Assignment 2:
Responsible AI, assessing and applying fairness and explainability in ML

nr:837377516
Wilfred Van Casteren
Open Universiteit
Masteropleiding AI
Responsible AI

May 19, 2024

Abstract

In this writing I will get and give an insight in Fairlearn, a library used to assess fairness and mitigate fairness problems. Furthermore I will check out LIME and SHAP, two libraries that give an insight in the features that had the biggest influence on a single prediction or multiple predictions. During these experiments I haven't focussed on performance, of course some sort of accuracy had to be reached, before trying to assess fairness or explainability.

Contents

Chapter 1

Fairness

1.1 Fairness: A sociotechnical challenge

To get an idea of the concept of fairness when dealing with AI systems I have extensively studied a tutorial Assessing and mitgating unfairness in AI Systems and read, studied and/or watched other tutorials on the subject as well. All of the tutorials I focused on, used Fairlearn as the Python package to assess fairness and mitigate bias. I have studied the python notebook extensively and will reproduce the key takeaways of this notebook here and most of the code.

Before arriving there, I have studied An Introduction to Fairness in Machine Learning using Fairlearn, A deep dive into Fairness in Machine Learning using Fairlearn.

Even though one can often spot fairness-related harms when one sees them, there's no one-size-fits-all definition of fairness that applies to all AI systems in all contexts. Fairness in AI is a fundamentally sociotechnical challenge, so it cannot be approached from purely social or technical perspectives. This can make it a particularly daunting landscape to navigate. But, although there are few easy answers, there are a variety of emerging strategies and tools for assessing and mitigating fairness-related harms, as well as a deepening understand of the challenge throughout society.

One point that needs to be emphasized is that it is not possible to fully "debias" an AI system or to guarantee its fairness. This is because there are so many reasons why AI systems can cause fairness-related harms. As a result, it is not recommended to use words like debiasing and unbiased, which can set up unrealistic expectations. Instead, one should be talking about assessing and mitigating fairness-related harms.

1.1.1 Fairness-related harms

Broadly speaking, there are five main types of fairness-related harms that can be caused by an AI system:

- Allocation harms: Does the system allocate opportunities or resources in a way that negatively impacts a group of people?
- Quality of service harms: These harms focus on whether an AI system works as well for one person as it does for another.
- Stereotyping harms: These harms occur when a system promotes or reinforces harmful or negative stereotypes about a group of people.
- Demeaning harm: The fourth type of harm is when an AI system is actively demeaning.
- Over- or under-representation: Does the system over-represent or under-represent certain groups of people, or possibly erase some groups of people entirely?

Harms are not mutually exclusive These harms are neither mutually exclusive. Any single AI system can exhibit multiple types of fairness-related harms and can exhibit different harms for different groups of people.

1.2 Introduction to the health care scenario

This complete chapter is considering an automated system for recommending patients for high-risk care management programs. These programs seek to improve the care of patients with complex health needs by providing additional resources, including greater attention from trained providers, to help ensure that care is well coordinated. Most health systems use these programs as the cornerstone of population health management efforts, and they are widely considered effective at improving outcomes and satisfaction while reducing costs. Because the programs are themselves expensive —with costs going toward teams of dedicated nurses, extra primary care appointment slots, and other scarce resources— health systems rely extensively on algorithms to identify patients who will benefit the most. In practice, the modeling of health needs would use large data sets covering a wide range of diagnoses. In this writing, a publicly available clinical dataset that focuses on diabetic patients only will be used.

1.2.1 Dataset and task

A clincial dataset of hospital re-admissions over a ten-year period (1998-2008) for diabetic patients across 130 different hospitals in the US will be used. Each record represents the hospital admission records for a patient diagnosed with diabetes whose stay lasted one to fourteen days.

The features describing each encounter include demographics, diagnoses, diabetic medications, number of visits in the year preceding the encounter, and payer information, as well as whether the patient was readmitted after release, and whether the readmission occurred within 30 days of the release. See also Table 1.1.

Features	**Description**
race, gender, age	demographic features
medicare, medicaid	insurance information
admission_source_id	emergency, referral, or other
had_emergency, had_inpatient_days, had_outpatient_days	hospital visits in prior year
medical_specialty	admitting physician's specialty
time_in_hospital, num_lab_procedures, num_procedures, num_medications, primary_diagnosis, number_diagnoses, max_glu_serum, A1Cresult, insulin	description of the hospital visit
discharge_disposition_id	discharged to home or not
readmitted, readmit_binary, readmit_30_days	readmission information

Table 1.1: Description of Features

A classification model, which decides whether the patients should be suggested to their primary care physicians for an enrollment into the high-risk care management program, will be developed. The positive prediction will mean recommendation into the care program.

Decision point: Task definition

- A hospital readmission within 30 days can be viewed as a proxy that the patients needed more assistance at their release time, so it will be the label to be predicted.
- Because of the class imbalance, our performance will be measured via balanced accuracy. Another key performance consideration is how many patients are recommended for care, a metric referred to as selection rate.

Fairness considerations

- Which groups are most likely to be disproportionately negatively affected? Previous work suggests that groups with different race and ethnicity can be differently affected. To make things a little bit more difficult I will also consider gender as a feature that affects treatment. By no way I am implying that people of various genders and ethnicity receive different treatment.
- What are the harms? The key harms here are allocation harms. In particular, false negatives, i.e., don't recommend somebody who will be readmitted.
- How should we measure those harms?

Code to get the data

Listing 1.1: Get data

```
df = pd.read_csv(
"https://raw.githubusercontent.com/fairlearn/talks/main/2021_scipy_tutorial/data/
    ↪ diabetic_preprocessed.csv")
print(df.head())

df = df.drop_duplicates(keep='first')

df = df.drop(columns=[
"discharge_disposition_id",
"readmitted"
])

df = df.drop(df[df['gender'] == 'Unknown/Invalid'].index)
df["race"] = df["race"].replace({"Asian": "Other", "Hispanic": "Other"})
df["diabetesMed"] = df["diabetesMed"].replace({"Yes": 1, "No": 0})
df["A1Cresult"] =  df["A1Cresult"].fillna("NotTested")
df["max_glu_serum"] =  df["max_glu_serum"].fillna("NotTested")


for col_name in categorical_features():
 df[col_name] = df[col_name].astype("category")
return df
```

1.3 Dataset and its sensitive groups

Things already done Before progressing, it should be mentioned some work is already done:

- merge the three smallest race groups Asian, Hispanic, Other, but also retain the original groups for auxiliary assessments
- drop the gender group Unknown/Invalid, because the sample size is so small that no meaningful fairness assessment is possible

As a matter of fact this dataset has undergone some form of preprocessing, but let's not take this into consideration for this exercise.

1.3.1 Some figures

Sensitive features For this case study gender and race will be considered being sensitive features. To get an idea of the different groups of people represented in the dataset, some graphs were produced. For a graph showing different counts for gender, see Figure 1.1. For a graph showing different counts for gender, see Figure 1.2. As can be seen

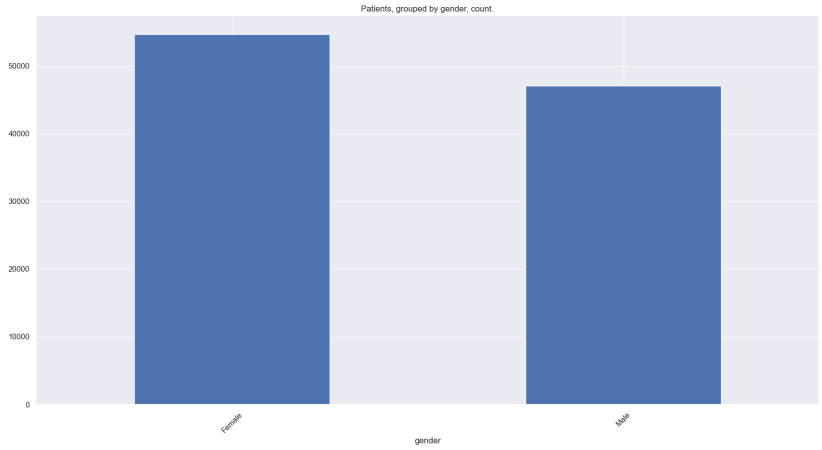

Figure 1.1: Patients, grouped by gender, counted.

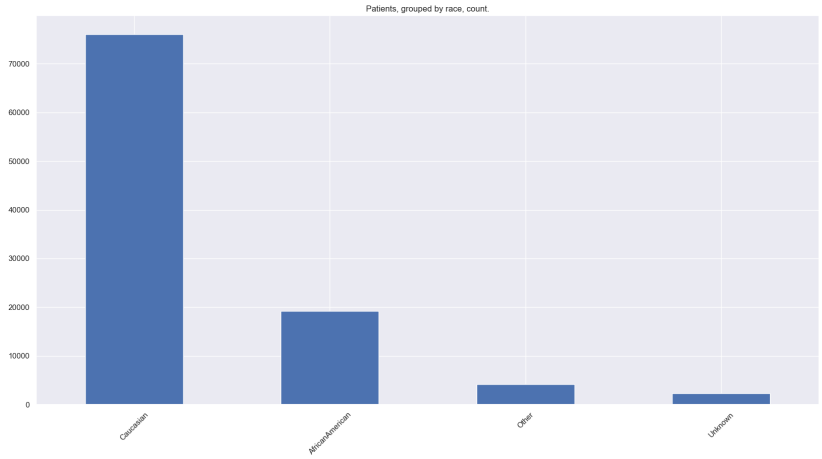

Figure 1.2: Patients, grouped by race, counted.

As can be seen from the graphs, a lot of variety is present, large differences in group size, and small group sizes. Small sample sizes have two implications:

- assessment: the impacts of the AI system on smaller groups are harder to assess, because due to fewer data points a much larger uncertainty (error bars) in our estimates can be seen
- model training: fewer training data points mean that our model fails to appropriately capture any data patterns specific to smaller groups, which means that its predictive performance on these groups could be worse

The target value Combining the target value readmit_30_days with the sensitive features under consideration, even smaller numbers, and bigger differences become apparent, see Figure 1.3. As numbers for various groups aren't big enough to be significant, it will be very difficult to guarantee fairness. Actually some more data should be collected.

