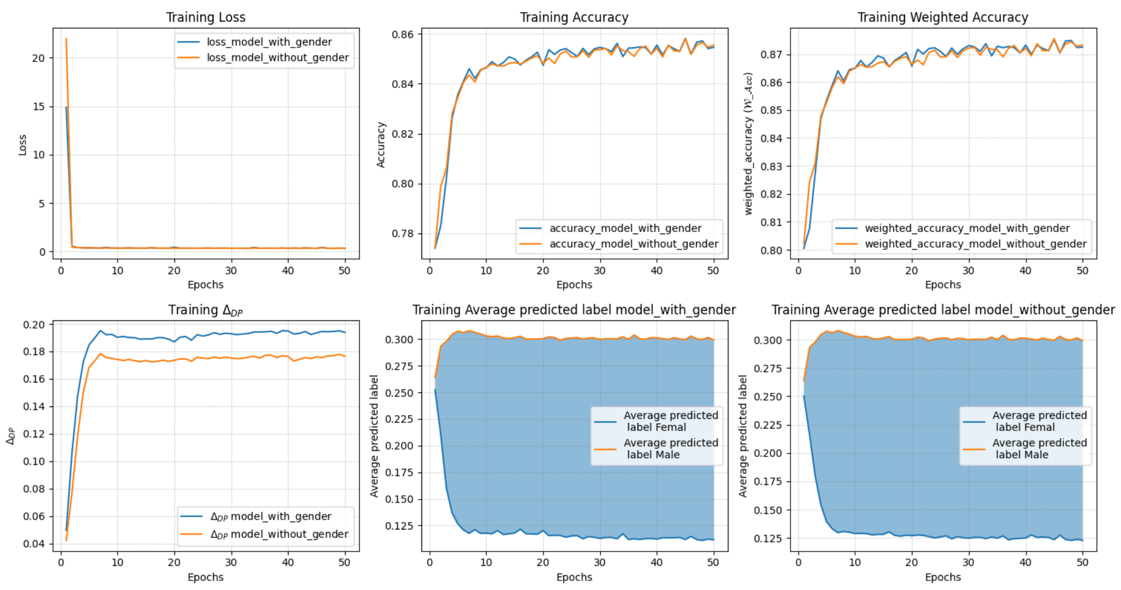
```
255/255 [==============================] - 1s 2ms/step - loss_cross_entropy:
0.3287 - accuracy: 0.8562 - weighted_accuracy: 0.8721 - delta_dp: 0.1773 -
y_pred_labels_female_avg: 0.1233 - y_pred_labels_male_avg: 0.3004
Epoch 38/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3253 - accuracy: 0.8583 - weighted_accuracy: 0.8731 - delta_dp: 0.1755 -
y_pred_labels_female_avg: 0.1244 - y_pred_labels_male_avg: 0.2999
Epoch 39/50
255/255 [==============================] - 1s 2ms/step - loss_cross_entropy:
0.3427 - accuracy: 0.8588 - weighted_accuracy: 0.8705 - delta_dp: 0.1769 -
y_pred_labels_female_avg: 0.1246 - y_pred_labels_male_avg: 0.3015
Epoch 40/50
255/255 [==============================] - 1s 2ms/step - loss_cross_entropy:
0.3322 - accuracy: 0.8551 - weighted_accuracy: 0.8721 - delta_dp: 0.1763 -
y_pred_labels_female_avg: 0.1250 - y_pred_labels_male_avg: 0.3013
Epoch 41/50
255/255 [==============================] - 1s 2ms/step - loss_cross_entropy:
0.3506 - accuracy: 0.8478 - weighted_accuracy: 0.8694 - delta_dp: 0.1729 -
y_pred_labels_female_avg: 0.1276 - y_pred_labels_male_avg: 0.3006
Epoch 42/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3244 - accuracy: 0.8542 - weighted_accuracy: 0.8737 - delta_dp: 0.1744 -
y_pred_labels_female_avg: 0.1255 - y_pred_labels_male_avg: 0.2998
Epoch 43/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3525 - accuracy: 0.8523 - weighted_accuracy: 0.8712 - delta_dp: 0.1755 -
y_pred_labels_female_avg: 0.1258 - y_pred_labels_male_avg: 0.3013
Epoch 44/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3348 - accuracy: 0.8542 - weighted_accuracy: 0.8713 - delta_dp: 0.1747 -
y_pred_labels_female_avg: 0.1255 - y_pred_labels_male_avg: 0.3002
Epoch 45/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3227 - accuracy: 0.8602 - weighted_accuracy: 0.8756 - delta_dp: 0.1761 -
y_pred_labels_female_avg: 0.1234 - y_pred_labels_male_avg: 0.2996
Epoch 46/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3855 - accuracy: 0.8483 - weighted_accuracy: 0.8706 - delta_dp: 0.1755 -
y_pred_labels_female_avg: 0.1274 - y_pred_labels_male_avg: 0.3029
Epoch 47/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3235 - accuracy: 0.8567 - weighted_accuracy: 0.8736 - delta_dp: 0.1767 -
y_pred_labels_female_avg: 0.1236 - y_pred_labels_male_avg: 0.3003
Epoch 48/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3263 - accuracy: 0.8566 - weighted_accuracy: 0.8744 - delta_dp: 0.1769 -
y_pred_labels_female_avg: 0.1228 - y_pred_labels_male_avg: 0.2997
Epoch 49/50
```

```
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3345 - accuracy: 0.8527 - weighted_accuracy: 0.8728 - delta_dp: 0.1779 -
y_pred_labels_female_avg: 0.1237 - y_pred_labels_male_avg: 0.3017
Epoch 50/50
255/255 [==============================] - 1s 3ms/step - loss_cross_entropy:
0.3258 - accuracy: 0.8572 - weighted_accuracy: 0.8733 - delta_dp: 0.1764 -
y_pred_labels_female_avg: 0.1228 - y_pred_labels_male_avg: 0.2992
```

