6.1HD - Explainable AI

October 12, 2022

Welcome to your assignment this week!

To better understand explainable AI, in this assignment, we will look at the **LIME** framework to explain potential black-box machine learning models in a model-agnostic way. We use a real-world dataset on Census income, also known as the ***Adult dataset*** available in the *UCI* ML Repository where we will predict if the potential income of people is more than $50K/year or not.

For this assignment, we will use:

- XGBoost (XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable.). Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees.
- Decision Tree which is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements. Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal, but are also a popular tool in machine learning.

LIME GitHub

After this assignment you will be able to: use the **LIME** framework to explain potential black-box machine learning models in a model-agnostic way.

Run the following cell to load the packages you will need.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import gridspec

import seaborn as sns
import lime
from lime.lime_tabular import LimeTabularExplainer
import shap
import itertools

##
from utils import *
##

from sklearn.model_selection import train_test_split
```

```
from collections import Counter
import xgboost as xgb
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier

import graphviz
from io import StringIO
from sklearn import datasets,tree
import pydot
import pydotplus
from sklearn.tree import export_graphviz
from IPython.display import Image


import warnings
warnings.filterwarnings('ignore')
#plt.style.use('fivethirtyeight')
%matplotlib inline

shap.initjs()
```

```
<frozen importlib._bootstrap>:219: RuntimeWarning:
scipy._lib.messagestream.MessageStream size changed, may indicate binary
incompatibility. Expected 56 from C header, got 64 from PyObject
```

```
<IPython.core.display.HTML object>
```

Next, let's load the census dataset. Run the following cell to load the features `X` and the labels `y`.

[2]:
```
X_raw, y = shap.datasets.adult(display=True)

X_raw = X_raw.drop(columns=['Capital Gain']) # These two features are␣
 ↪intentionally removed.
X_raw = X_raw.drop(columns=['Capital Loss']) # These two features are␣
 ↪intentionally removed.

labels = np.array([int(label) for label in y])

print('The shape of X_raw is: ',X_raw.shape)
print('The shape of y is: ',labels.shape)
```

```
The shape of X_raw is:  (32561, 10)
The shape of y is:  (32561,)
```

You've loaded:

- `X_raw`: a DataFrame containing 32,561 instances with 12 features.

- `y`: the list of binary labels for the 32,561 examples. If salary of instance i is more than \$50K: $y^{(i)} = 1$ otherwise: $y^{(i)} = 0$.

1 Understanding the Census Income Dataset

Let's now take a look at our dataset attributes and understand their meaning and significance.

Num	Attribute Name	Type	Description
1	Age	Continuous	Represents age of the person (Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked)
2	Workclass	Categorical	Represents the workclass of the person. (Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked).
3	Education-Num	Categorical	Numeric representation of educational qualification.Ranges from 1-16.(Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool)
4	Marital Status	Categorical	Represents the marital status of the person(Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse)
5	Occupation	Categorical	Represents the type of profession job of the person(Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces)
6	Relationship	Categorical	Represents the relationship status of the person(Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried)
7	Race	Categorical	Represents the race of the person(White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black)
8	Sex	Categorical	Represents the gender of the person(Female, Male)
9	Capital Gain	Continuous	The total capital gain for the person
10	Capital Loss	Continuous	The total capital loss for the person
11	Hours per week	Continuous	Total hours spent working per week
12	Country	Categorical	The country where the person was born
13	Income Label	Categorical	The class label column is the one we want to predict

We have a total of 12 features and our objective is to predict if the income of a person will be more than \$50K (True) or less than \$50K (False). Hence we will be building and interpreting a classification model.

Let's have a look at the first three instances of the dataset (you can use `X.head(3)` to see the content of the dataset):

	Age	Workclass	Education-Num	Marital Status	Occupation	Relationship	Race	Sex	Capital Gain	Capital Loss	Hours per week	Country
0	39.0	State-gov	13.0	Never-married	Adm-clerical	Not-in-family	White	Male	2174.0	0.0	40.0	United-States
1	50.0	Self-emp-not-inc	13.0	Married-civ-spouse	Exec-managerial	Husband	White	Male	0.0	0.0	13.0	United-States
2	38.0	Private	9.0	Divorced	Handlers-cleaners	Not-in-family	White	Male	0.0	0.0	40.0	United-States

For example, the first person is 39 years old, works for the state governement, is a Male and was born in the US. By using `y[0:3]` you can get binary values indicating whether these persons have an income higher than $50K or no.

2 Pre-processing

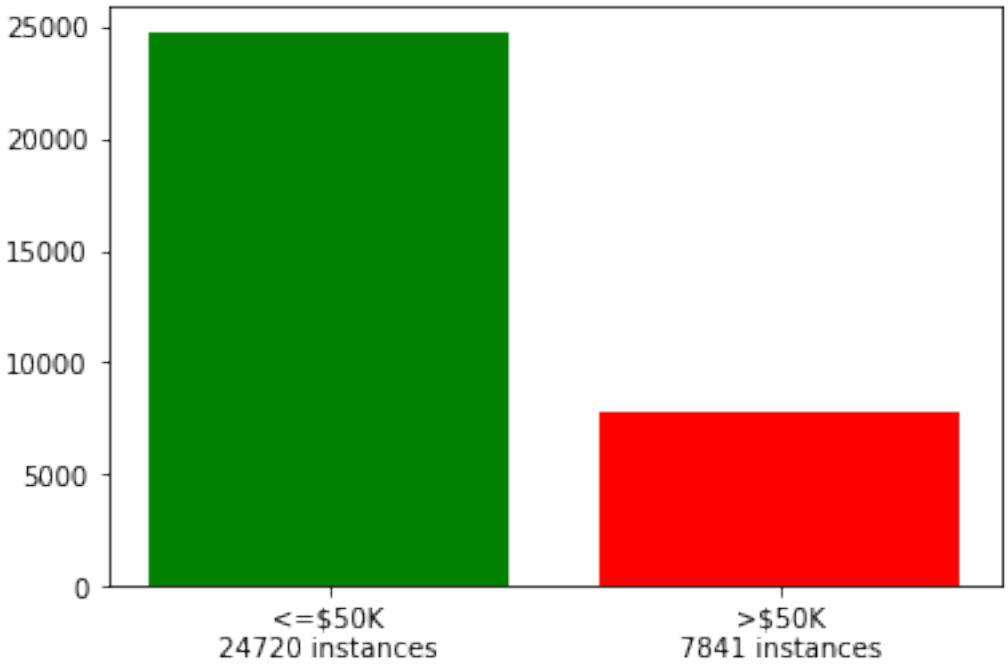
Converting the categorical columns with string values to numeric representations. Typically the XGBoost model can handle categorical data natively being a tree-based model so we don't one-hot encode the features

[3]:
```
mapping = {}
cat_cols = X_raw.select_dtypes(['category']).columns
for col in cat_cols:
    mapping[col] = dict( enumerate(X_raw[col].cat.categories ) )
indices_cat_cols = [ list(X_raw.columns).index(x) for x in cat_cols ]
X = X_raw.copy()
X[cat_cols] = X_raw[cat_cols].apply(lambda x: x.cat.codes)
headers=list(X.columns)
X.head(3)
```

[3]:
```
    Age  Workclass  Education-Num  Marital Status  Occupation  Relationship  \
0  39.0          7           13.0               4           1             1
1  50.0          6           13.0               2           4             0
2  38.0          4            9.0               0           6             1

