graphs

June 19, 2025

[ ]:
```
#%%

import os
import sys

sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))


import numpy as np
import pandas as pd
import torch
from matplotlib import pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.model_selection import GridSearchCV, train_test_split

from neural_net.preprocessing import prepare_data
from utils.graphs import compare, compare_fairness, compare_for_one_model
from utils.neural_utils import NeuralNetwork, predict

# sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))



X_train_data = pd.read_csv("../law_data.csv")
y_train_data = X_train_data.pop("first_pf")

X_train_data = pd.get_dummies(data=X_train_data)
X_train, X_test, y_train, y_test = train_test_split(X_train_data, y_train_data,␣
 ↪test_size=0.3, random_state=42)
```

[ ]:
```
#%%

neural_network = NeuralNetwork(input_size=X_train.shape[1])
# neural_network.load_state_dict(torch.load('neural_net/model.pth'))
```

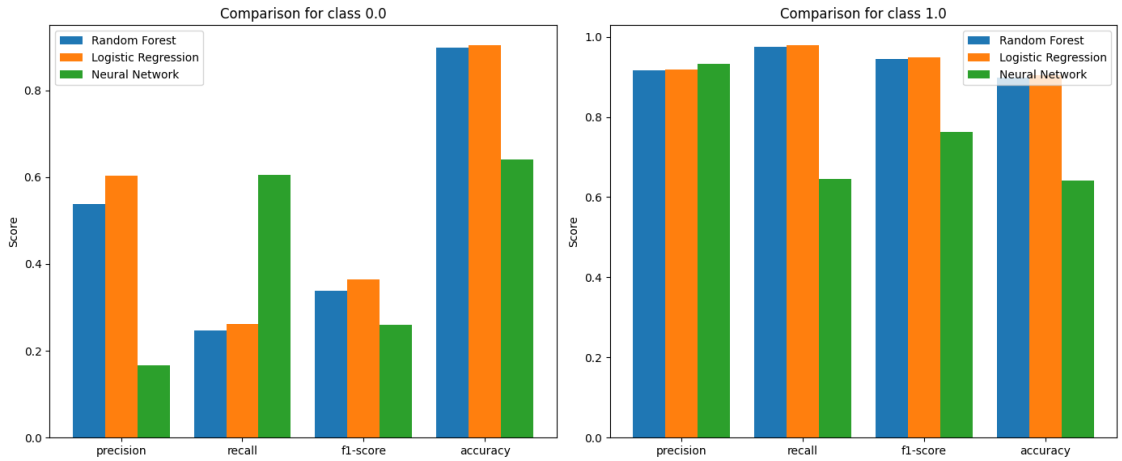
[ ]:
```
#%%

model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

regressor = LogisticRegression(max_iter=1000, random_state=42)
regressor.fit(X_train, y_train)
```

```
/home/tristan/Software/Development/Git/stage-2nde-inria/env/lib/python3.12/site-
packages/sklearn/linear_model/_logistic.py:470: ConvergenceWarning: lbfgs failed
to converge after 1000 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT

Increase the number of iterations to improve the convergence (max_iter=1000).
You might also want to scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[ ]:
```
LogisticRegression(max_iter=1000, random_state=42)
```

[ ]:
```
#%%

predictions = model.predict(X_test)
predictions_reg = regressor.predict(X_test)
predictions_nn = predict(neural_network, prepare_data(X_test))
print(classification_report(y_test, predictions_nn))