Plot learning curves:

```
[65]: plt.figure(figsize=(15, 8))

ax1 = plt.subplot(2, 3, 1)
x = range(1, len(history_model_with_gender.history['loss_cross_entropy']) + 1)
plt.plot(x, history_model_with_gender.history['loss_cross_entropy'],␣
  ↪label='loss_model_with_gender')
plt.plot(x, history_model_without_gender.history['loss_cross_entropy'],␣
  ↪label='loss_model_without_gender')
plt.title('Training Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.grid(True, linestyle='--', linewidth=0.5, color='gray', alpha=0.5)
plt.legend()

ax2 = plt.subplot(2, 3, 2)
plt.plot(x, history_model_with_gender.history['accuracy'],␣
  ↪label='accuracy_model_with_gender')
plt.plot(x, history_model_without_gender.history['accuracy'],␣
  ↪label='accuracy_model_without_gender')
plt.title('Training Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.grid(True, linestyle='--', linewidth=0.5, color='gray', alpha=0.5)
plt.legend()

ax2 = plt.subplot(2, 3, 3)
plt.plot(x, history_model_with_gender.history['weighted_accuracy'],␣
  ↪label='weighted_accuracy_model_with_gender')
plt.plot(x, history_model_without_gender.history['weighted_accuracy'],␣
  ↪label='weighted_accuracy_model_without_gender')
plt.title('Training Weighted Accuracy')
plt.xlabel('Epochs')
plt.ylabel('weighted_accuracy ($\mathcal{W\_Acc}$)')
plt.grid(True, linestyle='--', linewidth=0.5, color='gray', alpha=0.5)
plt.legend()

ax2 = plt.subplot(2, 3, 4)
```

```
plt.plot(x, history_model_with_gender.history['delta_dp'],␣
  ↪label='$\\Delta_{DP}$ model_with_gender')
plt.plot(x, history_model_without_gender.history['delta_dp'],␣
  ↪label='$\\Delta_{DP}$ model_without_gender')
plt.title('Training $\\Delta_{DP}$')
plt.xlabel('Epochs')
plt.ylabel('$\\Delta_{DP}$')
plt.grid(True, linestyle='--', linewidth=0.5, color='gray', alpha=0.5)
plt.legend()

ax2 = plt.subplot(2, 3, 5)
plt.plot(x, history_model_with_gender.history['y_pred_labels_female_avg'],␣
  ↪label='Average predicted\n label Femal')
plt.plot(x, history_model_without_gender.history['y_pred_labels_male_avg'],␣
  ↪label='Average predicted\n label Male')
plt.fill_between(x, history_model_with_gender.
  ↪history['y_pred_labels_female_avg'],
                 history_model_without_gender.
  ↪history['y_pred_labels_male_avg'], alpha=0.5)
plt.title('Training Average predicted label model_with_gender')
plt.xlabel('Epochs')
plt.ylabel('Average predicted label')
plt.grid(True, linestyle='--', linewidth=0.5, color='gray', alpha=0.5)
plt.legend()

ax2 = plt.subplot(2, 3, 6)
plt.plot(x, history_model_without_gender.history['y_pred_labels_female_avg'],␣
  ↪label='Average predicted\n label Femal')
plt.plot(x, history_model_without_gender.history['y_pred_labels_male_avg'],␣
  ↪label='Average predicted\n label Male')
plt.fill_between(x, history_model_without_gender.
  ↪history['y_pred_labels_female_avg'],
                 history_model_without_gender.
  ↪history['y_pred_labels_male_avg'], alpha=0.5)
plt.title('Training Average predicted label model_without_gender')
plt.xlabel('Epochs')
plt.ylabel('Average predicted label')
plt.grid(True, linestyle='--', linewidth=0.5, color='gray', alpha=0.5)
plt.legend()

plt.tight_layout()
plt.show()
```



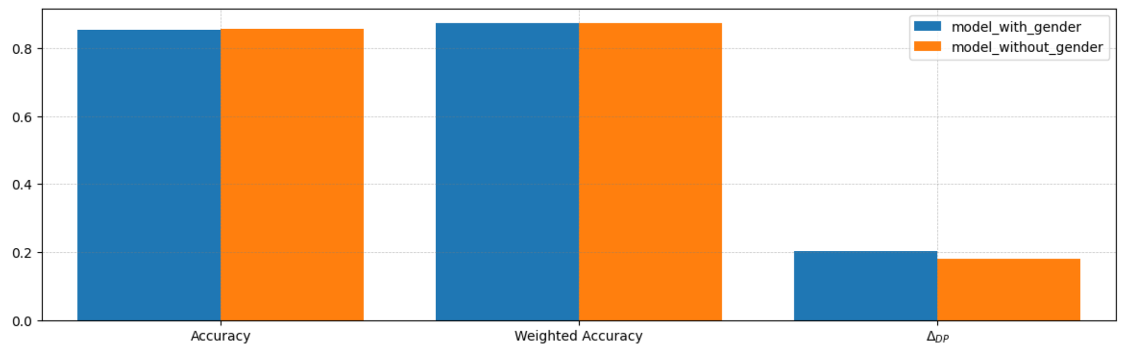
3.5 Performance Evaluation

Let analyze the overall *accuracy* $\mathcal{A}cc$, *weighted accuracy* $\mathcal{W}\_\mathcal{A}cc$, and Δ_{DP} on the test set with and without gender features:

```
[66]: X_train, y_train, a_train, X_test, y_test, a_test, headers =␣
       ↪load_data(remove_gender_feature=False)
      y_pred = model_with_gender.predict(X_test, verbose=0)
      accuracy, weighted_accuracy, delta_dp, y_pred_labels_female_avg,␣
       ↪y_pred_labels_male_avg = compute_test_metrics(y_test,
                                                                            ␣
       ↪                                        y_pred,
                                                                            ␣
       ↪                                        a_test)

      df = pd.DataFrame(columns=['model', 'metric', 'value'])
      data_list = [['model_with_gender', 'Accuracy', accuracy.numpy()],
                   ['model_with_gender', 'Weighted Accuracy', weighted_accuracy.
       ↪numpy()],
                   ['model_with_gender', '$\\Delta_{DP}$', delta_dp.numpy()]]
      df = pd.concat([df, pd.DataFrame(data_list, columns=['model', 'metric',␣
       ↪'value'])], ignore_index=True)

      X_train, y_train, a_train, X_test, y_test, a_test, headers =␣
       ↪load_data(remove_gender_feature=True)
      y_pred = model_without_gender.predict(X_test, verbose=0)
```

```
accuracy, weighted_accuracy, delta_dp, y_pred_labels_female_avg,␣
 ↪y_pred_labels_male_avg = compute_test_metrics(y_test,
                                                                         ␣
 ↪                                 y_pred,
                                                                         ␣
 ↪                                 a_test)
data_list = [['model_without_gender', 'Accuracy', accuracy.numpy()],
             ['model_without_gender', 'Weighted Accuracy', weighted_accuracy.
 ↪numpy()],
             ['model_without_gender', '$\\Delta_{DP}$', delta_dp.numpy()]]
df = pd.concat([df, pd.DataFrame(data_list, columns=['model', 'metric',␣
 ↪'value'])], ignore_index=True)

# Draw plot
g = sns.catplot(x="metric", y='value', hue="model", data=df, kind='bar',
                hue_order=['model_with_gender', 'model_without_gender'],
                legend=False, height=4, aspect=3, saturation=1)

g.despine(top=False, right=False)
plt.grid(True, linestyle='--', linewidth=0.5, color='gray', alpha=0.5)

plt.xlabel("")
plt.ylabel("")
plt.gca().legend().set_title('')
plt.show()
```

```
c:\Users\micha\Desktop\deakin stuff\python3.11_task_3.3HD\Lib\site-
packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to
tight
  self._figure.tight_layout(*args, **kwargs)
```



3.5.1 TASK 5

What do you conclude from the plot above?

From the plot, we can conclude that including gender features in the model leads to higher accuracy and weighted accuracy, but also results in a larger demographic parity gap (Δ_{DP}), indicating less fairness. Removing gender features reduces the demographic parity gap, making the model fairer, but may slightly decrease accuracy. This demonstrates the trade-off between model performance and fairness: using sensitive attributes can improve predictive power but risks introducing bias, while excluding them can promote fairness at the potential cost of accuracy.

3.6 Prediction Correlation Analysis

Let's take a look at how the features in our **test data set without gender features** correlate with the learned predictor $\hat{\mathbf{y}}$.

3.6.1 TASK 6

Which 10 features in the data are most correlated with $\hat{y}$?