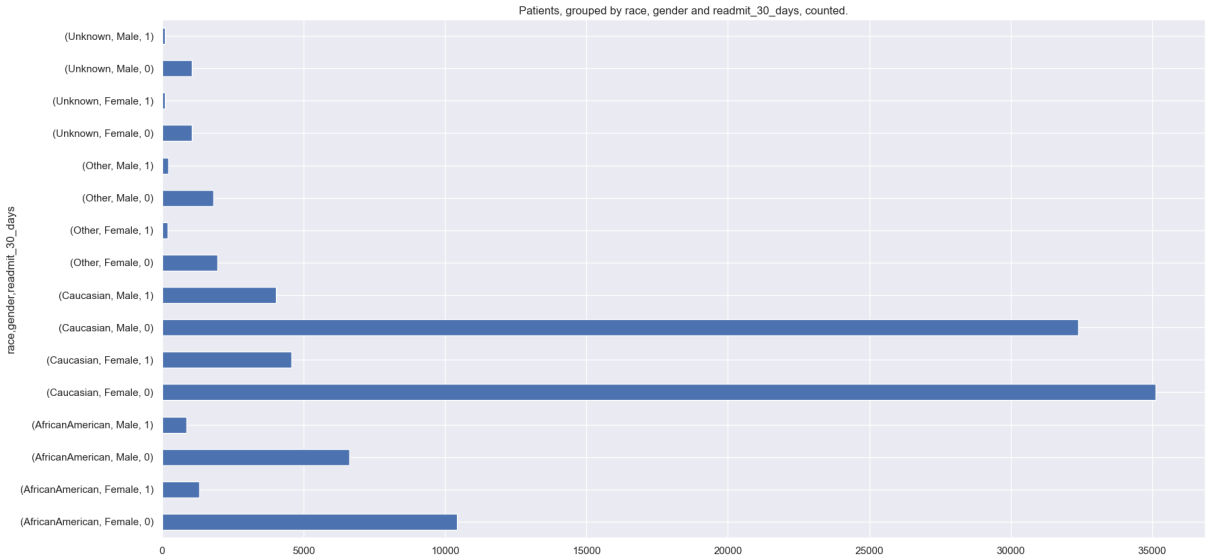

Figure 1.3: Patients, grouped by gender, race and readmit_30_days, counted.

Decision point: How to address smaller group sizes? When the data set lacks coverage of certain groups, it means that it will be hard to impossible to reliably assess any fairness-related issues. There are three interventions (which could be carried out in a combination):

- collect more data: collect more data for groups with fewer samples
- buckets: merge some of the groups
- drop small groups

The choice of strategy depends on our existing understanding of which groups are at the greatest risk of a harm. In particular, pooling the groups with widely different risks could mask the extent of harms. Dropping small groups leads to the representational harm of erasure.

If any groups are merged or dropped, these decisions should be annotated / explained (in the datasheet, which we discuss below).

Code for this part

To produce the graphs see code in Listing 1.3

Listing 1.2: Sensitive variables counts

```
def show_counts_sensitive_variables(df):
 plt.rcParams["figure.figsize"] = (20, 10)
 df["race"].value_counts().plot(kind='bar', rot=45, title='Patients, grouped by race
    ↪ , count.')
 plt.savefig('./generated/' + 'race_counted.png')
 plt.show()
 df.gender.value_counts().plot(kind='bar', rot=45, title='Patients, grouped by
    ↪ gender, count.')
 plt.savefig('./generated/' + 'gender_counted.png')
 plt.show()
 dfg = df.groupby(by=["race", "gender", "readmit_30_days"]).size()
 dfg.plot(kind='barh', title='Patients, grouped by race, gender and readmit_30_days,
    ↪  counted.')
 plt.savefig('./generated/' + 'patients_grouped_counted.png')
 plt.show()
```

1.3.2 Examining the choice of label

In this section the question of whether our choice of label (readmission within 30 days) aligns with our goal (identify patients that would benefit from the care management program).

A framework particularly suited for this analysis is called measurement modeling. The goal of measurement modeling is to describe the relationship between what is cared about and what can be measured. The thing that is cared about is referred to as the construct and what is to be observed is referred to as the measurement. In our case:

- construct = greatest benefit from the care management program
- measurement = readmission within 30 days (in the absence of such program). In this case, the measurement coincides with the classification label

The act of operationalizing the construct via a specific measurement corresponds to making certain assumptions. In this case, the following assumption is made: the greatest benefit from the care management program would go to patients that are (in the absence of such a program) most likely to be readmitted into the hospital within 30 days.

1.3.3 Checking whether our assumptions apply?

Establishing construct validity means demonstrating, in a variety of ways, that the measurements obtained from measurement model are both meaningful and useful:

- Does the operationalization capture all relevant aspects of the construct purported to be measured?
- Do the measurements look plausible?
- Do they correlate with other measurements of the same construct? Or do they vary in ways that suggest that the operationalization may be inadvertently capturing aspects of other constructs?
- Are the measurements predictive of measurements of any relevant observable properties (and other unobservable theoretical constructs) thought to be related to the construct, but not incorporated into the operationalization?

Predictive validity, an aspect of construct validity, will be focussed on. Predictive validity which refers to the extent to which the measurements obtained from a measurement model are predictive of measurements of any relevant observable properties related to the construct purported to be measured, but not incorporated into the operationalization.

The predictions do not need to be chronological, meaning that we do not necessarily need to be predicting future from the past. Also, the predictions do not need to be causal (going from causes to effects). It has to be ensured the predicted property is not part of the measurement whose validity we're checking.

Predictive validity The purpose here is to show that our measurement readmit_30_days is correlated with patient characteristics that are related to our construct "benefiting from care management". One such characteristic is the general patient health, where it is expected that patients that are less healthy are more likely to benefit from care management.

While the data does not contain full health records that would make it possible to holistically measure general patient health, the data does contain two relevant features: had_emergency and had_inpatient_days, which indicate whether the patient spent any days in the emergency room or in the hospital (but non-emergency) in the preceding year.

Predictive validity between readmit_30_days and had_emergency

To establish predictive validity, from Figure 1.4 it can be established readmit_30_days is predictive of had_emergency. When combined with a sensitive variables like race, things get less obvious, see Figure 1.5.

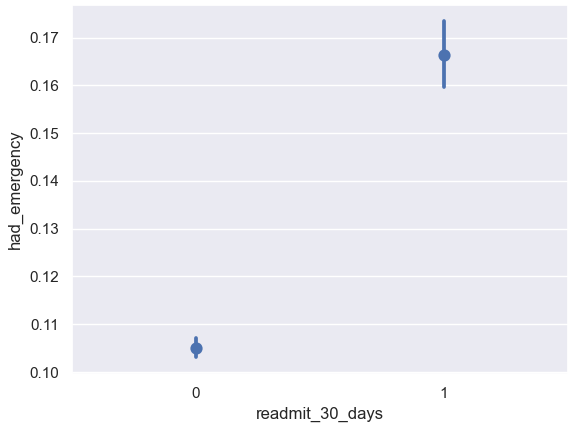

Figure 1.4: Difference in correlation between had_emergency and readmit_30_days, with a confidence interval of 95%.

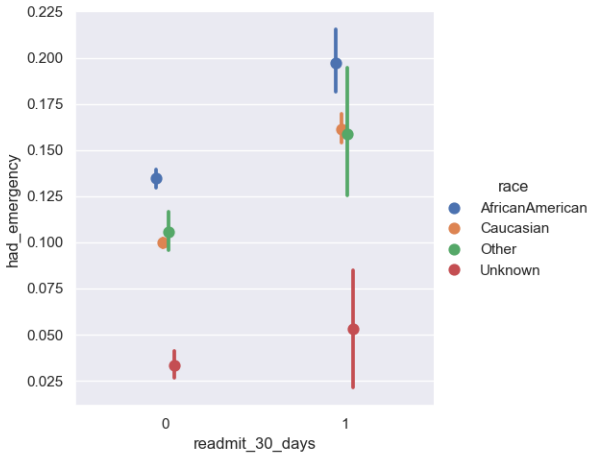

Figure 1.5: Difference in correlation between had_emergency and readmit_30_days, per race category, with a confidence interval of 95%.

Predictive validity between readmit_30_days and had_inpatient_days

When trying to establish predictive validity, from Figure 1.4 can be established readmit_30_days is predictive of had_emergency. When combined with a sensitive variables like race, things get less obvious, see Figure 1.5.

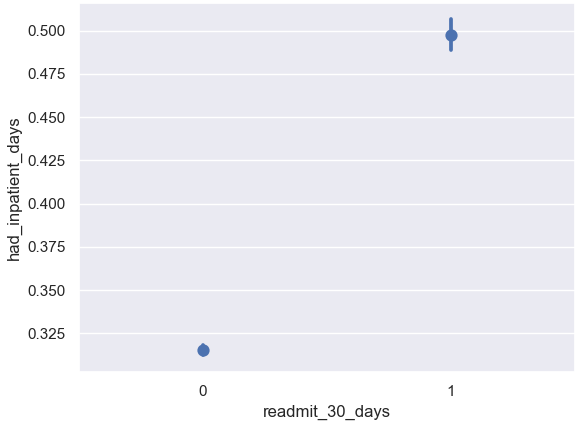

Figure 1.6: Difference in correlation between readmit_30_days and had_inpatient_days, with a confidence interval of 95%.

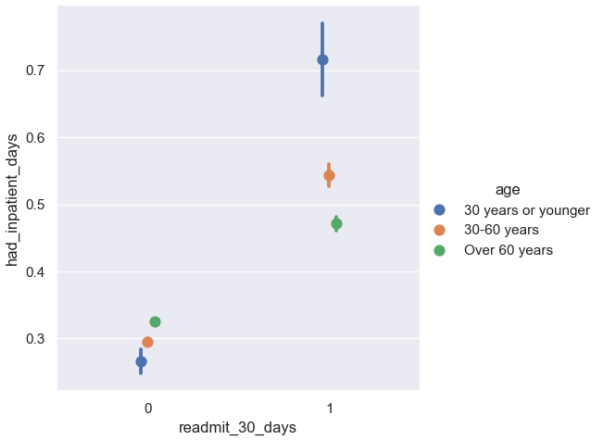

Figure 1.7: Difference in correlation between readmit_30_days and had_inpatient_days, per age category, with a confidence interval of 95%.