compare(
    [classification_report(y_test, predictions, output_dict=True),
        classification_report(y_test, predictions_reg, output_dict=True),
        classification_report(y_test, predictions_nn, output_dict=True)
        ],
        model_names=["Random Forest", "Logistic Regression", "Neural Network"],
)
```

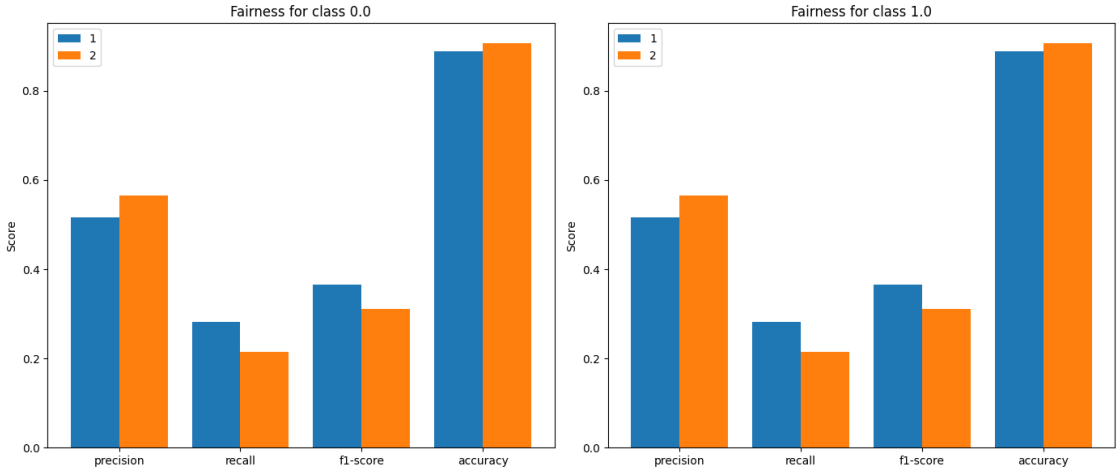
```
              precision    recall  f1-score   support

         0.0       0.17      0.60      0.26       685
         1.0       0.93      0.65      0.76      5853

    accuracy                           0.64      6538
   macro avg       0.55      0.62      0.51      6538
weighted avg       0.85      0.64      0.71      6538

['0.0', '1.0']
```



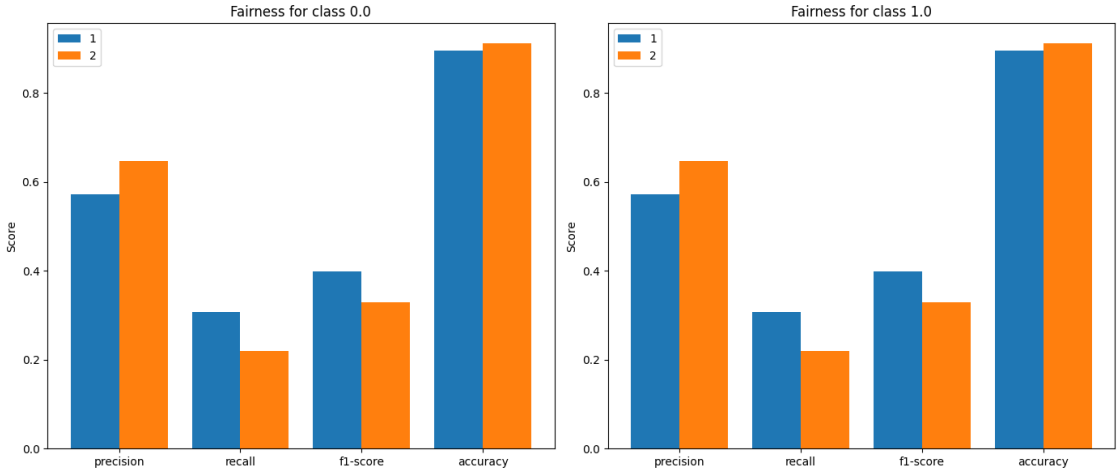
[ ]:
```
# %%

# Prediction per sex
results_sex = {
    "rf": {},
    "reg": {},
    "nn": {}
}

sex = X_test.groupby("sex")
for name, groups in sex:
    pred_rf = model.predict(groups)
    pred_reg = regressor.predict(groups)
    pred_nn = predict(neural_network, prepare_data(groups))

    results_sex["rf"][name] = classification_report(y_test.loc[groups.index],␣
 ↪pred_rf, output_dict=True)
    results_sex["reg"][name] = classification_report(y_test.loc[groups.index],␣
 ↪pred_reg, output_dict=True)
    results_sex["nn"][name] = classification_report(y_test.loc[groups.index],␣
 ↪pred_nn, output_dict=True)

    print("\n", name, groups.shape[0])
    # compare(
    #     [classification_report(y_test.loc[groups.index], pred_rf,␣
 ↪output_dict=True),
    #     classification_report(y_test.loc[groups.index], pred_reg,␣
 ↪output_dict=True),
    #     classification_report(y_test.loc[groups.index], pred_nn,␣
 ↪output_dict=True)],
```