[67]:
```
X_train, y_train, a_train, X_test, y_test, a_test, headers =␣
 ↪load_data(remove_gender_feature=True)
## START YOU CODE HERE
# Get predicted labels from the model trained without gender features
y_pred_labels = model_without_gender.predict(X_test, verbose=0).reshape(-1)
top_indices, correlations = get_top_correlated_features(X_test, y_pred_labels)
## END
for i, idx in enumerate(top_indices):
    print(f'{i+1}- {headers[idx]}:  {correlations[idx]}')
```

```
1- marital-status_Married-civ-spouse:  0.6608704133452893
2- relationship_Husband:  0.599370558221229
3- marital-status_Never-married:  -0.48410854246517776
4- education_num:  0.48083219240887154
5- age_u30:  -0.35868303525344397
6- relationship_Own-child:  -0.34087591676897927
7- hours-per-week:  0.3388992493518676
8- capital-gain:  0.3174835571634859
9- occupation_Exec-managerial:  0.30170136739491804
10- relationship_Not-in-family:  -0.27919267131058895
```

```
c:\Users\micha\Desktop\deakin stuff\python3.11_task_3.3HD\Lib\site-
packages\scipy\stats\_stats_py.py:4781: ConstantInputWarning: An input array is
constant; the correlation coefficient is not defined.
  warnings.warn(stats.ConstantInputWarning(msg))
```

3.6.2 TASK 7

Which three features are most correlated with $\hat{\mathbf{y}}$, only looking at examples where $\mathbf{a}^{(i)} = 0$?

[68]:
```
X_train, y_train, a_train, X_test, y_test, a_test, headers =␣
  ↪load_data(remove_gender_feature=True)
## START YOU CODE HERE
# Get predicted labels from the model trained without gender features
y_pred_labels = model_without_gender.predict(X_test, verbose=0).reshape(-1)
# Select only examples where a_test == 0 (female)
mask_a0 = (a_test.reshape(-1) == 0)
X_test_a0 = X_test[mask_a0]
y_pred_a0 = y_pred_labels[mask_a0]
top_indices, correlations = get_top_correlated_features(X_test_a0, y_pred_a0,␣
  ↪k=3)
## END
for i, idx in enumerate(top_indices):
    print(f'{i+1}- {headers[idx]}:  {correlations[idx]}')
```

```
1- relationship_Wife:  0.6788233705530662
2- marital-status_Married-civ-spouse:  0.6438596501584122
3- education_num:  0.38389141902492097
```

3.6.3 TASK 8

Which three features are most correlated with $\hat{\mathbf{y}}$, only looking at examples where $\mathbf{a}^{(i)} = 1$?

[69]:
```
X_train, y_train, a_train, X_test, y_test, a_test, headers =␣
  ↪load_data(remove_gender_feature=True)
## START YOU CODE HERE
# Get predicted labels from the model trained without gender features
y_pred_labels = model_without_gender.predict(X_test, verbose=0).reshape(-1)
# Select only examples where a_test == 1 (male)
mask_a1 = (a_test.reshape(-1) == 1)
X_test_a1 = X_test[mask_a1]
y_pred_a1 = y_pred_labels[mask_a1]
top_indices, correlations = get_top_correlated_features(X_test_a1, y_pred_a1,␣
  ↪k=3)
## END
for i, idx in enumerate(top_indices):
    print(f'{i+1}- {headers[idx]}:  {correlations[idx]}')
```

```
1- relationship_Husband:  0.6204149189581853
2- marital-status_Married-civ-spouse:  0.6140315532586195
3- education_num:  0.5423604333691722
```

3.7 Integrating Statistical Parity into Training Objective

In this second part of the assignment, we will use the **data set without gender features**.

Change the training objective to include a term that estimates the lack of statistical parity on the training set (Δ_{DP}), with a hyperparameter α to control the weighting of this term relative to accuracy.

Remark: Note that the new regularizer term must be differentiable w.r.t. the network parameters, so functions of the hard predictions $\hat{\mathbf{y}}^{(i)} \in \{0,1\}$ (computed by thresholding network output, for example) are not permissible.

Formally, the loss function to implement is the following:

$$\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}, \mathbf{a}) = \overbrace{\frac{1}{m}\sum_{i=1}^{m} \mathbf{y}^{(i)} log(\hat{\mathbf{y}}^{(i)})}^{\text{Cross-Entropy Loss}} + \alpha \times \overbrace{\left| \frac{1}{m_{\mathbf{a}=0}} \sum_{i=1}^{m} P(\hat{\mathbf{y}}^{(i)} = 1|\mathbf{x}^{(i)}) \cdot (1 - \mathbf{a}^{(i)}) - \frac{1}{m_{\mathbf{a}=1}} \sum_{i=1}^{m} P(\hat{\mathbf{y}}^{(i)} = 1|\mathbf{x}^{(i)}) \cdot \mathbf{a}^{(i)} \right|}^{\Delta_{DP}\text{ Loss}} \quad (4)$$

where $P(\hat{\mathbf{y}}^{(i)} = 1|\mathbf{x}^{(i)})$ is the predicted probability of the positive class for the example $\mathbf{x}^{(i)}$.

3.7.1 TASK 9

Let's first define the model:

[70]:
```python
class CustomModel_Extended(CustomModel):
    def __init__(self, alpha=6, **kwargs):
        super(CustomModel_Extended, self).__init__(**kwargs)

        # Define the losses to track
        self.total_loss_tracker = keras.metrics.Mean(name="total_loss")
        self.loss_delta_dp_tracker = keras.metrics.Mean(name="loss_delta_dp")

        # Hyperparameter alpha
        self.alpha = alpha

    @property
    def metrics(self):
        return [
            self.total_loss_tracker,
            self.loss_cross_entropy_tracker,
            self.loss_delta_dp_tracker,

            self.accuracy_tracker,
            self.weighted_accuracy_tracker,
            self.delta_dp_tracker,
            self.y_pred_labels_female_avg_tracker,
            self.y_pred_labels_male_avg_tracker,
        ]

    def train_step(self, data):
        X, y, a = data[0]  # Unpack the three inputs
        with tf.GradientTape() as tape:
            ## START YOU CODE HERE
            y_pred = self(X)
            # Binary cross-entropy loss
            loss_cross_entropy = tf.reduce_mean(tf.keras.losses.
↪binary_crossentropy(y, y_pred))

            # Demographic Parity (DP) differentiable loss (soft predictions)
            a = tf.reshape(a, [-1])
            y_pred = tf.reshape(y_pred, [-1])
            female_mask = tf.equal(a, 0)
            male_mask = tf.equal(a, 1)
            m_female = tf.reduce_sum(tf.cast(female_mask, tf.float32))
            m_male = tf.reduce_sum(tf.cast(male_mask, tf.float32))
            y_pred_female_avg = tf.reduce_sum(y_pred * tf.cast(female_mask, tf.
↪float32)) / (m_female + 1e-8)
            y_pred_male_avg = tf.reduce_sum(y_pred * tf.cast(male_mask, tf.
↪float32)) / (m_male + 1e-8)
            loss_delta_dp = tf.abs(y_pred_female_avg - y_pred_male_avg)
            # Total loss: cross-entropy + alpha * DP loss
            total_loss = loss_cross_entropy + self.alpha * loss_delta_dp
            ## END

        # Compute gradients and update weights
        grads = tape.gradient(total_loss, self.trainable_weights)
        self.optimizer.apply_gradients(zip(grads, self.trainable_weights))
        self.total_loss_tracker.update_state(total_loss)
        self.loss_cross_entropy_tracker.update_state(loss_cross_entropy)
        self.loss_delta_dp_tracker.update_state(loss_delta_dp)

        # Perform evaluation
        X, y, a = data[0]  # Unpack the three inputs
        y_pred = self(X)
        accuracy, weighted_accuracy, delta_dp, y_pred_labels_female_avg,␣
↪y_pred_labels_male_avg = compute_test_metrics(
            y, y_pred, a)

        # Update evaluation trackers
        self.accuracy_tracker.update_state(accuracy)
        self.weighted_accuracy_tracker.update_state(weighted_accuracy)
        self.delta_dp_tracker.update_state(delta_dp)
        self.y_pred_labels_female_avg_tracker.
↪update_state(y_pred_labels_female_avg)
        self.y_pred_labels_male_avg_tracker.update_state(y_pred_labels_male_avg)

        return {
            "total_loss": self.total_loss_tracker.result(),
            "loss_cross_entropy": self.loss_cross_entropy_tracker.result(),
            "loss_delta_dp": self.loss_delta_dp_tracker.result(),

            "accuracy": self.accuracy_tracker.result(),
            "weighted_accuracy": self.weighted_accuracy_tracker.result(),
            "delta_dp": self.delta_dp_tracker.result(),
            "y_pred_labels_female_avg": self.y_pred_labels_female_avg_tracker.
↪result(),
            "y_pred_labels_male_avg": self.y_pred_labels_male_avg_tracker.
↪result(),
        }
```