Similar graphs could of course be made using combinations of other features and sensitive variables.

1.3.4 Label imbalance

The frequency of different labels is an important descriptive characteristic in classification settings for several reasons:

- some classification algorithms and performance measures might not work well with datasets with extreme class imbalance
- in binary classification settings, the ability to evaluate error is often driven by the size of the smaller of the two classes (again, the smaller the sample the larger the uncertainty in estimates)
- label imbalance may exacerbate the problems due to smaller group sizes in fairness assessment

As can be seen in Figure 1.8, the target label is heavily skewed towards the patients not being readmitted within 30 days. In our dataset, only 11% of patients were readmitted within 30 days.

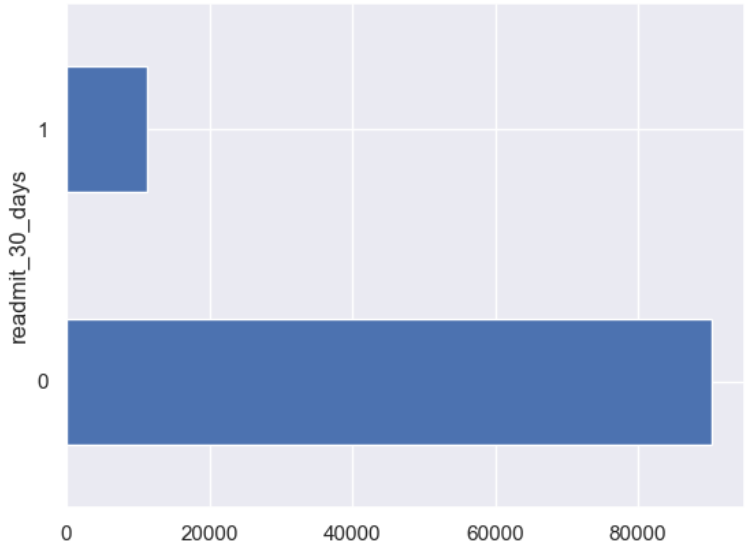

Figure 1.8: Number of patients tha was readmitted_30_days.

Since there are fewer positive examples, much larger uncertainty (error bars) in our estimates of false negative rates(FNR) are expected compared with false positive rates (FPR). This means that there will be larger differences between training FNR and test FNR, even if there is no overfitting, simply because of the smaller sample sizes.

The target metric will be the balanced error rate, which is the average of FPR and FNR. The value of this metric is robust to different frequencies of positives and negatives. However, since half of the metric is contributed by FNR, the uncertainty in balanced error values are expected to behave similarly to the uncertainty of FNR.

To complicate things some more, in Figure 1.9 a graph to get an idea for the imbalance for every combination of the sensitive variables.

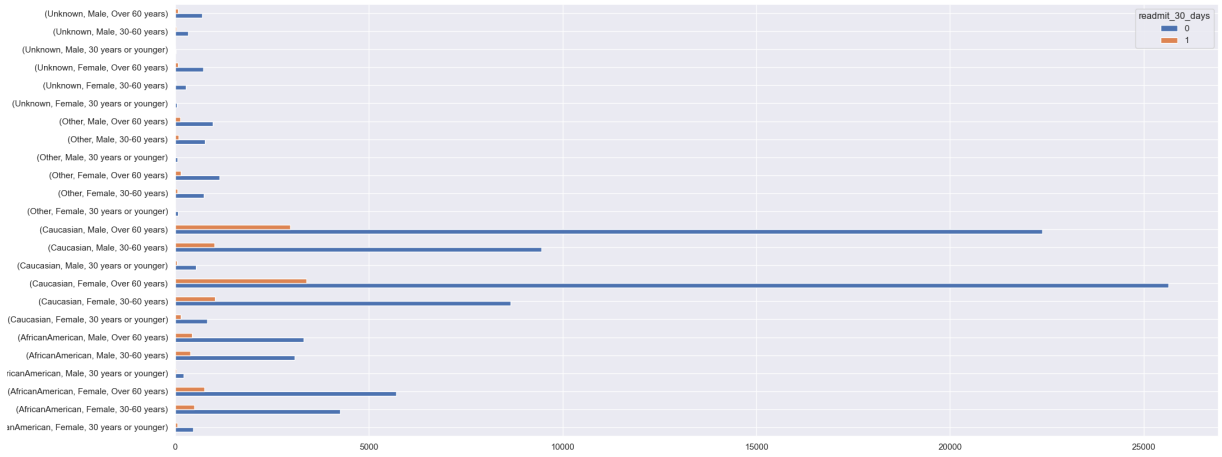

Figure 1.9: Number of patients that was readmitted_30_days, grouped by race, gender and age.

Listing 1.3: Sensitive variables and target counts

```
to_be_grouped = df[["gender", "race", "age", "readmit_30_days"]].copy()
to_be_grouped.groupby(['race', 'gender', 'age'], observed=False).readmit_30_days.
    ↪ value_counts().unstack(3).plot.barh(figsize=(24, 10))
plt.savefig(label_imbalance_dir() + 'li_race_gender_age.png')
if show:
 plt.show()
```

1.3.5 Proxies for sensitive features

In this section which of the features are highly predictive of the sensitive features race and gender. Such predictive features are called proxies.

The main purpose of this chapter is to assess the impact of the machine-learning model on different populations. But there are other concepts of fairness that seek to analyze how the model might be using information contained in the sensitive features. More pragmatically, certain uses of sensitive features (or proxies of it) might be illegal in some contexts.

Another reason to understand the proxies is because they might explain why there are differences in impact on different groups even when our model does not have access to the sensitive features directly.

In this section we briefly examine the identification of such proxies (not going into legal or causality considerations).

In the United States, Medicare and Medicaid are joint federal and state programs to help qualified individuals pay for healthcare expenses. Medicare is available to people over the age of 65 and younger individuals with severe illnesses. Medicaid is available to all individuals under the age of 65 whose adjusted gross income falls below the Federal Poverty Line.

Because Medicaid is available to low-income individuals, and race is correlated with socioeconomic status in the United States, a relationship between race and paying with Medicaid is expected, see Figure 1.10.

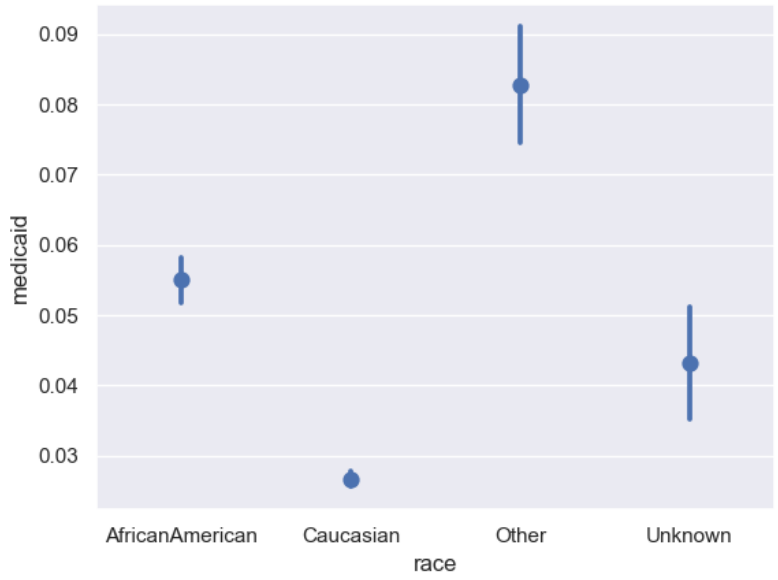

Figure 1.10: Correlation between race and medicaid.

To look at these kind of problem from a more general standpoint, in Figure 1.11 a table can be seen that contains the pairwise correlations between every feature and race.

	gender_Female	gender_Male	age_30 years or younger	age_30-60 years	age_Over 60 years	admission_source_id_Emergency	admission_source_id_Other
race_AfricanAmerican	0.07054487825193247	-0.07054487825193176	0.047182618180964093	0.1332917356046337	-0.1460878898512937	0.061244473373411705	-0.037896837138682395
race_Caucasian	-0.05545927181493135	0.05545927181493087	-0.04775159253935222	-0.1380499934351288	0.1509341049538551	-0.050018999478619215	0.022368614104615086
race_Other	-0.00901307095399732	0.009013070953997076	0.01144810164560649	0.042310634394086236	-0.04520547698860371	0.030927599094659468	-0.02391625805898194
race_Unknown	-0.011727885657968683	0.011727885657968591	3.32621268732415e-06	-0.004126694468165937	0.004038694871678817	-0.0567660258122372	0.06678037322031195

Figure 1.11: Correlation for race category and every other feature(limited to fit the width).

To produce such a limited table code in Listing 1.4 was used.

Listing 1.4: Sensitive variables and correlations for all other features

```
cor_df = pd.DataFrame()
df_ = df.copy()

filter_col = [col for col in df_ if col.startswith('race')]
for col in filter_col:
 corr_col_ = df_[df_.columns[0:]].corr()[col]
 new_list = filter_col.copy()
 cor_col = corr_col_.to_frame(col).T.copy()
 cor_col = cor_col.drop(columns=new_list, axis=1)
 cor_df = pd.concat([cor_df, cor_col.copy()], axis=0)
cor_df.to_csv(correlation_dir() +'corr_race.csv')
```

1.4 Training the initial model

In this section a classificaion model will be trained to predict the target variable (readmission within 30 days) while optimizing on balanced accuracy.

1.4.1 How to choose the classification model

When choosing between models (neural nets, decision tree, logistic regression) there are a variety of considerations.

Interpretability Interpretability is tightly linked with questions of fairness. There are several reasons why it is important to have interpretable models that are open to the stakeholder scrutiny:

- It allows discovery of fairness issues that were not discovered by the data science team.
- It provides a path toward recourse for those that are affected by the model.
- It allows for a face validity check, a ”sniff test”, by experts to verify that the model ”makes sense” (at the face value). While this step is subjective, it is really important when the model is applied to different populations than those on which the assessment was conducted.