   Race  Sex  Hours per week  Country
0     4    1            40.0       39
1     4    1            13.0       39
2     4    1            40.0       39
```

Let's have a look at the distribution of people with <= $50K (0) and > $50K (1) income:

[4]:
```
plt.bar([0], height=[Counter(labels)[0]], color="green")
plt.bar([1], height=[Counter(labels)[1]], color="red")
plt.xticks([0, 1], ['<=$50K\n'+str(Counter(labels)[0])+' instances',
                    '>$50K\n'+str(Counter(labels)[1])+' instances'])
plt.show()
```



3 Split Train and Test Datasets

As in any Machine Learning, we need to partition the dataset into two subsets – a training and testset. Please note that in practice, the dataset needs to be partitioned into three subsets, the third once being the validation set which will be used for hyperparameters tuning. However, in this assignment, we will not tune the hyperparameters.

Run the following to split the dataset accordingly:

[5]:
```
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.3,␣
↪random_state=42, stratify=y)
print('The shape of training set is: ',X_train.shape)
print('The shape of test set is: ',X_test.shape)
```

```
The shape of training set is:  (22792, 10)
The shape of test set is:  (9769, 10)
```

You've created:

- `X_train`: a trainig DataFrame containing 22,792 instances used for training.
- `y_train`: the list of binary labels for the 22,792 instances of the training set.
- `X_test`: a test DataFrame containing 9,769 instances used for testing.
- `y_test`: the list of binary labels for the 9,769 instances of the test set.

We note that since we are using a stratified splitting, the distribution of samples in the training and test set is similar to the distribution in the dataset, i.e., roughly 24% of positive examples in each subset.

3.1 Training the classification model

Now we train and build a boosting classification model on our training data using XGBoost (XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable.). Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees.

Run the following to start the training of the classifier:

[6]:
```
xgc = xgb.XGBClassifier(n_estimators=500, max_depth=5, base_score=0.5,
                        objective='binary:logistic', random_state=42)
xgc.fit(X_train, y_train)
```

[6]:
```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=5, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=500,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=42,
              reg_alpha=0, reg_lambda=1, …)
```

4 Predictions on the test data

Now that the classifier is trained, let's make few predictions on the test set:

[7]:
```
predictions = xgc.predict(X_test)
print("The values predicted for the first 20 test examples are:")
print(predictions[:20])

print("The true values are:")
print(y_test[:20])
```

```
The values predicted for the first 20 test examples are:
[1 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 1 1 0 0]
The true values are:
[1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0]
```

As you can see, our classier is making only 2 errors!

5 Model Performance

Let's now evaluate the performance of our classifier on the test set. For that, we will call `sklearn.metrics.classification_report()` which returns a text report showing the main classification metrics including: Presicion, Recall, and F1-Score. The reported averages include macro

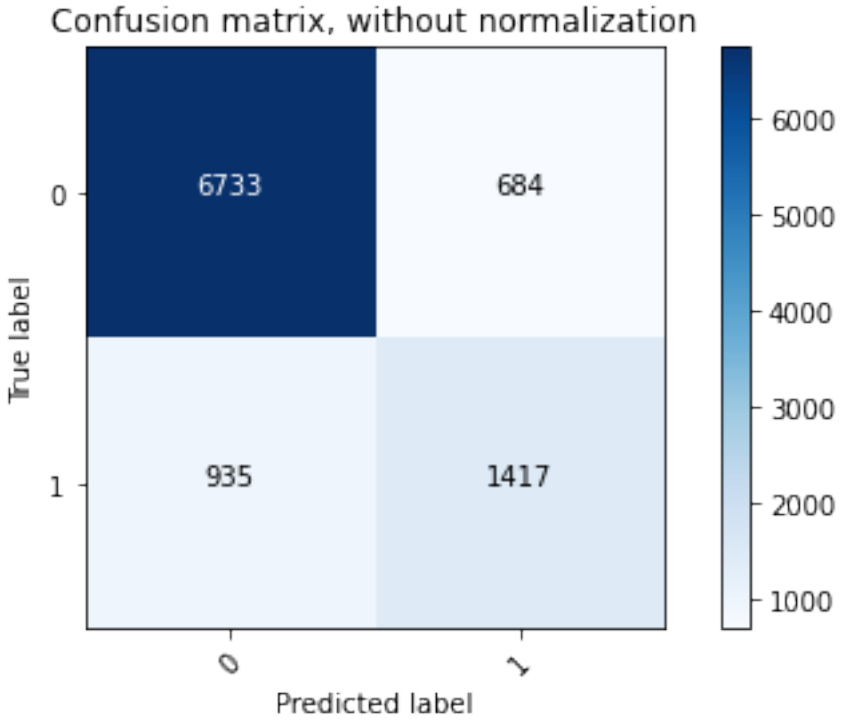
average (averaging the unweighted mean per label) and weighted average (averaging the support-weighted mean per label).

[8]:
```
report = classification_report(y_test, predictions)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.88      0.91      0.89      7417
           1       0.67      0.60      0.64      2352