Let's now train our model:

[71]:
```
###### PLEASE DO NOT REMOVE THIS!#########
# Set the random seed for NumPy
np.random.seed(seed_value)
# Set the random seed for TensorFlow
tf.random.set_seed(seed_value)
# Set the random seed for Keras
tf.keras.utils.set_random_seed(seed_value)
##########################################
X_train, y_train, a_train, X_test, y_test, a_test, headers =␣
↪load_data(remove_gender_feature=True)

model = CustomModel_Extended(alpha=0.64)
model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0001))
model.build(input_shape=X_train.shape)
print(model.summary())
history = model.fit([X_train, y_train, a_train], epochs=50, batch_size=128)
```

```
Model: "custom_model__extended_84"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_270 (Dense)           multiple                  7168

 dense_271 (Dense)           multiple                  2080

 dense_272 (Dense)           multiple                  33

=================================================================
Total params: 9297 (36.32 KB)

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_270 (Dense)           multiple                  7168

 dense_271 (Dense)           multiple                  2080

 dense_272 (Dense)           multiple                  33
```

```
=================================================================
Total params: 9297 (36.32 KB)
Trainable params: 9281 (36.25 KB)
Non-trainable params: 16 (64.00 Byte)
_________________________________________________________________
None
Epoch 1/50
255/255 [==============================] - 2s 4ms/step - total_loss: 22.0026 -
loss_cross_entropy: 21.9789 - loss_delta_dp: 0.0370 - accuracy: 0.7613 -
weighted_accuracy: 0.8019 - delta_dp: 0.0367 - y_pred_labels_female_avg: 0.2492
- y_pred_labels_male_avg: 0.2586
Epoch 2/50
255/255 [==============================] - 1s 3ms/step - total_loss: 0.4986 -
loss_cross_entropy: 0.4711 - loss_delta_dp: 0.0430 - accuracy: 0.7925 -
weighted_accuracy: 0.8238 - delta_dp: 0.0425 - y_pred_labels_female_avg: 0.2330
- y_pred_labels_male_avg: 0.2731
Epoch 3/50
255/255 [==============================] - 1s 3ms/step - total_loss: 0.4742 -
loss_cross_entropy: 0.4421 - loss_delta_dp: 0.0503 - accuracy: 0.7974 -
weighted_accuracy: 0.8256 - delta_dp: 0.0496 - y_pred_labels_female_avg: 0.2194
- y_pred_labels_male_avg: 0.2678
Epoch 4/50
255/255 [==============================] - 1s 3ms/step - total_loss: 0.4568 -
loss_cross_entropy: 0.4250 - loss_delta_dp: 0.0497 - accuracy: 0.7997 -
weighted_accuracy: 0.8271 - delta_dp: 0.0495 - y_pred_labels_female_avg: 0.2172
- y_pred_labels_male_avg: 0.2654
Epoch 5/50
255/255 [==============================] - 1s 5ms/step - total_loss: 0.4494 -
loss_cross_entropy: 0.4138 - loss_delta_dp: 0.0557 - accuracy: 0.8044 -
weighted_accuracy: 0.8343 - delta_dp: 0.0555 - y_pred_labels_female_avg: 0.2098
- y_pred_labels_male_avg: 0.2634
Epoch 6/50
255/255 [==============================] - 1s 4ms/step - total_loss: 0.4361 -
loss_cross_entropy: 0.3988 - loss_delta_dp: 0.0584 - accuracy: 0.8102 -
weighted_accuracy: 0.8405 - delta_dp: 0.0569 - y_pred_labels_female_avg: 0.2041
- y_pred_labels_male_avg: 0.2578
Epoch 7/50
255/255 [==============================] - 1s 3ms/step - total_loss: 0.4286 -
loss_cross_entropy: 0.3918 - loss_delta_dp: 0.0575 - accuracy: 0.8233 -
weighted_accuracy: 0.8457 - delta_dp: 0.0571 - y_pred_labels_female_avg: 0.2052
- y_pred_labels_male_avg: 0.2592
Epoch 8/50
255/255 [==============================] - 1s 3ms/step - total_loss: 0.4490 -
loss_cross_entropy: 0.4122 - loss_delta_dp: 0.0574 - accuracy: 0.8162 -
weighted_accuracy: 0.8443 - delta_dp: 0.0570 - y_pred_labels_female_avg: 0.2065
- y_pred_labels_male_avg: 0.2585
Epoch 9/50
255/255 [==============================] - 1s 3ms/step - total_loss: 0.4217 -
```

```
loss_cross_entropy: 0.3859 - loss_delta_dp: 0.0561 - accuracy: 0.8261 -
weighted_accuracy: 0.8480 - delta_dp: 0.0550 - y_pred_labels_female_avg: 0.2041
- y_pred_labels_male_avg: 0.2547
Epoch 10/50
255/255 [==============================] - 1s 3ms/step - total_loss: 0.4195 -
loss_cross_entropy: 0.3848 - loss_delta_dp: 0.0543 - accuracy: 0.8308 -
weighted_accuracy: 0.8476 - delta_dp: 0.0531 - y_pred_labels_female_avg: 0.2054
- y_pred_labels_male_avg: 0.2534
Epoch 11/50
255/255 [==============================] - 1s 3ms/step - total_loss: 0.4118 -
loss_cross_entropy: 0.3755 - loss_delta_dp: 0.0567 - accuracy: 0.8321 -
weighted_accuracy: 0.8493 - delta_dp: 0.0551 - y_pred_labels_female_avg: 0.2077
- y_pred_labels_male_avg: 0.2519
Epoch 12/50