Model expressiveness How well can the model separate positive examples from negative examples? How well can it do so given the available dataset size? Can it do so across all groups or does it need to trade off performance on one group against performance on another group?

Some additional considerations are training time (this impacts the ability to iterate), familiarity (this impacts the ability to fine tune and debug), and carbon footprint (this impacts the Earth climate both directly and indirectly by normalizing unnecessarily heavy workloads).

Decision point: Model type A logistic regression model will be used

- Interpretability. Logistic models over a small number of variables (as used here) are highly interpretable in the sense that stakeholders can simulate them easily.
- Model expressiveness. Logistic regression predictions are described by a linear weighting of the feature values. The concern might be that this is too simple.

1.4.2 Prepare test and train datasets

As already mentioned in the task definition, the target variable is readmission within 30 days, and the sensitive features for the purposes of fairness assessment is race and gender. The code to create the datasets is in Listing 1.5

Listing 1.5: Prepare test and train datasets

```
def prepare_test_train_datasets(df, random_seed):
 target_variable = "readmit_30_days"
 sensitive = ["race", "gender"]
 Y, A = df.loc[:, target_variable], df.loc[:, sensitive]
 X = pd.get_dummies(df.drop(columns=["race", "gender", "readmit_binary", "
   ↪ readmit_30_days"]))
 X_train, X_test, Y_train, Y_test, A_train, A_test, df_train, df_test =
   ↪ train_test_split(X,Y,A,df,test_size=0.50,stratify=Y,random_state=random_seed)
 return X_train, X_test, Y_train, Y_test, A_train, A_test, df_train, df_test
```

Balanced accuracy The performance metric is balanced accuracy, so for the purposes of training (but not evaluation!) the data set will be resampled, so that it has the same number of positive and negative examples. This means that we can use estimators that optimize standard accuracy (although some estimators allow the use of importance weights).

Listing 1.6: Balanced accuracy

```
def resample_dataset(X_train, Y_train, A_train):
 negative_ids = Y_train[Y_train == 0].index
 positive_ids = Y_train[Y_train == 1].index
 balanced_ids = positive_ids.union(np.random.choice(a=negative_ids, size=len(
   ↪ positive_ids)))

```

```
    X_train = X_train.loc[balanced_ids, :]
    Y_train = Y_train.loc[balanced_ids]
    A_train = A_train.loc[balanced_ids, :]
    return X_train, Y_train, A_train
```

To get an idea of the balance considering sensitive features gender and race an assesment can be made for all or some dataset returned by Listing 1.6 and Listing 1.5.

Sensitive feature counts for A_train datasets balanced by target

A serious imbalance can be seen in the training dataset for every combination of the sensitive features.

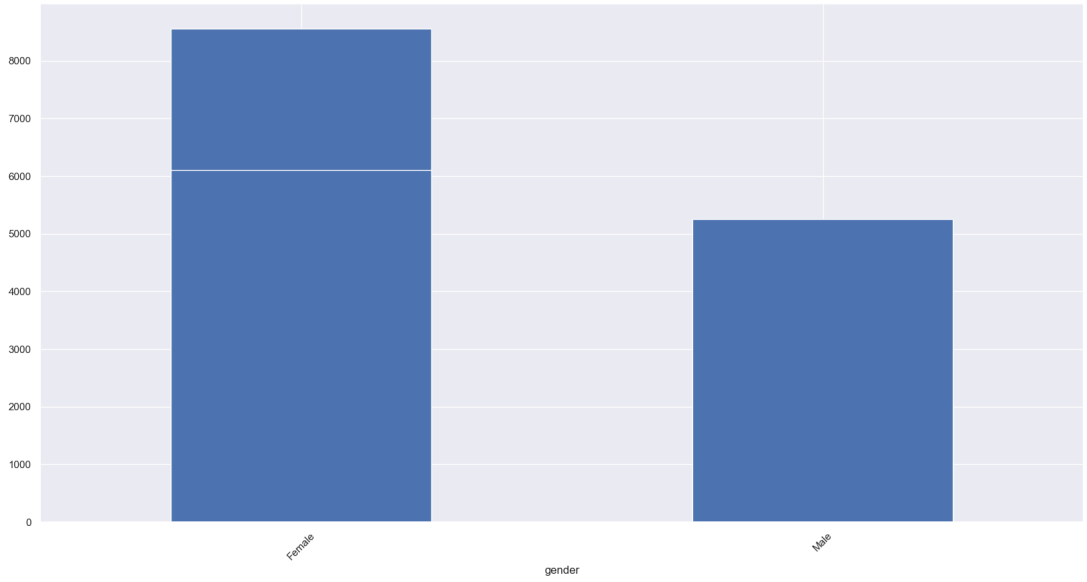

Figure 1.12: Gender counts for A_train dataset balanced by target

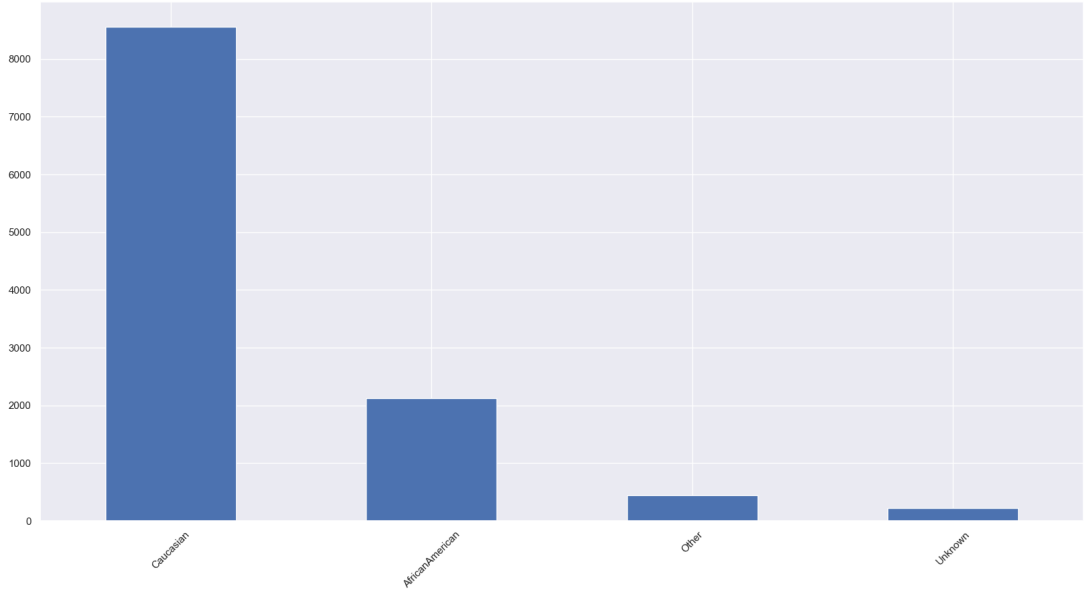

Figure 1.13: Race counts for A_train dataset balanced by target

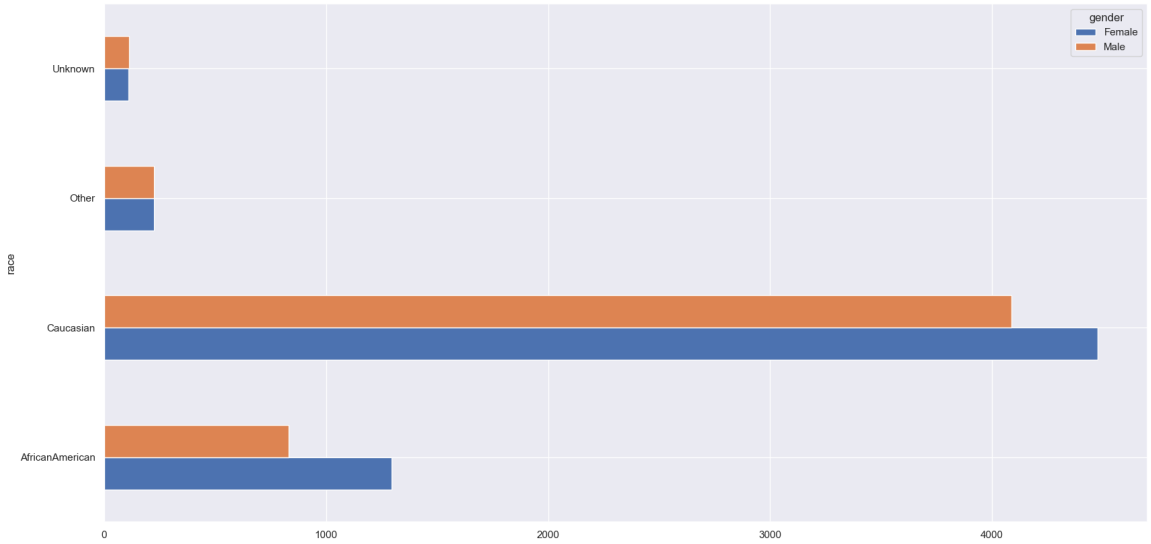

Figure 1.14: Counts for gender and race for A_train dataset balanced by target

Sensitive feature counts for A_test datasets balanced by target

A serious imbalance can be seen in the testing dataset for every combination of the sensitive features.