    accuracy                           0.83      9769
   macro avg       0.78      0.76      0.76      9769
weighted avg       0.83      0.83      0.83      9769
```

To get more details, let's print the confusion matrix:

[9]:
```
class_labels = list(set(labels))
cnf_matrix = confusion_matrix(y_test, predictions)
plot_confusion_matrix(cnf_matrix, classes=class_labels,
                      title='Confusion matrix, without normalization')
```



Task 1: Please provide comments on the performance of the classifier. *** ##### From the

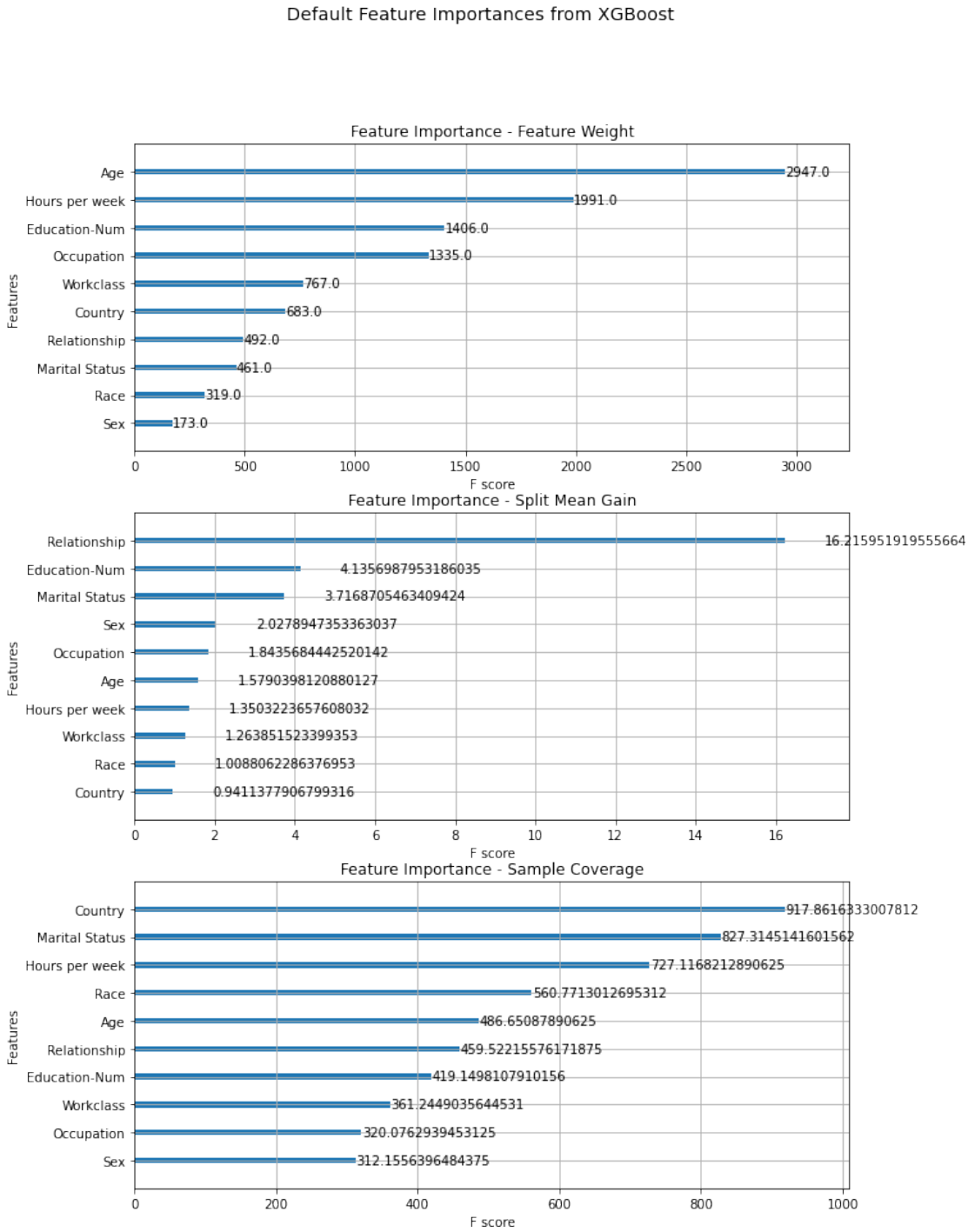
above confusion matrix we can observe that the classifier is biased towards class label 0 as has high correct samples for label 0 (90%) whereas low correct samples for label 1 (60%). This shows that while the overall accuracy is decent (83%), the overall performance of the classfiier is not good. Also, the dataset is imbalanced and the classfier is biased towards the dominant class.

6 Feature importance:

The global feature importance calcuations that come with XGBoost, enables us to view feature importances based on the following:

- **Feature Weights**: This is based on the number of times a feature appears in a tree across the ensemble of trees.
- **Gain**: This is based on the average gain of splits which use the feature.
- **Coverage**: This is based on the average coverage (number of samples affected) of splits which use the feature.

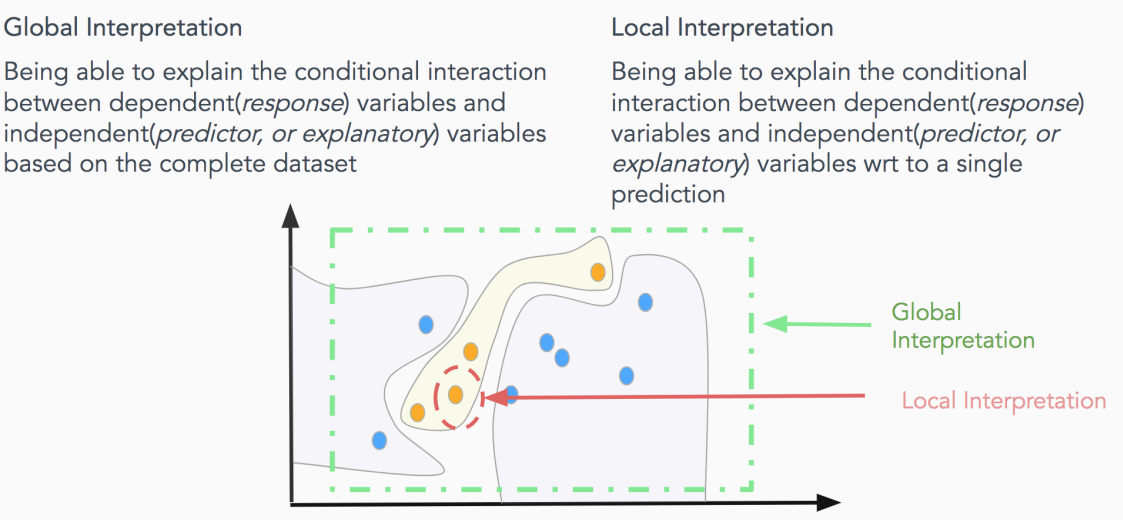
[10]:
```
fig = plt.figure(figsize = (10, 15))
title = fig.suptitle("Default Feature Importances from XGBoost", fontsize=14)

ax1 = fig.add_subplot(3,1, 1)
xgb.plot_importance(xgc, importance_type='weight', ax=ax1)
t = ax1.set_title("Feature Importance - Feature Weight")

ax2 = fig.add_subplot(3,1, 2)
xgb.plot_importance(xgc, importance_type='gain', ax=ax2)
t = ax2.set_title("Feature Importance - Split Mean Gain")