255/255 [==============================] - 1s 5ms/step - total_loss: 0.4150 -
loss_cross_entropy: 0.3776 - loss_delta_dp: 0.0585 - accuracy: 0.8354 -
weighted_accuracy: 0.8499 - delta_dp: 0.0581 - y_pred_labels_female_avg: 0.2016
- y_pred_labels_male_avg: 0.2510
Epoch 13/50
255/255 [==============================] - 1s 4ms/step - total_loss: 0.4187 -
loss_cross_entropy: 0.3827 - loss_delta_dp: 0.0561 - accuracy: 0.8309 -
weighted_accuracy: 0.8497 - delta_dp: 0.0542 - y_pred_labels_female_avg: 0.2023
- y_pred_labels_male_avg: 0.2502
Epoch 14/50
255/255 [==============================] - 1s 3ms/step - total_loss: 0.4043 -
loss_cross_entropy: 0.3683 - loss_delta_dp: 0.0563 - accuracy: 0.8349 -
weighted_accuracy: 0.8527 - delta_dp: 0.0563 - y_pred_labels_female_avg: 0.1997
- y_pred_labels_male_avg: 0.2474
Epoch 15/50
255/255 [==============================] - 1s 3ms/step - total_loss: 0.4939 -
loss_cross_entropy: 0.4560 - loss_delta_dp: 0.0592 - accuracy: 0.8308 -
weighted_accuracy: 0.8461 - delta_dp: 0.0586 - y_pred_labels_female_avg: 0.2059
- y_pred_labels_male_avg: 0.2567
Epoch 16/50
255/255 [==============================] - 1s 2ms/step - total_loss: 0.4280 -
loss_cross_entropy: 0.3918 - loss_delta_dp: 0.0567 - accuracy: 0.8334 -
weighted_accuracy: 0.8521 - delta_dp: 0.0566 - y_pred_labels_female_avg: 0.2011
- y_pred_labels_male_avg: 0.2503
Epoch 17/50
255/255 [==============================] - 1s 3ms/step - total_loss: 0.4120 -
loss_cross_entropy: 0.3740 - loss_delta_dp: 0.0593 - accuracy: 0.8334 -
weighted_accuracy: 0.8525 - delta_dp: 0.0587 - y_pred_labels_female_avg: 0.1983
- y_pred_labels_male_avg: 0.2496
Epoch 18/50
255/255 [==============================] - 1s 3ms/step - total_loss: 0.4105 -
loss_cross_entropy: 0.3716 - loss_delta_dp: 0.0608 - accuracy: 0.8364 -
weighted_accuracy: 0.8519 - delta_dp: 0.0593 - y_pred_labels_female_avg: 0.1966
- y_pred_labels_male_avg: 0.2484
```

```
Epoch 19/50
255/255 [==============================] - 1s 2ms/step - total_loss: 0.4071 -
loss_cross_entropy: 0.3703 - loss_delta_dp: 0.0575 - accuracy: 0.8339 -
weighted_accuracy: 0.8533 - delta_dp: 0.0566 - y_pred_labels_female_avg: 0.1963
- y_pred_labels_male_avg: 0.2461
Epoch 20/50
255/255 [==============================] - 1s 3ms/step - total_loss: 0.4096 -
loss_cross_entropy: 0.3730 - loss_delta_dp: 0.0572 - accuracy: 0.8345 -
weighted_accuracy: 0.8522 - delta_dp: 0.0564 - y_pred_labels_female_avg: 0.1978
- y_pred_labels_male_avg: 0.2477
Epoch 21/50
255/255 [==============================] - 1s 3ms/step - total_loss: 0.4032 -
loss_cross_entropy: 0.3673 - loss_delta_dp: 0.0560 - accuracy: 0.8333 -
weighted_accuracy: 0.8535 - delta_dp: 0.0555 - y_pred_labels_female_avg: 0.1984
- y_pred_labels_male_avg: 0.2478
Epoch 22/50
255/255 [==============================] - 1s 3ms/step - total_loss: 0.4132 -
loss_cross_entropy: 0.3754 - loss_delta_dp: 0.0592 - accuracy: 0.8340 -
weighted_accuracy: 0.8528 - delta_dp: 0.0584 - y_pred_labels_female_avg: 0.1981
- y_pred_labels_male_avg: 0.2488
Epoch 23/50
255/255 [==============================] - 1s 2ms/step - total_loss: 0.4003 -
loss_cross_entropy: 0.3637 - loss_delta_dp: 0.0572 - accuracy: 0.8331 -
weighted_accuracy: 0.8558 - delta_dp: 0.0571 - y_pred_labels_female_avg: 0.1964
- y_pred_labels_male_avg: 0.2449
Epoch 24/50
255/255 [==============================] - 1s 3ms/step - total_loss: 0.3990 -
loss_cross_entropy: 0.3614 - loss_delta_dp: 0.0588 - accuracy: 0.8354 -
weighted_accuracy: 0.8544 - delta_dp: 0.0578 - y_pred_labels_female_avg: 0.1966
- y_pred_labels_male_avg: 0.2464
Epoch 25/50
255/255 [==============================] - 1s 3ms/step - total_loss: 0.4027 -
loss_cross_entropy: 0.3655 - loss_delta_dp: 0.0581 - accuracy: 0.8352 -
weighted_accuracy: 0.8527 - delta_dp: 0.0578 - y_pred_labels_female_avg: 0.1976
- y_pred_labels_male_avg: 0.2478
Epoch 26/50
255/255 [==============================] - 1s 2ms/step - total_loss: 0.4061 -
loss_cross_entropy: 0.3693 - loss_delta_dp: 0.0575 - accuracy: 0.8345 -
weighted_accuracy: 0.8530 - delta_dp: 0.0559 - y_pred_labels_female_avg: 0.1984
- y_pred_labels_male_avg: 0.2482
Epoch 27/50
255/255 [==============================] - 1s 3ms/step - total_loss: 0.3961 -
loss_cross_entropy: 0.3590 - loss_delta_dp: 0.0580 - accuracy: 0.8361 -
weighted_accuracy: 0.8544 - delta_dp: 0.0576 - y_pred_labels_female_avg: 0.1945
- y_pred_labels_male_avg: 0.2462
Epoch 28/50
255/255 [==============================] - 1s 3ms/step - total_loss: 0.4018 -
loss_cross_entropy: 0.3652 - loss_delta_dp: 0.0573 - accuracy: 0.8355 -
```