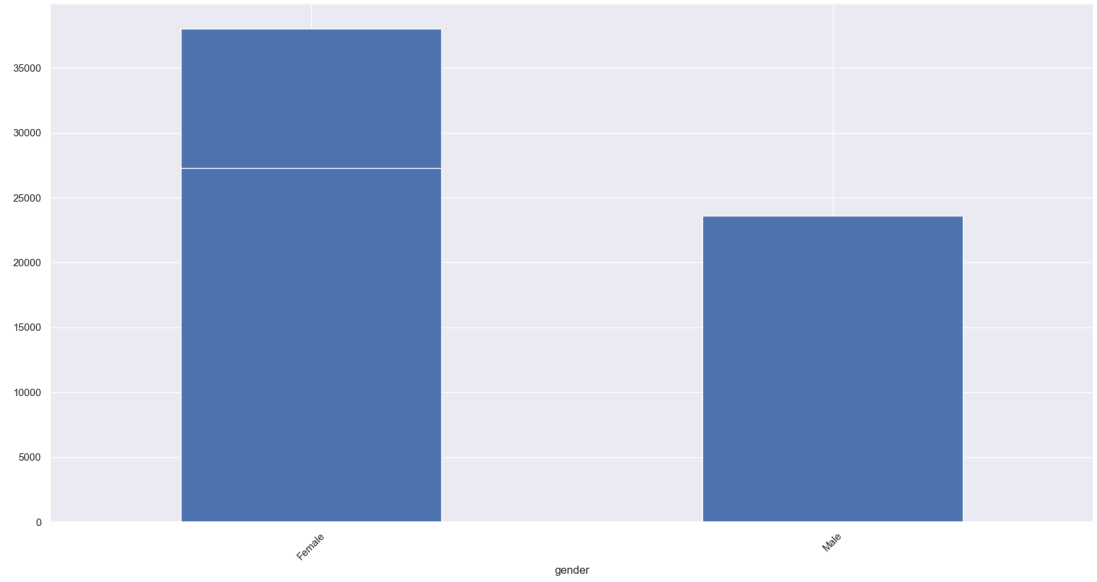

Figure 1.15: Gender counts for A_test dataset not balanced by target

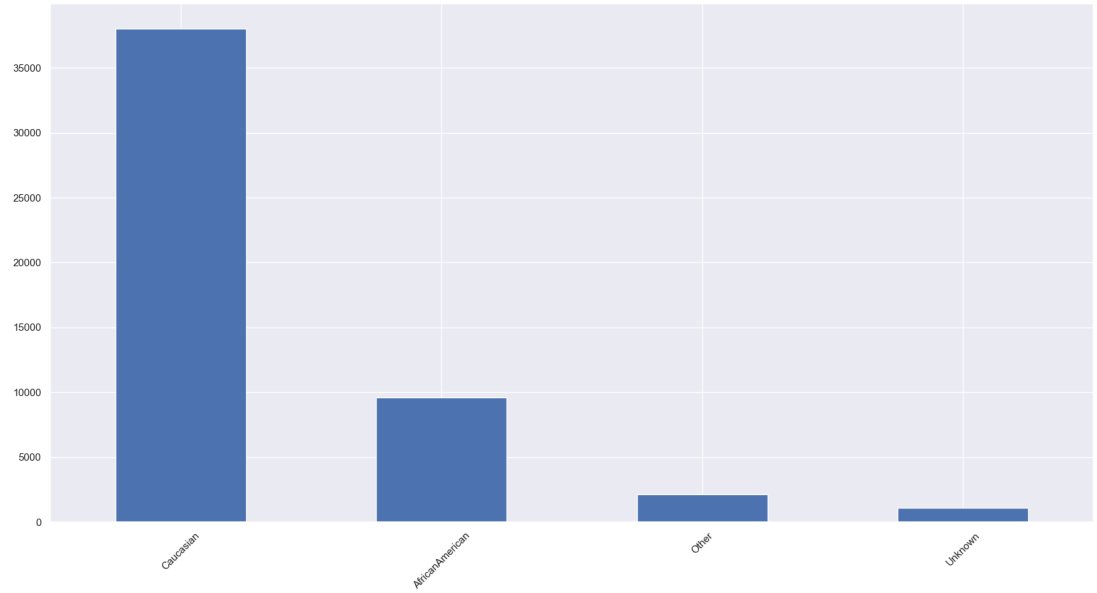

Figure 1.16: Race counts for A_test dataset not balanced by target

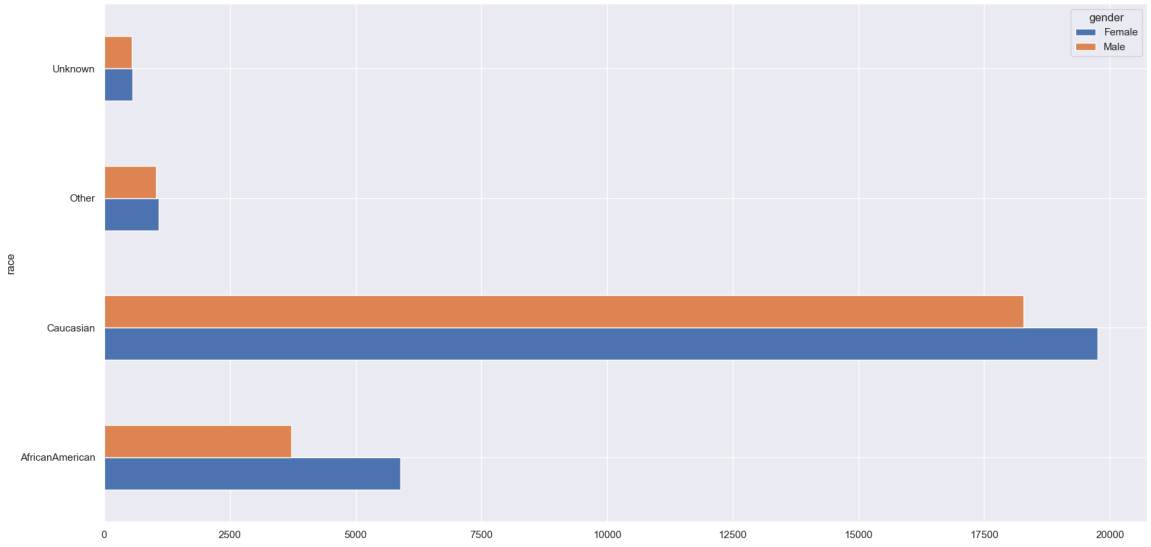

Figure 1.17: Counts for gender and race for A_test dataset not balanced by target

1.4.3 Training a logistic regression model

Some initial code

Some code to initiate and evaluate the training of a LR-model can be found in Listing 1.7.

Listing 1.7: Training a logistic regression model and analyse its performance

```
Y_pred_proba, Y_pred, unmitigated_pipeline = train_model_lr(X_train_bal, Y_train_bal
    ↪ , X_test)
display_performance(Y_test, Y_pred_proba, Y_pred)
coefficients(unmitigated_pipeline, X_test.columns)
```

1.4.4 Logistic regression: train and predict

Logistic regression is a statistical method used to analyze the relationship between a dependent variable and one or more independent variables. It is commonly used to predict the probability of a binary outcome, such as whether a customer will buy a product or not.

The logistic regression model is based on the logistic function, which is a sigmoid curve that maps any real-valued number into a value between 0 and 1.

Listing 1.7 refers to Listing 1.8, in this piece of code a pipeline is being constructed (line 2), after which it is trained (line 3) using the balanced data. Line 4 produces a list of probabilities for every item in the X_test set. These probabilities represent the probability to be part of the group of patients that will be readmitted in hospital before 30 days.

Listing 1.8: Training a logistic regression model

```
1 def train_model_lr(X_train_bal, Y_train_bal, X_test):
2 unmitigated_pipeline = Pipeline(steps=[("preprocessing", StandardScaler()), ("
    ↪ logistic_regression", LogisticRegression(max_iter=1000))])
3 unmitigated_pipeline.fit(X_train_bal, Y_train_bal)
4 Y_pred_proba = unmitigated_pipeline.predict_proba(X_test)[:, 1]
5 Y_pred = unmitigated_pipeline.predict(X_test)
6 return Y_pred_proba, Y_pred, unmitigated_pipeline
```

ROC curve

It can be more flexible to predict probabilities of an observation belonging to each class in a classification problem rather than predicting classes directly.

This flexibility comes from the way that probabilities may be interpreted using different thresholds that allow the operator of the model to trade-off concerns in the errors made by the model, such as the number of false positives compared to the number of false negatives. This is required when using models where the cost of one error outweighs the cost of other types of errors.

A ROC curve is a diagnostic tool that helps in the interpretation of probabilistic forecast for binary (two-class) classification predictive modeling problems.

ROC Curves summarize the trade-off between the true positive rate and false positive rate for a predictive model using different probability thresholds.

The ROC curve on Figure 1.18 shows the LR-model is only slightly better than an unskilled model. But performance is good enough to demonstrate its predictive qualities towards the target variable readmitted_30_days.

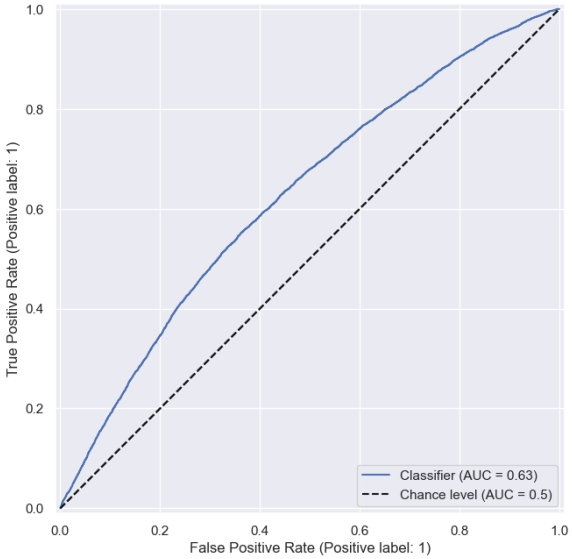

Figure 1.18: ROC curve for the logistic regression model

Code for the ROC curve To produce the a ROC curve use the code in Listing 1.9.

Listing 1.9: Display performance for a LR model

```
def display_performance(Y_test, Y_pred_proba, Y_pred, show=False):
p = RocCurveDisplay.from_predictions(Y_test, Y_pred_proba, plot_chance_level=True)
p.plot()
if show:
plt.show()
plt.savefig(generated_dir(True) + 'lr_roc_curve.png')
```

Exponentiated coefficients

The regular logistic regression coefficients represent the log odds that an observation is in the target class ("1") given the values of its X variables. To make these coefficients more intuitive, Figure 1.19 shows the exponentiated values for the coefficients retrieved from the trained LR-model.