ax3 = fig.add_subplot(3,1, 3)
xgb.plot_importance(xgc, importance_type='cover', ax=ax3)
t = ax3.set_title("Feature Importance - Sample Coverage")
```



Task 2: Please provide comments on the above global feature importance calcuations.

From the feature importance calculations we can observe that different methods of feature importance calculations rank carious features according to their relevance in predictions of income, Hours per week and Education number feature in top three of two methods, hence being the most prominent features to contribute in overall income according to the above methods. Age is another important feature as it shows the most weight in feature weight method and there is a significant difference from the other features' weights.

7 Model Interpretation Methods



8 LIME:

Lime is able to explain any black box classifier, with two or more classes. All we require is that the classifier implements a function that takes in raw text or a numpy array and outputs a probability for each class. LIME tries to fit a global surrogate model, LIME focuses on fitting local surrogate models to explain why single predictions were made.

Since XGBoost has some issues with feature name ordering when building models with dataframes (we also needed feature names in the previous `plot_importance()` calls), we will build our same model with numpy arrays to make LIME work. Remember the model being built is the same ensemble model which we treat as our black box machine learning model.

Note the difference with the previous `fit` call:

xgc_np.fit(X_train, y_train) vs. xgc_np.fit(X_train.values, y_train)

[11]:
```
xgc_np = xgb.XGBClassifier(n_estimators=500, max_depth=5, base_score=0.5,
                           objective='binary:logistic', random_state=42)
mymodel = xgc_np.fit(X_train.values, y_train)
```

`LimeTabularExplainer` class helps in explaining predictions on tabular (i.e. matrix) data. For numerical features, it perturbs them by sampling from a Normal(0,1) and doing the inverse operation of mean-centering and scaling, according to the means and stds in the training data. For

categorical features, it perturbs by sampling according to the training distribution, and making a binary feature that is 1 when the value is the same as the instance being explained.

explain_instance() function generates explanations for a prediction. First, we generate neighborhood data by randomly perturbing features from the instance. We then learn locally weighted linear (surrogate) models on this neighborhood data to explain each of the classes in an interpretable way.

[12]:
```
headers=list(X.columns)
explainer = LimeTabularExplainer(X_train.values, feature_names=headers,␣
 ↪discretize_continuous=True,
                                  categorical_features=indices_cat_cols,
                                  class_names=['<= $50K', '> $50K'],verbose=True)
```

8.1 When a person's income <= $50K

Lime shows which features were the most influential in the model taking the correct decision of predicting the person's income as below $50K. The below explanation shows features each contributing to push the model output from the base value (the average model output over the training dataset we passed) to the actual model output. Features pushing the prediction higher are shown in red, those pushing the prediction lower are in green.

Task 3: Please find a person for which the income is <= $50K and the prediction is correct.

---

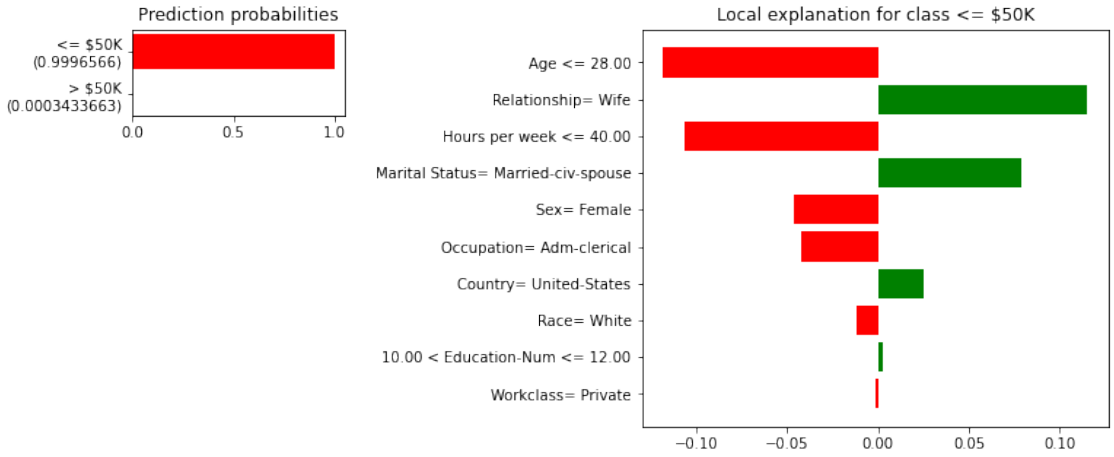
[13]:
```
# Change only the value of to select that person:
i = 1
############

exp1 = explainer.explain_instance(X_test.iloc[i].values, xgc_np.predict_proba,
                                  distance_metric='euclidean',
                                  num_features=len(X_test.iloc[i].values))
proba1 = xgc_np.predict_proba(X_test.values)[i]

print("********************")
print('Test id: ' , i)
print('Probability(',exp1.class_names[0],") =", exp1.predict_proba[0])
print('Probability(',exp1.class_names[1],") =", exp1.predict_proba[1])
print('Predicted Label:', predictions[i])
print('True class: ' , y_test[i])
print("********************")
```

```
Intercept 0.20705968799823424
Prediction_local [0.10207659]
Right: 0.0003433663
********************
Test id:  1
```

```
Probability( <= $50K ) = 0.9996566
Probability( > $50K ) = 0.0003433663
Predicted Label: 0
True class:  0
********************
```

The classifier got this example right (it predicted income <= $50K). Let's have a look at the explanation provided by LIME:

[14]:
```
plot_explanation(exp1, mapping)
```




```
<Figure size 432x288 with 0 Axes>
```

Task 4: Please provide comments on the above explanation provided by LIME.

As noted in task 3, LIME also confirms with this example that age and hours per week are significant features which contribute to the person earning over 50k, here the age is the most importnat factor which groups this sample and hours per week is 2nd most important contributing to this prediction. Apart from this, the figure also shows the thresholds for each variable as found by LIME, these threshold form the boundary line after which the prediction changes, the futher away from this threshold the value, the higher the impact of the feature to affect the prediction. This observation is used in following tasks to change the prediction of the classifier.

Task 5: Please change the value of one or two features the change the prediction of the classifier:

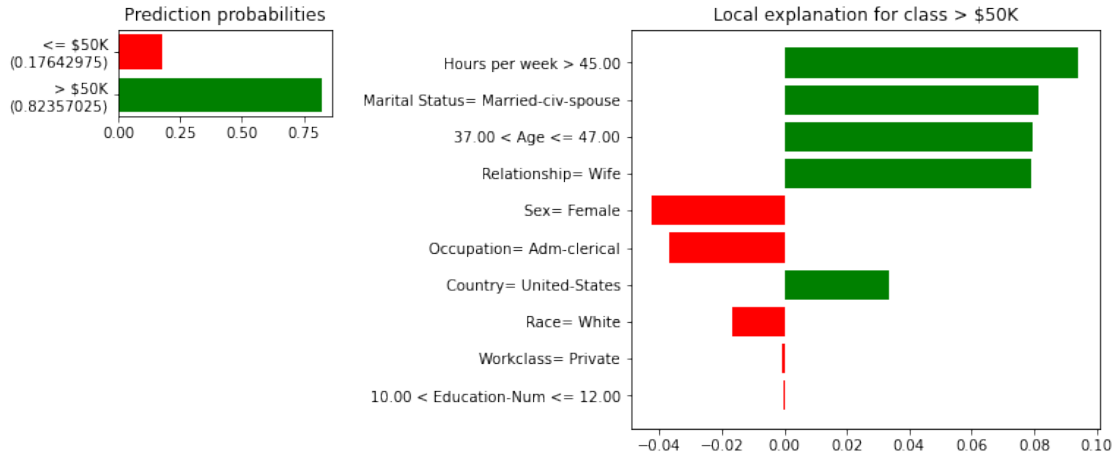
[15]:
```
instanceModified1 = X_test.iloc[i]
instanceModified1['Age'] = 45
instanceModified1['Hours per week'] = 50
```