```
    #     model_names=["Random Forest", "Logistic Regression", "Neural␣
 ↪Network"],
    #     label="Men" if name == 1 else "Women"
    # )


for model_name, results in results_sex.items():
    print(f"\nResults for {model_name}:")
    compare_fairness(results)
```

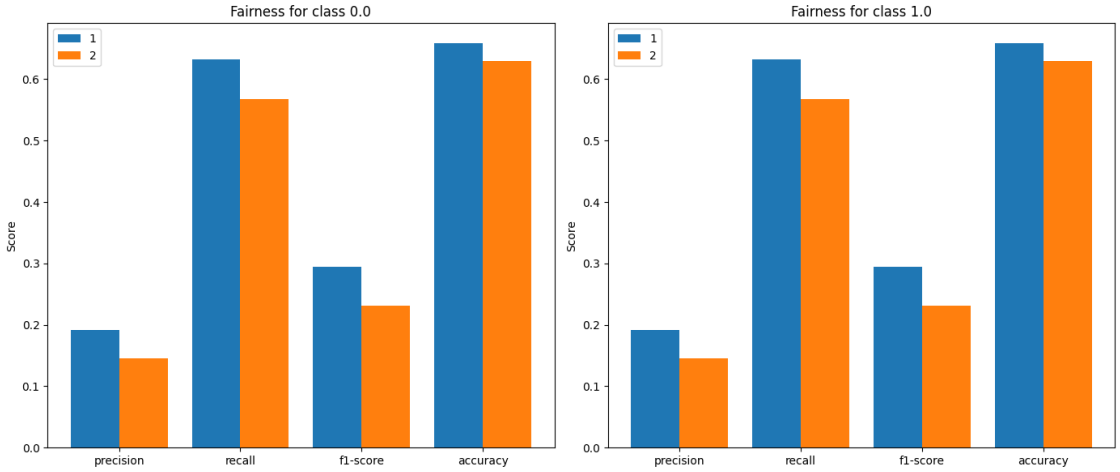
```
1 2894

2 3644

Results for rf:
```




```
Results for reg:
```



Results for nn:



[ ]:
```
# %%

# Prediction per ethnicity
ethnicities = ["Amerindian", "Asian", "Black", "Hispanic", "Mexican", "Other",␣
 ↪"Puertorican", "White"]

results_ethicities = {
    "rf": {},
    "reg": {},
    "nn": {}
}
```

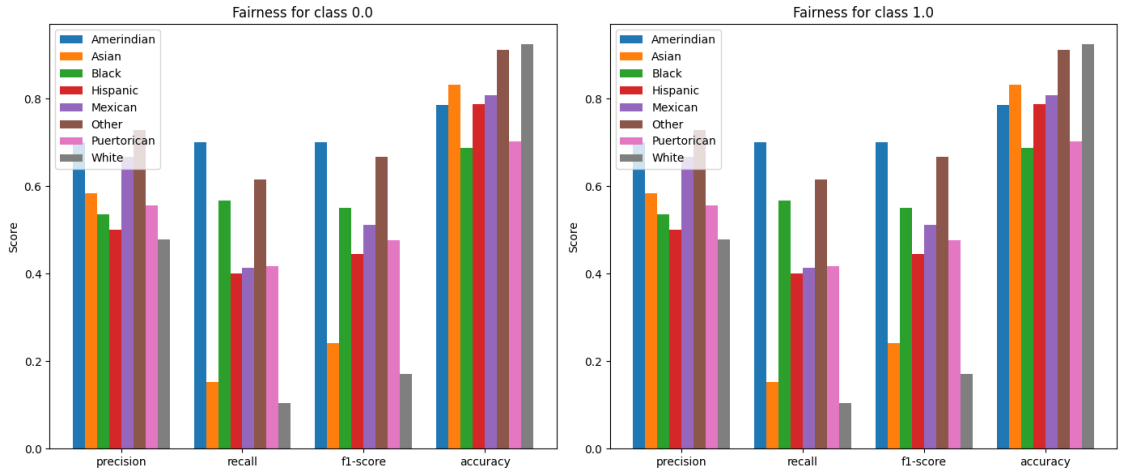
```
for ethnicity in ethnicities:
    group = X_test.groupby("race_"+ethnicity)
    for name, groups in group:
        if name == True:
            pred_rf = model.predict(groups)
            pred_reg = regressor.predict(groups)
            pred_nn = predict(neural_network, prepare_data(groups))