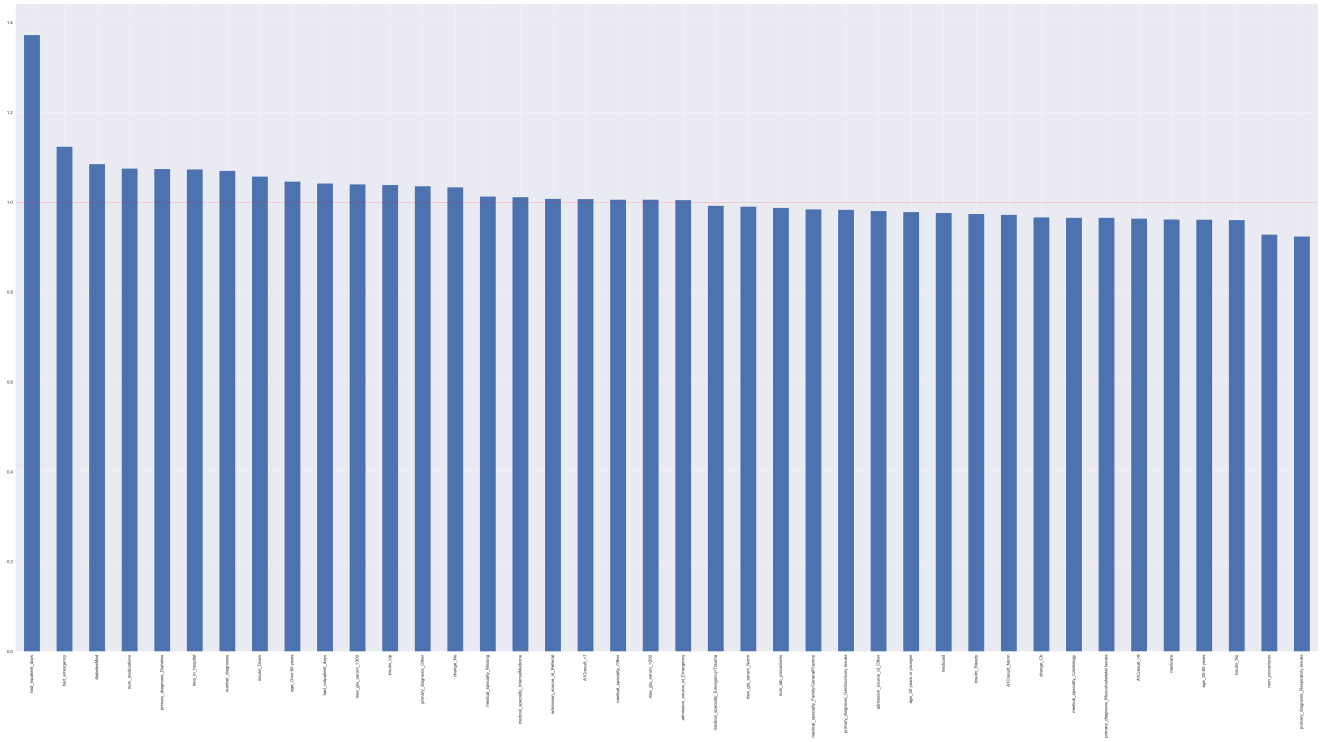

Figure 1.19: Regular coefficients exponentiated for the LR-model

Meaning of the exponentiated values For every one-unit increase in ”feature value”, the odds that the observation is in the target class are ”y-axis value” times as large as the odds that the observation is not in (y class) when all other variables are held constant. The red line shows which features are positive contributors to the readmitted_30_days target value, and which are negative contributors.

Code for the exponentiated coefficients To show the exponentiated coefficients for the LR-model see Listing 1.10

Listing 1.10: Display exponentiated coefficients for the LR-model

```
def coefficients(unmitigated_pipeline, columns, show=False):
 odds = np.exp(unmitigated_pipeline.named_steps["logistic_regression"].coef_[0])
 coefs = pd.DataFrame(odds, columns, columns=['coef']).sort_values(by='coef',
    ↪ ascending=False)
 coefs.plot.bar(figsize=(80, 40), legend=False, fontsize=15)
 plt.axhline(y=1, color='red', lw=.5)
 plt.savefig(generated_dir(True) + 'lr_coef1.png')
 if show:
  plt.show()
```

1.5 Fairness assessment

The steps of the assesment are as follows:

- Identify harms
- Identify the groups that might be harmed
- Quantify harms

- Compare quantified harms across the groups

In the health care scenario, we could consider two metrics for quantifying harms / benefits:

- false negative rate: fraction of patients that are readmitted within 30 days, but that are not recommended for the care management program; this quantifies harm
- selection rate: overall fraction of patients that are recommended for the care management program (regardless of whether they are readmittted with 30 days or not); this quantifies benefit; here the assumption is that all patients benefit similarly from the extra care.

There are several reasons for including selection rate in addition to false negative rate. To monitor how the benefits are allocated when focusing on groups that might be disadvantaged. Another reason is to get extra robustness in the assessment, because the measure (i.e., readmission within 30 days) is only an imperfect measure of the construct (who is most likely to benefit from the care management program). The auxiliary metrics, like selection rate, may alert us on large disparities in how the benefit is allocated.

1.5.1 Metrics used

Selection rate Overall fraction of patients that are recommended for the care management program

Balanced accuracy It's the arithmetic mean of sensitivity and specificity, its use case is when dealing with imbalanced data, i.e. when one of the target classes appears a lot more than the other.

Balanced accuracy = (TPR + TNR)/2

TPR = TP/(TP+FN)
TNR = TN/(TN+FP)

False negative rate FNR = 1-TPR

1.5.2 Unmitigated metrics

To get an idea of the metrics for the LR-model, see Figure 1.20. The female patient with an unknown race has the worst metrics, she has the lowest selection rate, while when not selected there is a high chance she should have been selected and finally the balanced accuracy is much lower, it differs up to 16 percentage points when compared with balanced accuracy measurements for other groups.

race	gender	selection_rate	false_negative_rate	balanced_accuracy	count
AfricanAmerican	Female	0.42	0.43	0.59	5889.0
AfricanAmerican	Male	0.4	0.42	0.61	3727.0
Caucasian	Female	0.42	0.43	0.59	19758.0
Caucasian	Male	0.39	0.47	0.58	18281.0
Other	Female	0.35	0.55	0.56	1086.0
Other	Male	0.36	0.44	0.61	1027.0
Unknown	Female	0.35	0.74	0.45	567.0
Unknown	Male	0.36	0.47	0.59	547.0

Figure 1.20: Unmitigated metrics for the LR-model

A more aggregated view on these metrics can be seen on Figure 1.21, values are rounded. The biggest difference for the selection_rates is 0.07, the ratio is the minimum ratio between groups for each metric. A

ratio of 0.84 is the value obtained when dividing 0.35 (lowest group selection rate) by 0.42 (highest group selection rate). The biggest differences for groups can be seel when comparing false_negative_rates.

	selection_rate	false_negative_rate	balanced_accuracy	count
difference	0.07	0.33	0.16	19211.0
ratio	0.84	0.56	0.74	0.03
group_min	0.35	0.42	0.45	547.0
group_max	0.42	0.74	0.61	19758.0

Figure 1.21: Unmitigated aggregated metrics for the LR-model

Code to produce metrics The code to generate these metrics is in Listing 1.11. The sensitive_features parameter is a dataframe containing the the sensitive feature columns.

Listing 1.11: Metrics

```
def metrics(metrics_dict, sensitive_features, Y_test, Y_pred):

 metricframe_ = MetricFrame(metrics=metrics_dict, y_true=Y_test, y_pred=Y_pred,
    ↪ sensitive_features=sensitive_features)

 metricframe_.by_group.round(2).to_csv(dir+ filename_part+"metrics.csv")

 metrics_aggregated = pd.DataFrame({
  'difference': metricframe_.difference(),
  'ratio': metricframe_.ratio(),
  'group_min': metricframe_.group_min(),
  'group_max': metricframe_.group_max()
  }).T

 metrics_aggregated.astype(float).round(2).to_csv(dir +filename_part +"metrics_agg.
    ↪ csv")

 metricframe_.by_group.plot.bar(subplots=True, layout=[2, 2], figsize=(12, 12),
    ↪ legend=False, rot=90, position=.5)
 plt.savefig(dir + filename_part + 'mf.png')
 plt.show()
```

The dictionary containing the metrics to be considered looks like in Listing 1.12.

Listing 1.12: Metrics dictionary

```
metrics_dict = {
 "selection_rate": selection_rate,
 "false_negative_rate": false_negative_rate,
 "balanced_accuracy": balanced_accuracy_score,
 "count": count
}
```

1.5.3 Mitigating differences using TresholdOptimizer

ThresholdOptimizer in the module fairlearn.postprocessing implements the approach proposed in [2], which takes in an existing (possibly pre-fit) machine learning model, uses its predictions as a scoring function, and identifies a separate threshold for each group defined by a sensitive feature in order to optimize some specified objective (such as balanced accuracy) subject to specified fairness constraints (such as false negative rate parity). The resulting classifier is a thresholded version of the provided machine learning model.

As can be seen in Figure 1.22, figures are better when considering false negative rates, in general they have dropped, and the differences between them have lessened. Selection rates have risen and differences have dropped significantly, while keeping more or less the same balanced accuracy.

race	gender	selection_rate	false_negative_rate	balanced_accuracy	count
AfricanAmerican	Female	0.44	0.4	0.59	5889.0
AfricanAmerican	Male	0.43	0.38	0.61	3727.0
Caucasian	Female	0.45	0.39	0.59	19758.0
Caucasian	Male	0.46	0.37	0.6	18281.0
Other	Female	0.42	0.44	0.58	1086.0
Other	Male	0.43	0.4	0.6	1027.0
Unknown	Female	0.5	0.67	0.41	567.0
Unknown	Male	0.41	0.43	0.59	547.0

Figure 1.22: Metrics for the LR-model after mitigation using the ThrresholdOptimizer

Code to get the optimizer The code to get the optimizer is in Listing 1.13.