```
print(instanceModified1)
```

```
Age               45.0
Workclass          4.0
Education-Num     11.0
Marital Status     2.0
Occupation         1.0
Relationship       5.0
Race               4.0
Sex                0.0
Hours per week    50.0
Country           39.0
Name: 26705, dtype: float32
```

[16]:
```
expM1 = explainer.explain_instance(instanceModified1.values, xgc_np.
 ↪predict_proba,
                                   distance_metric='euclidean',
                                  num_features=len(instanceModified1.values))
plot_explanation(expM1, mapping)
```

```
Intercept 0.047528334171081724
Prediction_local [0.31868374]
Right: 0.82357025
```




```
<Figure size 432x288 with 0 Axes>
```

Task 6: How did you choose these features for which you have changed the value? How did you chose these values?

Because the highest ranked features were Age and Hours per week, they contibute significantly in the prediction, hence the classfier now predicts the sample into the

class with annual income over 50k (label 1). The chosen values are on the other end of the threshold found by LIME.

8.2 When a person's income > $50K

Lime shows which features were the most influential in the model taking the correct decision of predicting the person's income as higher $50K. The below explanation shows features each contributing to push the model output from the base value (the average model output over the training dataset we passed) to the actual model output. Features pushing the prediction higher are shown in red, those pushing the prediction lower are in green.

Task 7: Please find a person for which the income is > $50K and the prediction is correct.

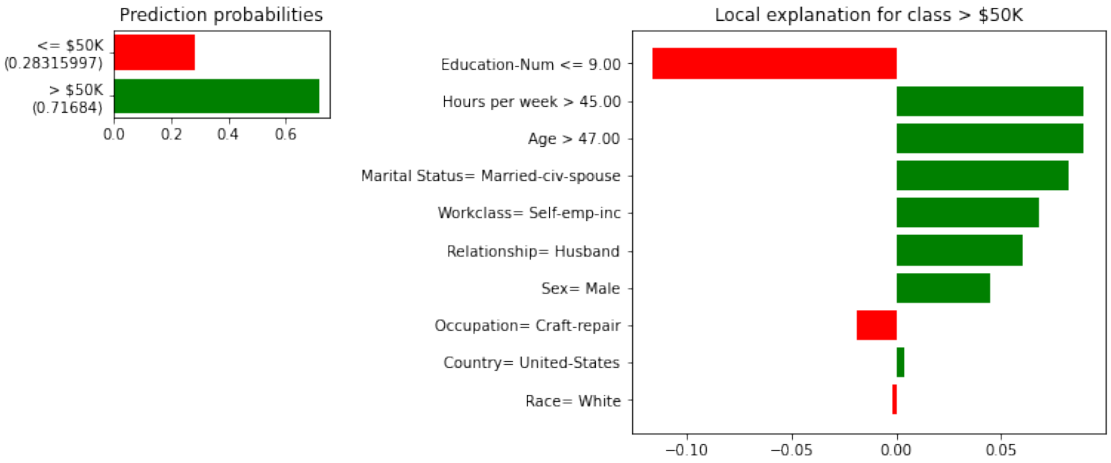
---

[17]:
```
# Change only the value of to select that person:
j = 12
###########

exp2 = explainer.explain_instance(X_test.iloc[j].values, xgc_np.predict_proba,
                                  distance_metric='euclidean',
                                  num_features=len(X_test.iloc[j].values))
proba2 = xgc_np.predict_proba(X_test.values)[j]

print("********************")
print('Test id: ' , j)
print('Probability(',exp2.class_names[0],") =", exp2.predict_proba[0])
print('Probability(',exp2.class_names[1],") =", exp2.predict_proba[1])
print('Predicted Label:', predictions[j])
print('True class: ' , y_test[j])
print("********************")
```

```
Intercept 0.04499000450376432
Prediction_local [0.34624989]
Right: 0.71684
********************
Test id:  12
Probability( <= $50K ) = 0.28315997
Probability( > $50K ) = 0.71684
Predicted Label: 1
True class:  1
********************
```

The classifier got this example right (it predicted income > $50K). Let's have a look at the explanation provided by LIME:

[18]:
```
plot_explanation(exp2, mapping)
```




```
<Figure size 432x288 with 0 Axes>
```

Task 8: Please provide comments on the above explanation provided by LIME.

For this sample, LIME chooses Hours per week and Martial Status to be the top two features that contribute to the classifier predicting this sample as class 1 [income over 50k]

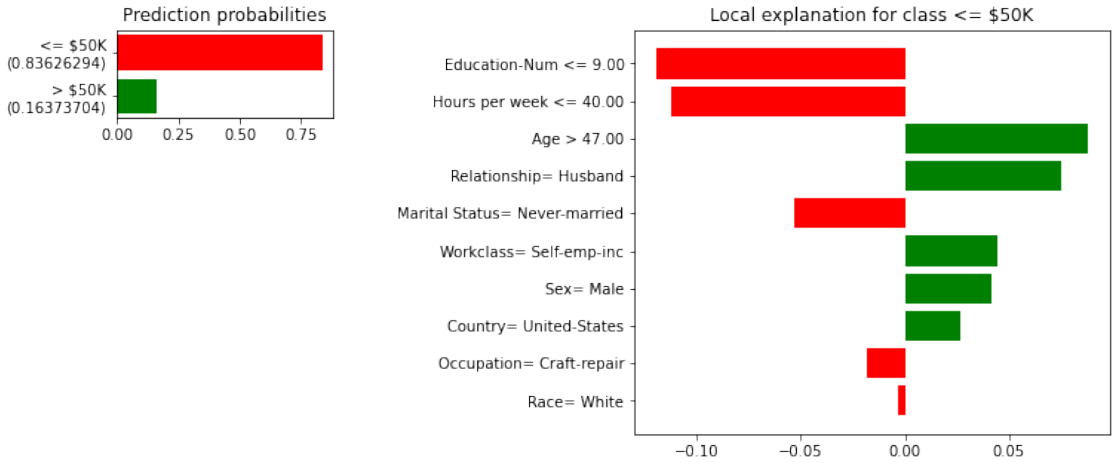
Task 9: Please change the value of one or two features the change the prediction of the classifier:

[19]:
```
instanceModified2 = X_test.iloc[j]
instanceModified2['Hours per week'] = 30
instanceModified2['Marital Status'] = 4.0
print(instanceModified2)
```

```
Age               48.0
Workclass          5.0
Education-Num      9.0
Marital Status     4.0
Occupation         3.0
Relationship       0.0
Race               4.0
Sex                1.0
Hours per week    30.0
Country           39.0
Name: 32462, dtype: float32
```

[20]:
```
expM2 = explainer.explain_instance(instanceModified2.values, xgc_np.
↪predict_proba,
                                   distance_metric='euclidean',
                                  num_features=len(instanceModified2.values))
plot_explanation(expM2, mapping)
```

```
Intercept 0.1817082463492808
Prediction_local [0.15071787]
Right: 0.16373704
```




