```
import matplotlib.pyplot as plt
import pickle as pck
import numpy as np
import pandas as pd
import tensorflow.compat.v1 as tf

tf.disable_eager_execution()
tf.logging.set_verbosity(tf.logging.ERROR)

from sklearn.compose import make_column_transformer
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
from aif360.metrics import ClassificationMetric
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.model_selection import StratifiedKFold
from aif360.datasets import AdultDataset
from aif360.algorithms.preprocessing.optim_preproc_helpers.data_preproc_functions import (
    load_preproc_data_adult,
)
from aif360.sklearn.preprocessing import ReweighingMeta
from aif360.sklearn.inprocessing import AdversarialDebiasing
from aif360.sklearn.postprocessing import CalibratedEqualizedOdds, PostProcessingMeta
from aif360.sklearn.datasets import fetch_adult
from aif360.sklearn.metrics import (
    disparate_impact_ratio,
    average_odds_error,
    generalized_fpr,
)
import keras
from keras.models import Sequential
from keras.layers import Dense
from scikeras.wrappers import KerasClassifier
from typing import Dict, Iterable, Any

privileged_groups = [{"sex": 1}]
unprivileged_groups = [{"sex": 0}]
dataset_orig = load_preproc_data_adult(["sex"])


# STEP 3: We split between training and test set.
train, test = dataset_orig.split([0.7], shuffle=True)
```

```
print("training data size", train.features.shape)
print("dataset feature names", train.feature_names)

# Normalize the dataset, both train and test. This should always be done in any machine lea
x_train = train.features
y_train = train.labels.ravel()

x_test = test.features
y_test = test.labels.ravel()


skf = StratifiedKFold(n_splits=5, random_state=7, shuffle=True)

model_scores_mlp = dict()
model_scores_lr = dict()
model_scores_svm = dict()
mlp_metrics = dict()
lr_metrics = dict()
svm_metrics = dict()


def get_mlp_classes(test, model):
    y_prob = model.predict(test.features)
    y_prob = pd.Series(y_prob.ravel())
    y_prob[y_prob > 0.5] = 1
    y_prob[y_prob < 0.5] = 0
    return np.array(y_prob)


def create_model(lr):
    # create model
    model = Sequential()
    model.add(Dense(30, input_dim=x_train.shape[1], activation="sigmoid"))
    model.add(Dense(30, activation="sigmoid"))
    model.add(Dense(1, activation="sigmoid"))
    opt = keras.optimizer_v1.adam(lr=lr)
    model.compile(loss="binary_crossentropy", metrics=["accuracy"], optimizer=opt)
    return model


def train_model(model_type, train, reg_param):
    x_train = train.features
    y_train = train.labels.ravel()
    if model_type == "mlp":
        model = create_model(reg_param)
        model.fit(x_train, y_train, epochs=100, batch_size=100, verbose=1)
```

```
        return model
    elif model_type == "lr":
        clf = make_pipeline(
            StandardScaler(),
            LogisticRegression(
                C=reg_param, max_iter=1000, verbose=1, solver="liblinear"
            ),
        )
        clf.fit(x_train, y_train)
        return clf
    elif model_type == "svm":
        clf = make_pipeline(StandardScaler(), SVC(C=reg_param, verbose=1, max_iter=500))
        clf.fit(x_train, y_train)
        return clf
    else:
        print("Please enter correct model type")


def test_metric(test, y_pred):
    test_pred = test.copy()
    test_pred.labels = y_pred
    metric = ClassificationMetric(
        test,
        test_pred,
        unprivileged_groups=unprivileged_groups,
        privileged_groups=privileged_groups,
    )
    metric = metric.equal_opportunity_difference()
    return metric


def val_score(lr=None, C_svm=None, C_lr=None):

    scores = []
    metrics = []
    for train_index, val_index in skf.split(x_train, y_train):
        train_cv = train.subset(train_index)
        test_cv = train.subset(val_index)
        if lr != None:
            clf = train_model("mlp", train_cv, lr)
            score = clf.evaluate(test_cv.features, test_cv.labels.ravel(), verbose=0)[1]
            labels = get_mlp_classes(test_cv, clf)
            metric = test_metric(test_cv, labels)
        elif C_svm != None:
            clf = train_model("svm", train_cv, C_svm)
            score = clf.score(test_cv.features, test_cv.labels.ravel())
```

```
            labels = clf.predict(test_cv.features)
            metric = test_metric(test_cv, labels)

        elif C_lr != None:
            clf = train_model("lr", train_cv, C_lr)
            score = clf.score(test_cv.features, test_cv.labels.ravel())
            labels = clf.predict(test_cv.features)
            metric = test_metric(test_cv, labels)

        scores.append(score)
        metrics.append(metric)