Listing 1.13: Getting the thresholdoptimizer

```
def get_threshold_optimizer(unmitigated_pipeline):
 postprocess_est = ThresholdOptimizer(
  estimator=unmitigated_pipeline,
  constraints="false_negative_rate_parity",
  objective="balanced_accuracy_score",
  prefit=True,
  predict_method='predict_proba'
  )
 return postprocess_est
```

Code to use the optimizer The code to get and use the optimizer, and to obtain the metrics is in Listing 1.15.

Listing 1.14: Using the thresholdoptimizer

```
to = get_threshold_optimizer(unmitigated_pipeline)
to.fit(X_train_bal, Y_train_bal, sensitive_features=A_train_bal)
Y_pred_postprocess = to.predict(X_test, sensitive_features=A_test)
metrics(metrics_dict, df_test[sensitive_features], Y_test, Y_pred_postprocess)
```

1.5.4 Mitigating differences using ExponentiatedGradient

With the ThresholdOptimizer, a model's decision boundary was transformed to satisfy our fairness constraints. One limitation of ThresholdOptimizer is that it needs access to the sensitive features at deployment time. In this section, we will show how to use the reductions approach proposed in [1] to obtain a model that satisfies the fairness constraints, but does not need access to sensitive features at deployment time.

The term "reductions" refers to a wrapper approach, which instead of wrapping an already trained model, wraps any standard classification or regression algorithm, such as LogisticRegression. In other words, an input to a reduction algorithm is an object that supports training on any provided (weighted) dataset. In addition, a reduction algorithm receives a data set that includes sensitive features. The goal, like with post-processing, is to optimize a performance metric (such as classification accuracy) subject to fairness constraints (such as an upper bound on differences between false negative rates).

The main reduction algorithm algorithm in Fairlearn is ExponentiatedGradient. It creates a sequence of reweighted datasets and retrains the wrapped model on each of them. The retraining process is guaranteed to find a model that satisfies the fairness constraints while optimizing the performance metric. The model returned by ExponentiatedGradient consists of several inner models, returned by the wrapped estimator. At deployment time, ExponentiatedGradient randomizes among these models according to a specific probability weights. Unfortunately after applying the ExpoenntiatedGradient nothing changed in the metrics.

race	gender	selection_rate	false_negative_rate	balanced_accuracy	count
AfricanAmerican	Female	0.42	0.43	0.59	5889.0
AfricanAmerican	Male	0.4	0.42	0.61	3727.0
Caucasian	Female	0.42	0.43	0.59	19758.0
Caucasian	Male	0.39	0.47	0.58	18281.0
Other	Female	0.35	0.55	0.56	1086.0
Other	Male	0.36	0.44	0.61	1027.0
Unknown	Female	0.35	0.74	0.45	567.0
Unknown	Male	0.36	0.47	0.59	547.0

Figure 1.23: Metrics for the LR-model after mitigation using the ExponentiatedGradient

Code to apply the exponentiated gradient

Listing 1.15: Using the exponentiated gradient

```
estimator = unmitigated_pipeline.named_steps['logistic_regression']
eg = get_exponentiated_gradient1(estimator, random_seed)
eg.fit(X_train_bal, Y_train_bal, sensitive_features=A_train_bal)
Y_pred_reductions = eg.predict(X_test, random_state=random_seed)
metrics(metrics_dict, df_test[sensitive_features], Y_test, Y_pred_reductions,
    ↪ use_log_reg, False, False)
```

Chapter 2

Explainable AI

2.1 SHAP

On this site I found some information on SHAP. SHAP — which stands for SHapley Additive exPlanations — is probably the state of the art in Machine Learning explainability. This algorithm was first published in [3] and it is a brilliant way to reverse-engineer the output of any predictive algorithm.

In a nutshell, SHAP values are used whenever one has a complex model (could be a gradient boosting, a neural network, or anything that takes some features as input and produces some predictions as output) and one wants to understand the decisions the model is making. The Shapley value indicates how much impact each feature has on the prediction, or (more precisely) how much each feature moves the prediction away from the average prediction.

SHAP values are based on game theory and assign an importance value to each feature in a model. Features with positive SHAP values positively impact the prediction, while those with negative values have a negative impact. The magnitude is a measure of how strong the effect is.

2.1.1 Properties of SHAP values

Additivity SHAP values are additive, which means that the contribution of each feature to the final prediction can be computed independently and then summed up. This property allows for efficient computation of SHAP values, even for high-dimensional datasets.

Local accuracy SHAP values add up to the difference between the expected model output and the actual output for a given input. This means that SHAP values provide an accurate and local interpretation of the model's prediction for a given input.

Missingness SHAP values are zero for missing or irrelevant features for a prediction. This makes SHAP values robust to missing data and ensures that irrelevant features do not distort the interpretation.

Consistency SHAP values do not change when the model changes unless the contribution of a feature changes. This means that SHAP values provide a consistent interpretation of the model's behavior, even when the model architecture or parameters change.

2.1.2 Get data and train LR-model

The code to get SHAP-values is in Listing 2.1. The same dataset and dataset split routine is chosen as in the previous chapter. On line 5 a trained logistic regression pipeline is returned. The LR-model is used together with the balanced training data to construct an explainer on line 7, this explainer is used on

line 8 to calculate SHAP-values for the X_test values. The SHAP-values are represented by a matrix that contains a row with feature SHAP-values, for every row in X.

Listing 2.1: Prepare for SHAP

```
clean_specific_dir(shap_dir())
df = load_dataset()
X_train, X_test, Y_train, Y_test, A_train, A_test, df_train, df_test =
    ↪ prepare_test_train_datasets(df, random_seed)
X_train_bal, Y_train_bal, A_train_bal = resample_dataset(X_train, Y_train, A_train)
pipeline = train_model_lr_(X_train_bal, Y_train_bal)
model = pipeline.named_steps['logistic_regression']
explainer = LinearExplainer(model, X_train_bal, feature_names=X_train_bal.columns.
    ↪ to_list())
shap_values = explainer(X_test)
```

In the next subsections the SHAP-values will be used to generate some plots.

2.1.3 Waterfall plot for one instance

To get an idea of SHAP-values for a specific example see Figure 2.1, this example is actually not to be recommended for the care program. The (negative) contribution to the expected value is mainly made by had_inpatient_days (this feature is zero for this instance) and num_medications. It is important to know that these values contribute to the predicted value for this specific sample, even if it is a wrong prediction. In this case however the prediction and the ground truth value are equal.

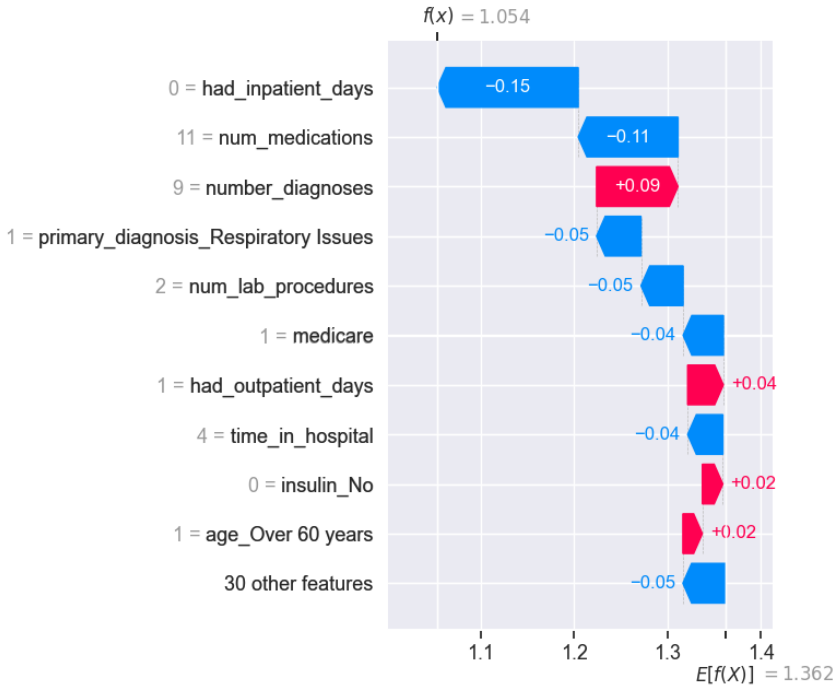

Figure 2.1: SHAP-values for one example in the dataset

The next example in Figure 2.2 is to be recommended for the care program, the predicted value for this example was indeed the recommendation. The (positive) contribution to the expected value is mainly made by had_inpatient_days (which is one for this instance), num_medications and time_in_hospital. The expected value is 1.362 for every example, this is without knowing any properties. In this case, this value gets raised by had_inpatient_days, num_medications and time_in_hospital.

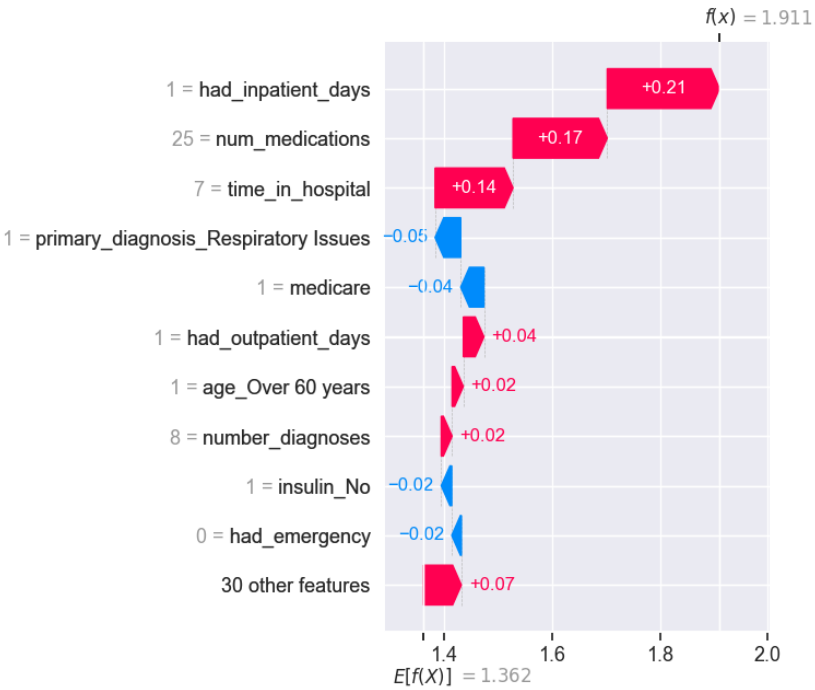

Figure 2.2: SHAP-values for another example in the dataset

Code for generating these plots

The code for generating a waterfall is actually one line, see line 15 or 11. To check whether these examples were indeed correctly classified, I had to do some checks, lines 5,6 and 8,9.