```
<Figure size 432x288 with 0 Axes>
```

Task 10: How did you choose these features for which you have changed the value? How did you chose these values?

As the chosen features were the highest two contributing features for the predicted class to be > 50k, hence changing their values changes the prediction to the other class. The chosen values are on the other end of the threshold found by LIME.

9 When a person's income actual is different than predicted

Lime shows which features were the most influential in the model taking the incorrect decision of predicting the person's income. The below explanation shows features each contributing to push the model output from the base value (the average model output over the training dataset we passed) to the actual model output. Features pushing the prediction higher are shown in red, those pushing the prediction lower are in green.

Task 11: Please find a person for which the the prediction is **incorrect**.

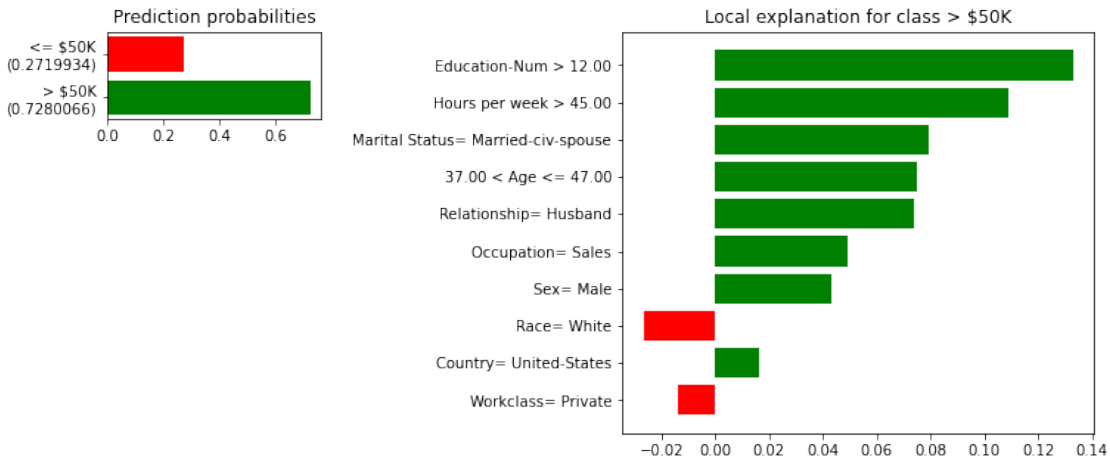
---

[21]:
```
# Change only the value of to select that person:
k = 4
###########

exp3 = explainer.explain_instance(X_test.iloc[k].values, xgc_np.predict_proba,
                                  distance_metric='euclidean',
                                  num_features=len(X_test.iloc[k].values))
proba3 = xgc_np.predict_proba(X_test.values)[k]

print("********************")
print('Test id: ' , k)
print('Probability(',exp3.class_names[0],") =", exp3.predict_proba[0])
print('Probability(',exp3.class_names[1],") =", exp3.predict_proba[1])
print('Predicted Label:', predictions[k])
print('True class: ' , y_test[k])
print("********************")
```

```
Intercept -0.03223714981639614
Prediction_local [0.5061155]
Right: 0.7280066
********************
Test id:  4
Probability( <= $50K ) = 0.2719934
Probability( > $50K ) = 0.7280066
Predicted Label: 1
True class:  0
********************
```

The classifier got this example classified incorrectly. Let's have a look at the explanation provided by LIME:

[22]:
```
plot_explanation(exp3, mapping)
```



```
<Figure size 432x288 with 0 Axes>
```

Task 12: Please provide comments on the above explanation provided by LIME.

According to LIME, Education-num and Hours per week are the top two features which contribute to this incorrect prediction.

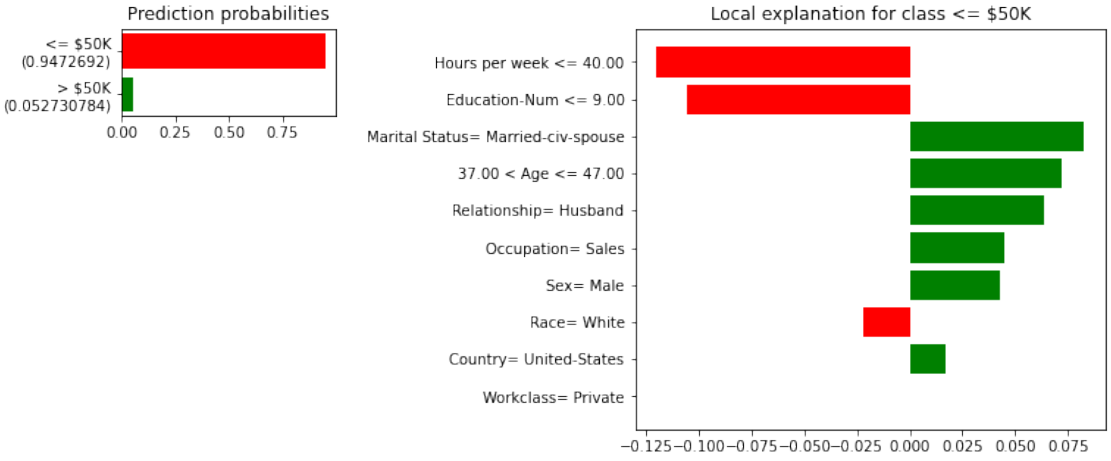
Task 13: Please change the value of one or two features the change the prediction of the classifier (to get a correct prediction):

[23]:
```
instanceModified3 = X_test.iloc[k]
instanceModified3['Education-Num'] = 8
instanceModified3['Hours per week'] = 25
print(instanceModified3)
```

```
Age               42.0
Workclass          4.0
Education-Num      8.0
Marital Status     2.0
Occupation        12.0
Relationship       0.0
Race               4.0
Sex                1.0
Hours per week    25.0
Country           39.0
Name: 26674, dtype: float32
```

[24]:
```
expM3 = explainer.explain_instance(instanceModified3.values, xgc_np.
 ↪predict_proba,
                                    distance_metric='euclidean',
                                   num_features=len(instanceModified3.values))
plot_explanation(expM3, mapping)
```

```
Intercept 0.15105598573782617
Prediction_local [0.2251093]
Right: 0.052730784
```