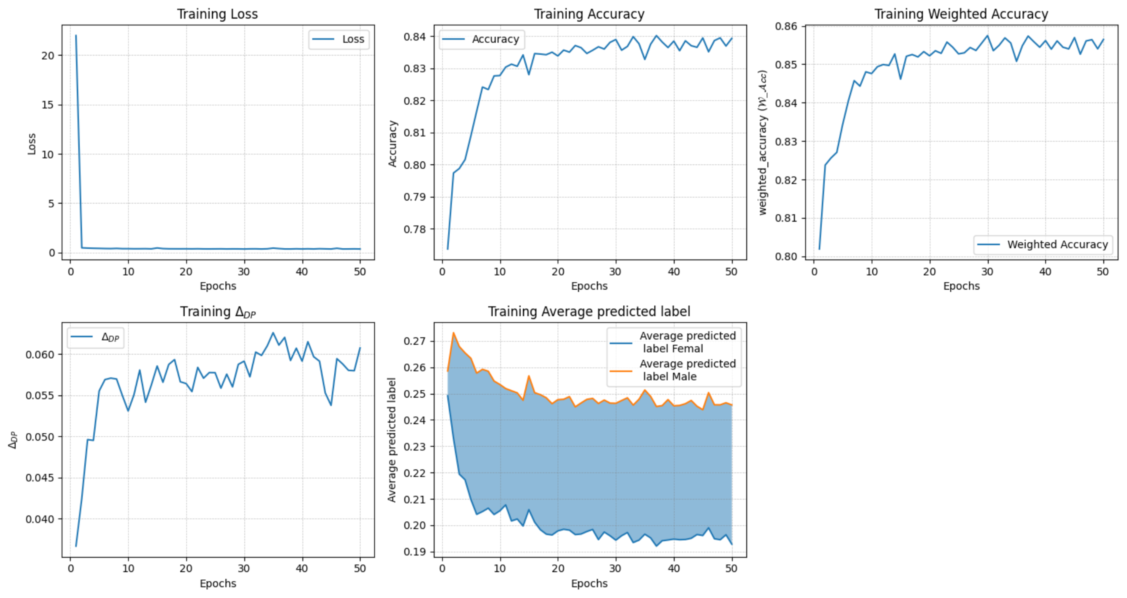
```
weighted_accuracy: 0.8536 - delta_dp: 0.0560 - y_pred_labels_female_avg: 0.1974
- y_pred_labels_male_avg: 0.2475
Epoch 29/50
255/255 [==============================] - 1s 4ms/step - total_loss: 0.4011 -
loss_cross_entropy: 0.3634 - loss_delta_dp: 0.0588 - accuracy: 0.8383 -
weighted_accuracy: 0.8555 - delta_dp: 0.0588 - y_pred_labels_female_avg: 0.1960
- y_pred_labels_male_avg: 0.2464
Epoch 30/50
255/255 [==============================] - 1s 4ms/step - total_loss: 0.3924 -
loss_cross_entropy: 0.3541 - loss_delta_dp: 0.0599 - accuracy: 0.8414 -
weighted_accuracy: 0.8575 - delta_dp: 0.0592 - y_pred_labels_female_avg: 0.1943
- y_pred_labels_male_avg: 0.2463
Epoch 31/50
255/255 [==============================] - 1s 4ms/step - total_loss: 0.4053 -
loss_cross_entropy: 0.3678 - loss_delta_dp: 0.0587 - accuracy: 0.8369 -
weighted_accuracy: 0.8535 - delta_dp: 0.0572 - y_pred_labels_female_avg: 0.1960
- y_pred_labels_male_avg: 0.2473
Epoch 32/50
255/255 [==============================] - 1s 4ms/step - total_loss: 0.4100 -
loss_cross_entropy: 0.3707 - loss_delta_dp: 0.0614 - accuracy: 0.8406 -
weighted_accuracy: 0.8549 - delta_dp: 0.0603 - y_pred_labels_female_avg: 0.1972
- y_pred_labels_male_avg: 0.2484
Epoch 33/50
255/255 [==============================] - 1s 3ms/step - total_loss: 0.3943 -
loss_cross_entropy: 0.3557 - loss_delta_dp: 0.0604 - accuracy: 0.8359 -
weighted_accuracy: 0.8569 - delta_dp: 0.0598 - y_pred_labels_female_avg: 0.1934
- y_pred_labels_male_avg: 0.2456
Epoch 34/50
255/255 [==============================] - 1s 4ms/step - total_loss: 0.4115 -
loss_cross_entropy: 0.3719 - loss_delta_dp: 0.0619 - accuracy: 0.8402 -
weighted_accuracy: 0.8555 - delta_dp: 0.0610 - y_pred_labels_female_avg: 0.1943
- y_pred_labels_male_avg: 0.2478
Epoch 35/50
255/255 [==============================] - 1s 4ms/step - total_loss: 0.4826 -
loss_cross_entropy: 0.4416 - loss_delta_dp: 0.0642 - accuracy: 0.8318 -
weighted_accuracy: 0.8508 - delta_dp: 0.0626 - y_pred_labels_female_avg: 0.1966
- y_pred_labels_male_avg: 0.2513
Epoch 36/50
255/255 [==============================] - 1s 4ms/step - total_loss: 0.4352 -
loss_cross_entropy: 0.3960 - loss_delta_dp: 0.0612 - accuracy: 0.8425 -
weighted_accuracy: 0.8548 - delta_dp: 0.0611 - y_pred_labels_female_avg: 0.1952
- y_pred_labels_male_avg: 0.2489
Epoch 37/50
255/255 [==============================] - 1s 4ms/step - total_loss: 0.3993 -
loss_cross_entropy: 0.3590 - loss_delta_dp: 0.0631 - accuracy: 0.8407 -
weighted_accuracy: 0.8573 - delta_dp: 0.0620 - y_pred_labels_female_avg: 0.1921
- y_pred_labels_male_avg: 0.2451
Epoch 38/50
```

```
255/255 [==============================] - 1s 4ms/step - total_loss: 0.3959 -
loss_cross_entropy: 0.3573 - loss_delta_dp: 0.0603 - accuracy: 0.8368 -
weighted_accuracy: 0.8558 - delta_dp: 0.0592 - y_pred_labels_female_avg: 0.1941
- y_pred_labels_male_avg: 0.2454
Epoch 39/50
255/255 [==============================] - 1s 3ms/step - total_loss: 0.4118 -
loss_cross_entropy: 0.3724 - loss_delta_dp: 0.0617 - accuracy: 0.8382 -
weighted_accuracy: 0.8544 - delta_dp: 0.0607 - y_pred_labels_female_avg: 0.1944
- y_pred_labels_male_avg: 0.2477
Epoch 40/50
255/255 [==============================] - 1s 4ms/step - total_loss: 0.3978 -
loss_cross_entropy: 0.3597 - loss_delta_dp: 0.0595 - accuracy: 0.8383 -
weighted_accuracy: 0.8562 - delta_dp: 0.0592 - y_pred_labels_female_avg: 0.1947
- y_pred_labels_male_avg: 0.2453
Epoch 41/50
255/255 [==============================] - 1s 6ms/step - total_loss: 0.4121 -
loss_cross_entropy: 0.3726 - loss_delta_dp: 0.0618 - accuracy: 0.8329 -
weighted_accuracy: 0.8539 - delta_dp: 0.0615 - y_pred_labels_female_avg: 0.1945
- y_pred_labels_male_avg: 0.2455
Epoch 42/50
255/255 [==============================] - 1s 5ms/step - total_loss: 0.3979 -
loss_cross_entropy: 0.3590 - loss_delta_dp: 0.0609 - accuracy: 0.8353 -
weighted_accuracy: 0.8561 - delta_dp: 0.0597 - y_pred_labels_female_avg: 0.1946
- y_pred_labels_male_avg: 0.2461
Epoch 43/50
255/255 [==============================] - 2s 8ms/step - total_loss: 0.4192 -
loss_cross_entropy: 0.3810 - loss_delta_dp: 0.0597 - accuracy: 0.8322 -
weighted_accuracy: 0.8544 - delta_dp: 0.0591 - y_pred_labels_female_avg: 0.1950
- y_pred_labels_male_avg: 0.2474
Epoch 44/50
255/255 [==============================] - 2s 6ms/step - total_loss: 0.4039 -
loss_cross_entropy: 0.3679 - loss_delta_dp: 0.0563 - accuracy: 0.8359 -
weighted_accuracy: 0.8540 - delta_dp: 0.0553 - y_pred_labels_female_avg: 0.1964
- y_pred_labels_male_avg: 0.2452
Epoch 45/50
255/255 [==============================] - 1s 5ms/step - total_loss: 0.3899 -
loss_cross_entropy: 0.3552 - loss_delta_dp: 0.0543 - accuracy: 0.8405 -
weighted_accuracy: 0.8569 - delta_dp: 0.0538 - y_pred_labels_female_avg: 0.1961
- y_pred_labels_male_avg: 0.2438
Epoch 46/50
255/255 [==============================] - 1s 4ms/step - total_loss: 0.4640 -
loss_cross_entropy: 0.4252 - loss_delta_dp: 0.0606 - accuracy: 0.8287 -
weighted_accuracy: 0.8526 - delta_dp: 0.0595 - y_pred_labels_female_avg: 0.1990
- y_pred_labels_male_avg: 0.2503
Epoch 47/50
255/255 [==============================] - 1s 4ms/step - total_loss: 0.3936 -
loss_cross_entropy: 0.3557 - loss_delta_dp: 0.0593 - accuracy: 0.8408 -
weighted_accuracy: 0.8561 - delta_dp: 0.0588 - y_pred_labels_female_avg: 0.1949
```

```
- y_pred_labels_male_avg: 0.2457
Epoch 48/50
255/255 [==============================] - 1s 4ms/step - total_loss: 0.3961 -
loss_cross_entropy: 0.3587 - loss_delta_dp: 0.0585 - accuracy: 0.8385 -
weighted_accuracy: 0.8564 - delta_dp: 0.0580 - y_pred_labels_female_avg: 0.1945
- y_pred_labels_male_avg: 0.2457
Epoch 49/50
255/255 [==============================] - 1s 4ms/step - total_loss: 0.4042 -
loss_cross_entropy: 0.3663 - loss_delta_dp: 0.0592 - accuracy: 0.8360 -
weighted_accuracy: 0.8540 - delta_dp: 0.0580 - y_pred_labels_female_avg: 0.1964
- y_pred_labels_male_avg: 0.2465
Epoch 50/50
255/255 [==============================] - 1s 4ms/step - total_loss: 0.3963 -
loss_cross_entropy: 0.3568 - loss_delta_dp: 0.0617 - accuracy: 0.8404 -
weighted_accuracy: 0.8565 - delta_dp: 0.0608 - y_pred_labels_female_avg: 0.1928
- y_pred_labels_male_avg: 0.2457
```