            results_ethicities["rf"][ethnicity] = classification_report(y_test.
 ↪loc[groups.index], pred_rf, output_dict=True)
            results_ethicities["reg"][ethnicity] = classification_report(y_test.
 ↪loc[groups.index], pred_reg, output_dict=True)
            results_ethicities["nn"][ethnicity] = classification_report(y_test.
 ↪loc[groups.index], pred_nn, output_dict=True)

            print("\n", ethnicity, groups.shape[0])

            # compare(
            #     [classification_report(y_test.loc[groups.index], pred_rf,␣
 ↪output_dict=True),
            #     classification_report(y_test.loc[groups.index], pred_reg,␣
 ↪output_dict=True),
            #     classification_report(y_test.loc[groups.index], pred_nn,␣
 ↪output_dict=True)],
            #     model_names=["Random Forest", "Logistic Regression", "Neural␣
 ↪Network"],
            #     label = ethnicity
            # )

for model_name, results in results_ethicities.items():
    print(f"\nResults for {model_name}:")
    compare_fairness(results)
```

```
Amerindian 28

Asian 261

Black 402

Hispanic 118

Mexican 120

Other 90
```

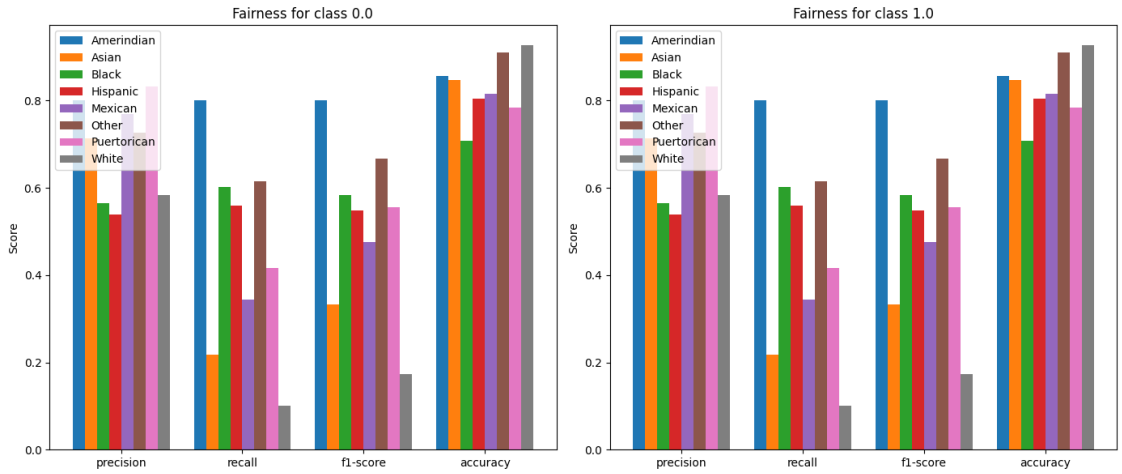
Puertorican 37

White 5482

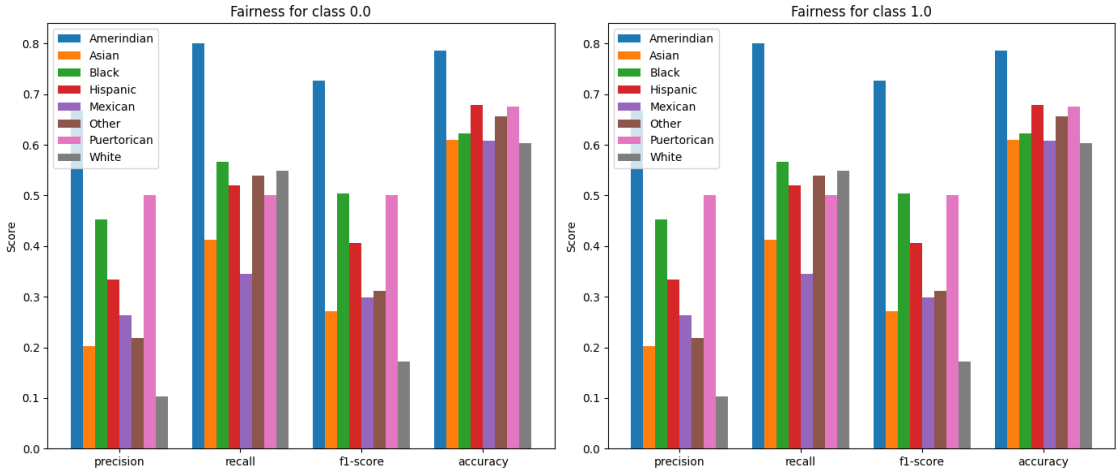
Results for rf:



Results for reg:



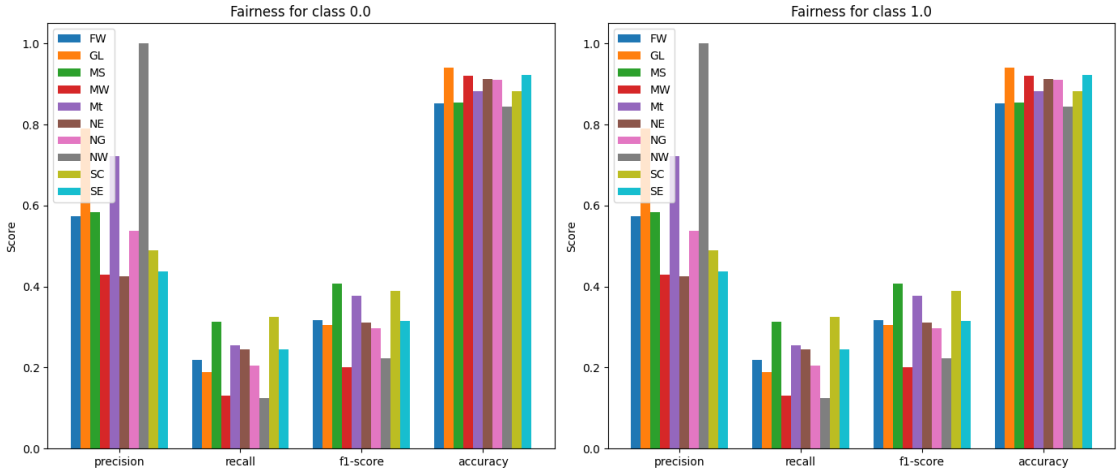
Results for nn:



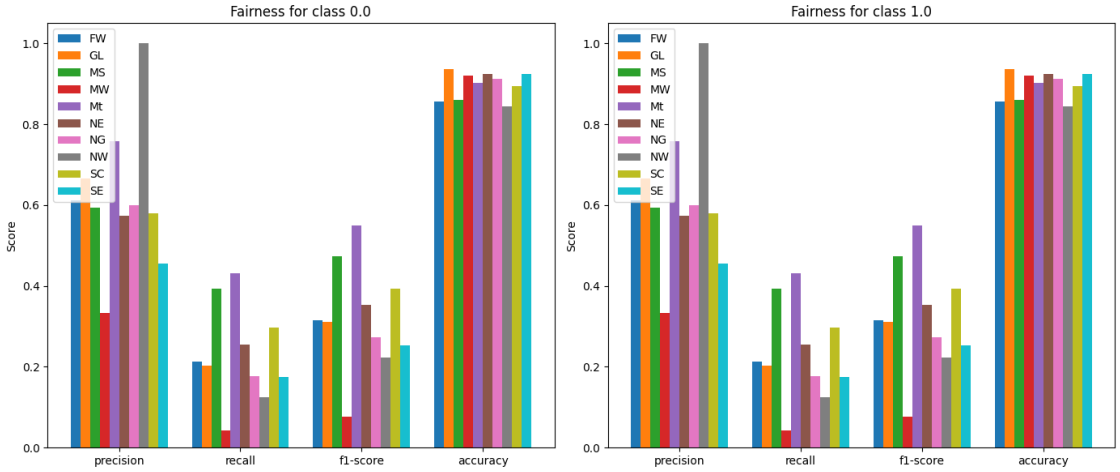
[ ]:
```
# %%