```
<Figure size 432x288 with 0 Axes>
```

Task 14: How did you choose these features for which you have changed the value? How did you chose these values?

As the chosen features have the highest contribution, hence changing their values make the classifier predict correctly. The values chosen in this task as well as previous task are towards the other end of the spectrum from the threshold found by LIME.

10 Decision Tree

A decision tree is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements.

Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal, but are also a popular tool in machine learning.

Let's use the DecisionTreeClassifier provided by sklearn on our dataset:

[25]:
```
tree = DecisionTreeClassifier(random_state=0, max_depth=4).fit(X_train.values,␣
↪y_train)
tree.fit(X_train.values, y_train)
```

[25]:
```
DecisionTreeClassifier(max_depth=4, random_state=0)
```

[26]:
```
predictions = tree.predict(X_test)
print("The values predicted for the first 20 test examples are:")
print(predictions[:20])

print("The true values are:")
```

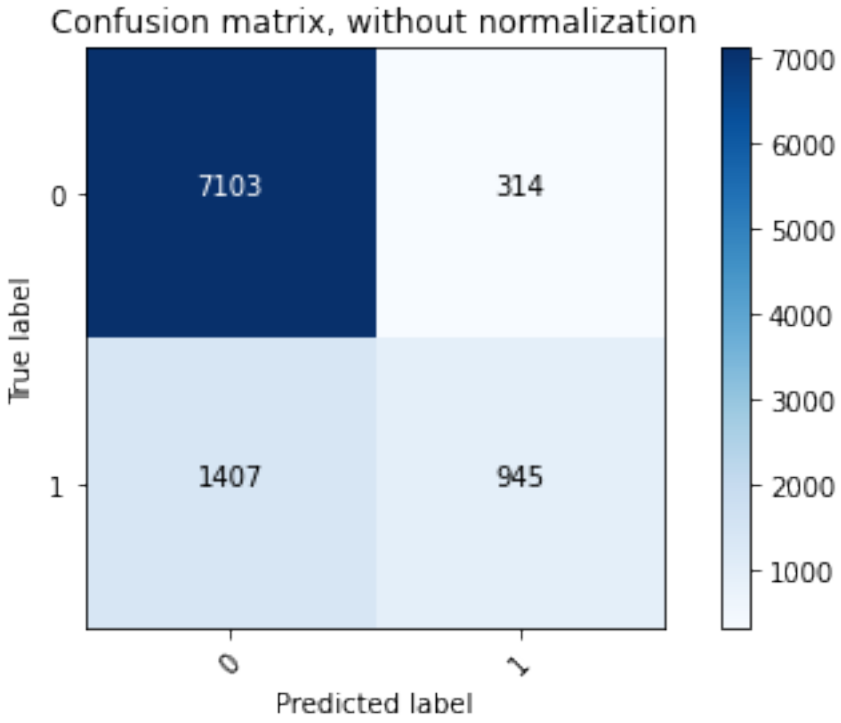
```
print(y_test[:20])
```

```
The values predicted for the first 20 test examples are:
[0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0]
The true values are:
[1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0]
```

[27]:
```
report = classification_report(y_test, predictions)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.83      0.96      0.89      7417
           1       0.75      0.40      0.52      2352

    accuracy                           0.82      9769
   macro avg       0.79      0.68      0.71      9769
weighted avg       0.81      0.82      0.80      9769
```

[28]:
```
class_labels = list(set(labels))
cnf_matrix = confusion_matrix(y_test, predictions)
plot_confusion_matrix(cnf_matrix, classes=class_labels,
                      title='Confusion matrix, without normalization')
```



Task 15: Please provide comments on the performance of the decision Tree.

Decision Tree also shows the same characteristics of being biased (high evaluation metrics) towards the dominant class of the dataset (label 0: income <= 50k).

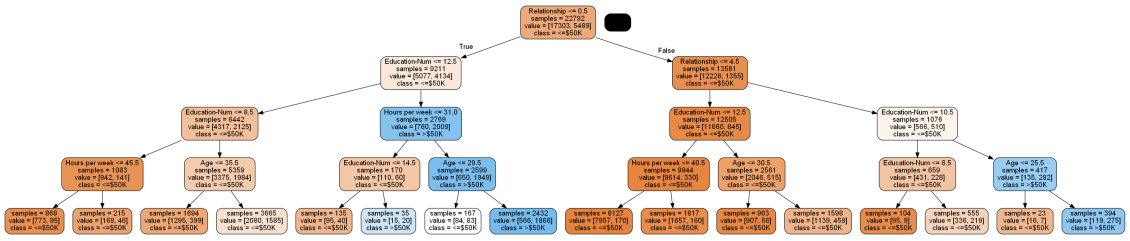
11 Visualzing the Tree

Let's generate a GraphViz representation of the decision tree:

[29]:
```
dot_data = StringIO()
export_graphviz(tree, out_file=dot_data, feature_names=headers,
                filled=True, rounded=True, impurity= False,␣
 ↪class_names=['<=$50K','>$50K'])
```

[30]:
```
graph=pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

[30]:



Task 16: Explain the tree structure (including the meaning of the colors). ##### The tree structure denotes the number of samples in each node, this signifies the subset of samples that have reached that node. The left side denotes the samples precited as true (class <= 50k) whereas the right side denotes false (> 50k) conditions. The leaf nodes denote the target class with most instances based on different splitting criterias. The differetn condtions represetn differtn questions or conditions based on which the decision tree decides to classify the samples.

12 Explanation using LIME

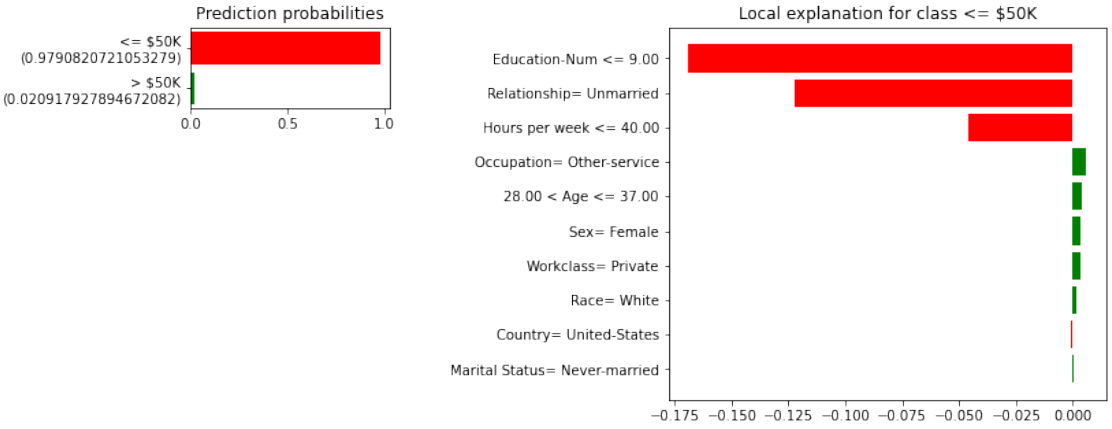
Select any person from the dataset and get the LIME explanation for its classification.

[31]:
```
# Change only the value of to select that person:
h = 8
############

exp4 = explainer.explain_instance(X_test.iloc[h].values, tree.predict_proba,
                                  distance_metric='euclidean',
                                  num_features=len(X_test.iloc[h].values))
```

```
proba1 = tree.predict_proba(X_test.values)[h]

print("**********************")
print('Test id: ' , h)
print('Probability(',exp4.class_names[0],") =", exp4.predict_proba[0])
print('Probability(',exp4.class_names[1],") =", exp4.predict_proba[1])
print('Predicted Label:', predictions[h])
print('True class: ' , y_test[h])
print("**********************")
```

```
Intercept 0.30042130778015363
Prediction_local [-0.019252]
Right: 0.020917927894672082
**********************
Test id:  8
Probability( <= $50K ) = 0.9790820721053279
Probability( > $50K ) = 0.020917927894672082
Predicted Label: 0
True class:  0
**********************
```

[32]:
```
plot_explanation(exp4, mapping)
```




```
<Figure size 432x288 with 0 Axes>
```

Task 17: Please provide comments on the above explanation provided by LIME using on the Tree structure above as a context to your explanation. ##### LIME ranks Education-Num, Relationship and Hours of week as top 3 features resposnible for this instance being classified as class label 0 [income <= 50k], this explanation aligns with the Tree structure as we can see that when education num is < 9, relationship < 4.5 and hours per week < 40 then the sample lies in the class <= 50k.