Plot learning curve

[72]:
```python
plt.figure(figsize=(15, 8))

ax1 = plt.subplot(2, 3, 1)
x = range(1, len(history.history['loss_cross_entropy']) + 1)
plt.plot(x, history.history['loss_cross_entropy'], label='Loss')
plt.title('Training Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.grid(True, linestyle='--', linewidth=0.5, color='gray', alpha=0.5)
plt.legend()

ax2 = plt.subplot(2, 3, 2)
plt.plot(x, history.history['accuracy'], label='Accuracy')
plt.title('Training Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.grid(True, linestyle='--', linewidth=0.5, color='gray', alpha=0.5)
plt.legend()

ax2 = plt.subplot(2, 3, 3)
plt.plot(x, history.history['weighted_accuracy'], label='Weighted Accuracy')
plt.title('Training Weighted Accuracy')
plt.xlabel('Epochs')
plt.ylabel('weighted_accuracy ($\mathcal{W\_Acc}$)')
plt.grid(True, linestyle='--', linewidth=0.5, color='gray', alpha=0.5)
plt.legend()

ax2 = plt.subplot(2, 3, 4)
plt.plot(x, history.history['delta_dp'], label='$\\Delta_{DP}$ ')
plt.title('Training $\\Delta_{DP}$')
plt.xlabel('Epochs')
plt.ylabel('$\\Delta_{DP}$')
plt.grid(True, linestyle='--', linewidth=0.5, color='gray', alpha=0.5)
plt.legend()

ax2 = plt.subplot(2, 3, 5)
plt.plot(x, history.history['y_pred_labels_female_avg'], label='Average␣
 ↪predicted\n label Femal')
plt.plot(x, history.history['y_pred_labels_male_avg'], label='Average␣
 ↪predicted\n label Male')
plt.fill_between(x, history.history['y_pred_labels_female_avg'],
                 history.history['y_pred_labels_male_avg'], alpha=0.5)
plt.title('Training Average predicted label')
plt.xlabel('Epochs')
plt.ylabel('Average predicted label')
plt.grid(True, linestyle='--', linewidth=0.5, color='gray', alpha=0.5)
plt.legend()

plt.tight_layout()
plt.show()
```



3.8 Exploring the Impact of α on Neural Network Performance

3.8.1 TASK 10

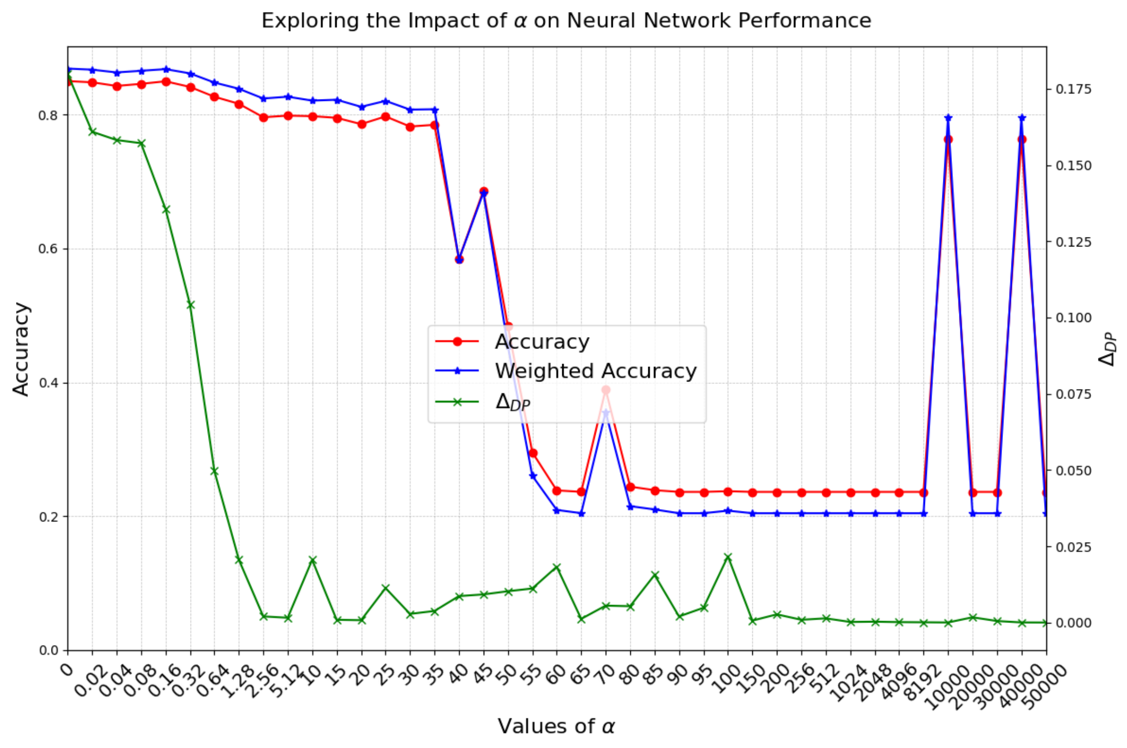
Determine a range of α that produces reasonable results when training the neural network and report this range in your writeup. Carry out a sequence of network trainings with α varying over this range, and produce a plot (similar to the one described above) where accuracy (overall and weighted) and α are shown as functions of α.