    return np.mean(scores), np.mean(metrics)


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

# Function for getting the best hyperparameter according to accuracy
best_hyperparam_acc = lambda model_scores: max(model_scores, key=model_scores.get)
# Function for getting the best hyperparameter with minimum absolute value of
# fairness score
best_hyperparam_sp = lambda model_scores: min(
    model_scores, key=lambda x: abs(model_scores[x])
)

for i in learning_rates:
    model_scores_mlp[str(i)], mlp_metrics[str(i)] = val_score(lr=i)

mlp_lr_acc = float(best_hyperparam_acc(model_scores_mlp))

mlp_lr_sp = float(best_hyperparam_sp(mlp_metrics))

best_mlp_acc = train_model("mlp", train, mlp_lr_acc)

best_mlp_sp = train_model("mlp", train, mlp_lr_sp)


for i in [1.0, 100000.0, 0.001, 0.000001, 0.000000001]:
    model_scores_lr[str(i)], lr_metrics[str(i)] = val_score(C_lr=i)
    model_scores_svm[str(i)], svm_metrics[str(i)] = val_score(C_svm=i)
# For learning rate with best accuracy
lr_C_acc = float(best_hyperparam_acc(model_scores_lr))
# For learning rate with least parity

lr_C_sp = float(best_hyperparam_sp(lr_metrics))
```

```
best_lr_acc = train_model("lr", train, lr_C_acc)
best_lr_sp = train_model("lr", train, lr_C_sp)

svm_C_acc = float(best_hyperparam_acc(model_scores_svm))

svm_C_sp = float(best_hyperparam_sp(svm_metrics))

best_svm_acc = train_model("svm", train, svm_C_acc)

best_svm_sp = train_model("svm", train, svm_C_sp)


def save_dict(dct, dct_name):
    pck.dump(dct, open(dct_name, "wb"))


def load_dict(dct_name):
    return pck.load(open(dct_name, "rb"))


save_dict(model_scores_lr, "model_scores_lr.pkl")
save_dict(model_scores_svm, "model_scores_svm.pkl")
save_dict(model_scores_mlp, "model_scores_nl.pkl")

# Testing saved results
print(load_dict("models_nl.pkl"))

mlp_dict = {
    "learning_rate": list(model_scores_mlp.keys()),
    "Average CV Accuracy": list(model_scores_mlp.values()),
    "Statistical parity Difference": list(mlp_metrics.values()),
}
svm_dict = {
    "C": list(model_scores_svm.keys()),
    "Average CV Accuracy": list(model_scores_svm.values()),
    "Statistical parity Difference": list(svm_metrics.values()),
}
lr_dict = {
    "C": list(model_scores_lr.keys()),
    "Average CV Accuracy": list(model_scores_lr.values()),
    "Statistical parity Difference": list(lr_metrics.values()),
}
model_perf = {
    "MLP": pd.DataFrame(mlp_dict),
    "SVM": pd.DataFrame(svm_dict),
    "LR": pd.DataFrame(lr_dict),
```

```
}
result_df = pd.concat(model_perf.values(), keys=model_perf.keys(), axis=1)
result_df.to_pickle("Cross_Val_Results.pkl")

mlp_acc_wise = {
    "Accuracy": [best_mlp_acc.evaluate(x_test, y_test, verbose=0)[1]],
    "Statistical Parity Difference": [
        test_metric(test, get_mlp_classes(test, best_mlp_acc))
    ],
}

mlp_sp_wise = {
    "Accuracy": [best_mlp_sp.evaluate(x_test, y_test, verbose=0)[1]],
    "Statistical Parity Difference": [
        test_metric(test, get_mlp_classes(test, best_mlp_sp))
    ],
}

lr_sp_wise = {
    "Accuracy": [best_lr_sp.score(x_test, y_test)],
    "Statistical Parity Difference": [test_metric(test, best_lr_sp.predict(x_test))],
}

lr_acc_wise = {
    "Accuracy": [best_lr_acc.score(x_test, y_test)],
    "Statistical Parity Difference": [test_metric(test, best_lr_acc.predict(x_test))],
}

svm_sp_wise = {
    "Accuracy": [best_svm_sp.score(x_test, y_test)],
    "Statistical Parity Difference": [test_metric(test, best_svm_sp.predict(x_test))],
}

svm_acc_wise = {
    "Accuracy": [best_svm_acc.score(x_test, y_test)],
    "Statistical Parity Difference": [test_metric(test, best_svm_acc.predict(x_test))],
}

model_perf_test = {
    "MLP Accuracy Wise": pd.DataFrame(mlp_acc_wise),
    "MLP Fairness Wise": pd.DataFrame(mlp_sp_wise),
    "SVM Accuracy Wise": pd.DataFrame(svm_acc_wise),
    "SVM Fairness Wise": pd.DataFrame(svm_sp_wise),
    "LR Accuracy Wise": pd.DataFrame(lr_acc_wise),
    "LR Fairness Wise": pd.DataFrame(lr_sp_wise),
}
```

```
test_result = pd.concat(model_perf_test.values(), keys=model_perf_test.keys(), axis=1)

test_result.to_pickle("models_testset.pkl")
print("Cross Val result: \n")
print(result_df)
print("Tets Set Result: \n")
print(test_result)
# params = {"optimizer__learning_rate": learning_rates, "loss": ["binary_crossentropy"]}
# model = KerasClassifier(
#     model=create_model,
#     epochs=100,
#     batch_size=100,
#     hidden_layer_sizes=(10,),
#     optimizer="adam",
#     optimizer__learning_rate=0.001,
# )
#
# print(cross_val_score(model, x_train, y_train, cv=5))
# clf = LogisticRegression()
# grid = GridSearchCV(model, params, refit=True, cv=5)
# grid.fit(x_train, y_train)
```