# Prediction per region
results_regions = {
    "rf": {},
    "reg": {},
    "nn": {}
}

regions = ["FW","GL","MS","MW","Mt","NE","NG","NW","PO","SC","SE"]
for region in regions:
    group = X_test.groupby("region_first_"+region)
    for name, groups in group:
        if name == True:
            pred = model.predict(groups)
            pred_reg = regressor.predict(groups)
            pred_nn = predict(neural_network, prepare_data(groups))

            results_regions["rf"][region] = classification_report(y_test.
 ↪loc[groups.index], pred, output_dict=True)
            results_regions["reg"][region] = classification_report(y_test.
 ↪loc[groups.index], pred_reg, output_dict=True)
            results_regions["nn"][region] = classification_report(y_test.
 ↪loc[groups.index], pred_nn, output_dict=True)

            print("\n", region, groups.shape[0])

            # compare(
            #     [classification_report(y_test.loc[groups.index], pred,␣
 ↪output_dict=True),
```

```
            #     classification_report(y_test.loc[groups.index], pred_reg,␣
 ↪output_dict=True),
            #     classification_report(y_test.loc[groups.index], pred_nn,␣
 ↪output_dict=True)],
            #     model_names=["Random Forest", "Logistic Regression", "Neural␣
 ↪NetworkS"],
            #     label = region
            # )

for model_name, results in results_regions.items():
    print(f"\nResults for {model_name}:")
    compare_fairness(results)
```

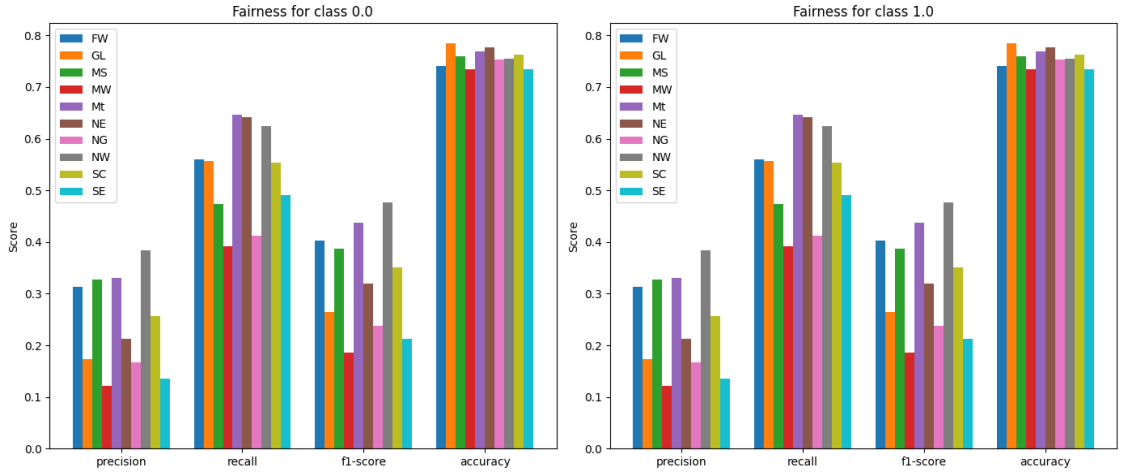
```
 FW 905

 GL 1131

 MS 701

 MW 298

 Mt 367

 NE 1300

 NG 365

 NW 45

 SC 642

 SE 784

Results for rf:
```



Results for reg:



Results for nn:



[ ]:
```
# %%

#Grid search cv
# parameters = {"n_estimators": [100, 250, 500, 1000, 2000], "max_depth": [10,␣
 ↪20, 30, 40, 50, 60, 70, 80, 90, 100]}
# rf_model = RandomForestClassifier(random_state=1)
# grid_search = GridSearchCV(rf_model, parameters, cv=5, n_jobs=-1,␣
 ↪return_train_score=True, verbose=3)
# grid_search.fit(X_train_data, y_train_data)
```

[ ]:
```
# %%

def export_model():
    return model, regressor
```