[73]:
```
###### PLEASE DO NOT REMOVE THIS!#########
# Set the random seed for NumPy
np.random.seed(seed_value)
# Set the random seed for TensorFlow
tf.random.set_seed(seed_value)
# Set the random seed for Keras
tf.keras.utils.set_random_seed(seed_value)
##########################################
X_train, y_train, a_train, X_test, y_test, a_test, headers =␣
 ↪load_data(remove_gender_feature=True)

alpha_values = [0, 0.02, 0.04, 0.08, 0.16, 0.32, 0.64, 1.28, 2.56, 5.12, 10,
                15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 80, 85, 90, 95,
                100, 150, 200, 256, 512, 1024, 2048, 4096, 8192, 10000, 20000,␣
 ↪30000,40000,50000]
accuracy_values = []
weighted_accuracy_values = []
delta_dp_values = []

for alpha in tqdm(alpha_values, ncols=80):
    ## START YOU CODE HERE
    # Reinitialize model for each alpha
    model = CustomModel_Extended(alpha=alpha)
    model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0001))
    model.build(input_shape=X_train.shape)
    # Train for fewer epochs for speed (e.g., 10 epochs)
    history = model.fit([X_train, y_train, a_train], epochs=10, batch_size=128,␣
 ↪verbose=0)
    # Evaluate on test set
    y_pred = model.predict(X_test, verbose=0)
    accuracy, weighted_accuracy, delta_dp, _, _ = compute_test_metrics(y_test,␣
 ↪y_pred, a_test)
    accuracy_values.append(float(accuracy.numpy()))
    weighted_accuracy_values.append(float(weighted_accuracy.numpy()))
    delta_dp_values.append(float(delta_dp.numpy()))
    ## END
```

```
100%|                        | 41/41 [05:16<00:00,  7.71s/it]
```

[74]:
```
fig, ax1 = plt.subplots(figsize=(12, 8))
plt.grid(True, linestyle='--', linewidth=0.5, color='gray', alpha=0.5)
x = list(range(len(alpha_values)))

ax1.plot(x, accuracy_values, 'r-o')
ax1.plot(x, weighted_accuracy_values, 'b-*')
```

```
ax1.set_xlabel('Values of $\\alpha$', fontsize=16)
ax1.set_ylabel('Accuracy', fontsize=16)
ax1.set_xlim(0, x[-1])
ax1.set_ylim(0, None)
ax1.set_xticks(x)

ax1.set_xticklabels(alpha_values, rotation=45, fontsize=14)

ax2 = ax1.twinx()
ax2.plot(x, delta_dp_values, 'g-x')
ax2.set_ylabel('$\\Delta_{DP}$', fontsize=16)
fig.legend(labels = ('Accuracy','Weighted Accuracy', '$\\Delta_{DP}$'),␣
 ↪loc='center', fontsize=16)

fig.suptitle('Exploring the Impact of $\\alpha$ on Neural Network Performance',␣
 ↪fontsize=16)
plt.tight_layout()
plt.show()
```



3.8.2 TASK 11

What do you observe from the plot above?

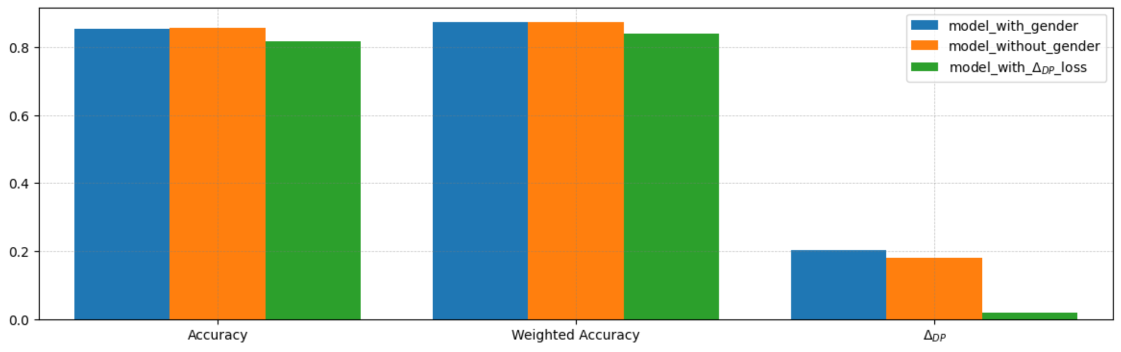
From the plot, we observe that as the value of α increases, the demographic parity gap (Δ_{DP}) generally decreases, indicating improved fairness. However, increasing α too much can lead to a reduction in both overall accuracy and weighted accuracy, as the model prioritizes fairness over predictive performance. There is a range of α values where the model achieves a good balance between accuracy and fairness, but beyond this range, the trade-off gets increasingly noticeable.. This demonstrates the importance of tuning α to achieve the desired balance for a given application.

3.9 Overall Performance Comparaison

Let's now see a performance comparaison of the the three models we have have built in this assigment: - Model trained with gender features. - Model trained without gender features. - Model trained without gender features and with cross-entropy and Δ_{DP} loss.

[75]:
```
data_list = [['model_with_$\\Delta_{DP}$_loss', 'Accuracy', accuracy_values[7]],
             ['model_with_$\\Delta_{DP}$_loss', 'Weighted Accuracy',␣
 ↪weighted_accuracy_values[7]],
             ['model_with_$\\Delta_{DP}$_loss', '$\\Delta_{DP}$',␣
 ↪delta_dp_values[7]]]
df_new = pd.concat([df, pd.DataFrame(data_list, columns=['model', 'metric',␣
 ↪'value'])], ignore_index=True)
# Draw plot
g = sns.catplot(x="metric", y='value', hue="model", data=df_new, kind='bar',
                hue_order=['model_with_gender',␣
 ↪'model_without_gender','model_with_$\\Delta_{DP}$_loss'],
                legend=False, height=4, aspect=3, saturation=1)

g.despine(top=False, right=False)
plt.grid(True, linestyle='--', linewidth=0.5, color='gray', alpha=0.5)

plt.xlabel("")
plt.ylabel("")
plt.gca().legend().set_title('')
plt.show()
```

```
c:\Users\micha\Desktop\deakin stuff\python3.11_task_3.3HD\Lib\site-
packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to
tight
  self._figure.tight_layout(*args, **kwargs)
```



3.9.1 TASK 12

What do you observe from the plot above?

The final comparison plot shows that the model trained with both cross-entropy and Δ_{DP} loss achieves a better balance between accuracy and fairness compared to the other two models. While the model with gender features has the highest accuracy, it also has the largest demographic parity gap, indicating less fairness. The model without gender features is fairer but may sacrifice some accuracy. The fairness-regularized model (with Δ_{DP} loss) reduces the demographic parity gap significantly while maintaining competitive accuracy, demonstrating that fairness constraints can help mitigate bias without severely impacting performance.

---

4 Congratulations!

Congratulations on completing the assignment! Your dedication and effort are commendable. By successfully working through the coding exercises and written exercises, you have demonstrated a strong understanding of the concepts and principles related to fairness in AI.

It is crucial to recognize the importance of considering fairness in AI systems. As AI technologies become more prevalent in various domains, including decision-making processes, it is imperative to ensure that these systems treat all individuals fairly and avoid perpetuating biases or discrimination. By incorporating fairness considerations into the design, development, and evaluation of AI models, we can strive to create more equitable and just outcomes for everyone. Your engagement with this assignment highlights the significance of fairness in AI and the need for continuous efforts to address biases and promote fairness in algorithmic decision-making. Well done!

Congratulations on finishing this notebook!