Listing 2.2: Waterfall plot

```
1  Y_test_list= Y_test.to_list()
2  Y_test_pred_proba = (unmitigated_pipeline.predict_proba(X_test)[:, 1] >= 0.5).astype
       ↪ (int)
3
4  first_instance_recommended = Y_test_list.index(1)
5  print("y_test value:",Y_test_list[first_instance_recommended])
6  print("Y_test_pred_proba:", Y_test_pred_proba[first_instance_recommended])
7  first_instance_not_recommended = Y_test_list.index(0)
8  print("Y_test value:", Y_test_list[first_instance_not_recommended])
9  print("Y_test_pred_proba:", Y_test_pred_proba[first_instance_not_recommended])
10
11 plots.waterfall(shap_values[first_instance_recommended], show=False, max_display=11)
12 plt.savefig(shap_dir() + "shap_waterfall_1.png")
13 plt.clf()
14
15 plots.waterfall(shap_values[first_instance_not_recommended], show=False, max_display
       ↪ =11)
16 plt.savefig(shap_dir() + "shap_waterfall_0.png")
17 plt.clf()
```

2.1.4 Summary plot

Passing a matrix of SHAP-values to the summary_plot function creates a global feature importance plot, where the global importance of each feature is taken to be the mean absolute value for that feature over all the given samples.

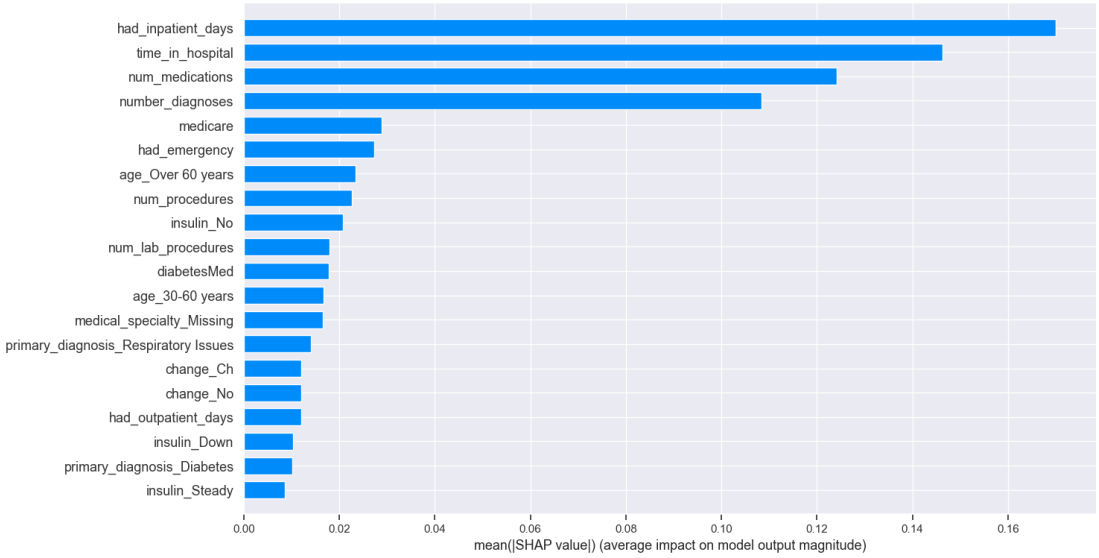

Figure 2.3: Summary plot for the complete test dataset

2.1.5 Beeswarm plot

The beeswarm plot is designed to display an information-dense summary of how the top features in a dataset impact the model's output. Every instance's feature value is represented by a dot. High values for num_of_medications feature have a positive impact on being recommended for the care program. As can be seen some instances having high values for num_of_medications, also had very high positve impact on model output, positive impact meaning raising chances of recommendation for the care program.

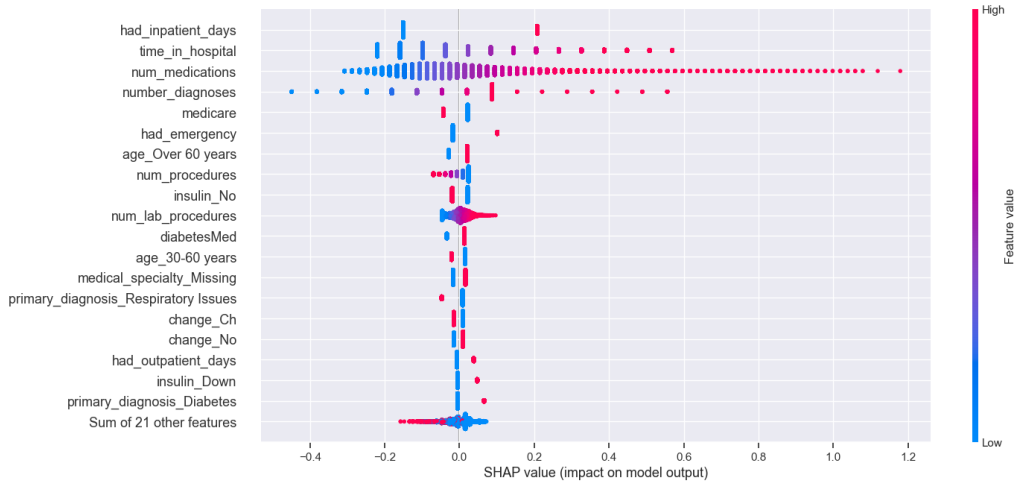

Figure 2.4: SHAP-values for the complete test dataset

Passing a matrix of SHAP-values to the beeswarm function creates a beeswarm plot, although sometimes it helps converting boolean values to 0 and 1 values.

2.2 LIME

LIME stands for Local Interpretable Model-Agnostic Explanations. The beauty of LIME its accessibility and simplicity, it was proposed in [4]. The core idea behind LIME though exhaustive is really intuitive and simple! Let's dive in and see what the name itself represents:

- Model agnosticism: LIME can give explanations for any given supervised learning model by treating it as a 'black box'.
- Local explanations mean that LIME gives explanations that are locally faithful within the surroundings or vicinity of the observation/sample being explained.

2.2.1 Explain instance

For this part the Boston Housing dataset is used, the purpose of the model will be to predict house prices based on variables describing the house and the area in which the house is located. The columns for the dataset are in Figure 2.5. The MDEV feature will be the target feature.

- CRIM - per capita crime rate by town
- ZN - proportion of residential land zoned for lots over 25,000 sq.ft.
- INDUS - proportion of non-retail business acres per town.
- CHAS - Charles River dummy variable (1 if tract bounds river; 0 otherwise)
- NOX - nitric oxides concentration (parts per 10 million)
- RM - average number of rooms per dwelling
- AGE - proportion of owner-occupied units built prior to 1940
- DIS - weighted distances to five Boston employment centres
- RAD - index of accessibility to radial highways
- TAX - full-value property-tax rate per $10,000
- PTRATIO - pupil-teacher ratio by town
- B - 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
- LSTAT - % lower status of the population
- MEDV - Median value of owner-occupied homes in $1000's

Figure 2.5: Features for the Boston housing set.

On the left of Figure 2.6 the predicted value can be seen, in the middle for some features contributions to the final predicted value can be seen, while on the right feature values can be seen for the example under observation.

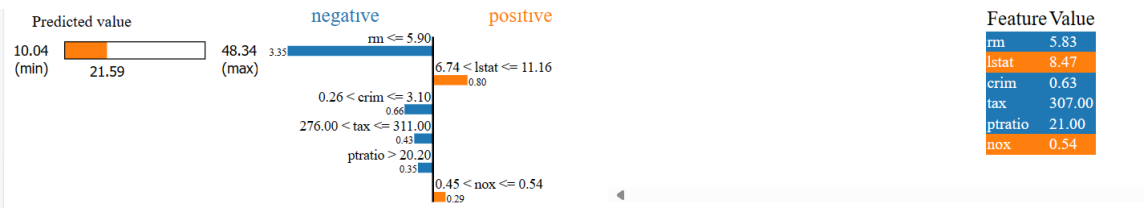

Figure 2.6: LIMEvalues for an individual example

2.2.2 Code

The code in Listing 2.3, opens a dataset, fills missing data with mean values, splits the data in target and influencing variables, creates test and train datasets, trains a RandomForestRegressor-model, produces a basic performance figure, and constructs a lime explainer, which is used to explain a single instance.

Listing 2.3: Lime

```
df = pd.read_csv('./data/BostonHousing.csv')
df = df.fillna(df.mean())
df.isnull().sum()
X = df[['lstat', 'rm', 'nox', 'ptratio', 'dis', 'age', 'b', 'zn', 'rad', 'tax', '
    ↪ chas', 'indus', 'crim']]
y = df['medv']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
    ↪ random_state=0)
model = RandomForestRegressor(max_depth=6, random_state=0, n_estimators=10)
model.fit(X_train, y_train)
print('R2 score for the model on test set =', model.score(X_test, y_test))

explainer = lime_tabular.LimeTabularExplainer(X_train.values, feature_names=X_train.
    ↪ columns.values.tolist(),
class_names=['medv'], verbose=True, mode='regression')

exp = explainer.explain_instance(X_test.values[5], model.predict, num_features=6)

fig = exp.as_pyplot_figure()
plt.tight_layout()
plt.show()
plt.savefig(lime_dir()+'lime_report.jpg')
exp.save_to_file(lime_dir()+'lime.html')
```

Bibliography

[1] Alekh Agarwal, Alina Beygelzimer, Miroslav Dudík, John Langford, and Hanna Wallach. A reductions approach to fair classification, 2018.

[2] Moritz Hardt, Eric Price, and Nathan Srebro. Equality of opportunity in supervised learning, 2016.

[3] Scott Lundberg and Su-In Lee. A unified approach to interpreting model predictions, 2017.

[4] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier, 2016.