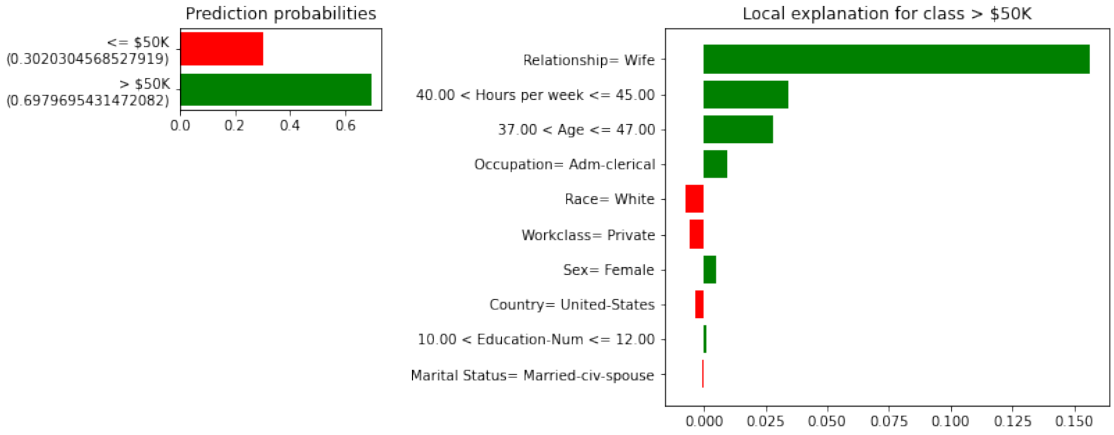
13 Your own test example

Task 18: Following the tree above, create your own test example that will be classified as income > $50K by the decision tree. Explain how you select the values for the features. Use LIME to provide explanation to that test example.

We can see that based on the values chosen from the tree conditions we get a sample which the classifier predicts as > $50K. Here Age, Hours per Week, education num and relationship are assigned values which should result in desired prediction. We can aso see from LIME explanation that Realtionship, Age and Hours per weeek contribute to the highest factors tresposible for the predcition hence proving the decision tree explaiability.

[33]:
```
instanceTest = X_test.iloc[0] #choosing the first sample as starting point

#changnin values based on tree
instanceTest['Age'] = 40
instanceTest['Workclass'] = 4.0
instanceTest['Education-Num'] = 11.0
instanceTest['Marital Status'] = 2.0
instanceTest['Occupation'] = 1.0
instanceTest['Relationship'] = 5.0
instanceTest['Sex'] = 0.0
instanceTest['Hours per week'] = 45
instanceTest['Country'] = 39.0

print(instanceTest)

expM1 = explainer.explain_instance(instanceTest.values, tree.predict_proba,
                                   distance_metric='euclidean',
                                  num_features=len(instanceTest.values))
plot_explanation(expM1, mapping)
```

```
Age               40.0
Workclass          4.0
Education-Num     11.0
Marital Status     2.0
Occupation         1.0
Relationship       5.0
Race               4.0
Sex                0.0
Hours per week    45.0
Country           39.0
Name: 5794, dtype: float32
Intercept 0.18041781086838882
Prediction_local [0.39704288]
Right: 0.6979695431472082
```




```
<Figure size 432x288 with 0 Axes>
```

14 LIME for explaining prediction images

Task 19: Use the CIFAR100 to build an image classifier, and then use the LIME framework with visualization to explain a few predicionts. You can use any classifier for this task (Neural network, Logistic regression, etc.).

I use LimeImageExplainer to display the areas of image that contribute most to the prediction, 5 randomly selected images are displayed to show the explainability for images.

[35]:
```
import random
from lime import lime_image

import tensorflow as tf
from tensorflow.keras.datasets import cifar100
from tensorflow.python.ops.numpy_ops import np_config

from sklearn.ensemble import RandomForestClassifier
from skimage.color import gray2rgb, rgb2gray, label2rgb # since the code wants␣
 ↪color images
from skimage.segmentation import mark_boundaries
np_config.enable_numpy_behavior()

(X_train, y_train), (X_test, y_test) = cifar100.load_data()
X_train = tf.image.rgb_to_grayscale(X_train)
nsamples, nx, ny, nc = X_train.shape
X_train = X_train.reshape((nsamples,nx*ny*nc))
print(X_train.shape)

clf =  RandomForestClassifier(n_estimators=100, max_depth=2,random_state=0)
```

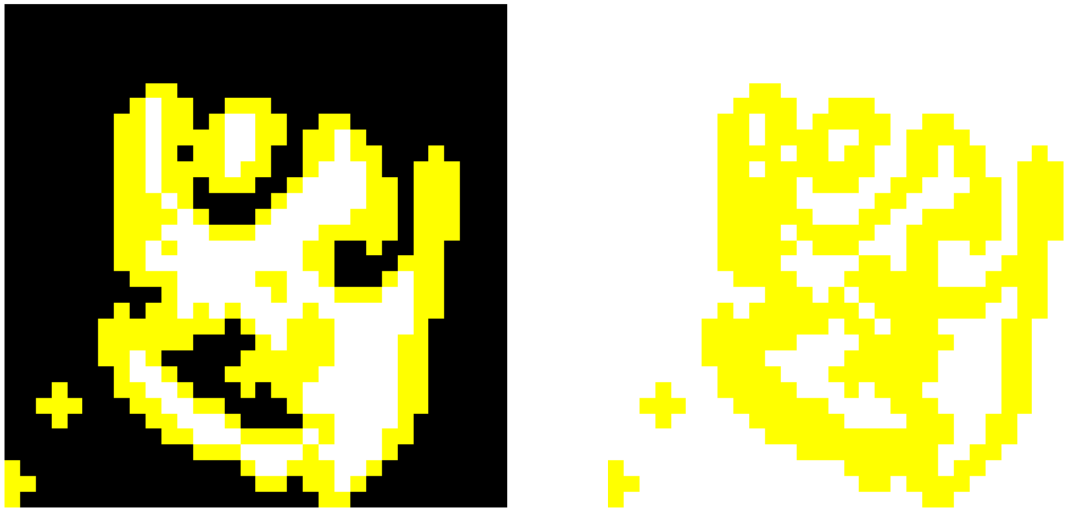
```
clf.fit(X_train, y_train)

explainer = lime_image.LimeImageExplainer()

def custom_pred(imgs):
    tot_probs = []
    for img in imgs:
        grayimg = rgb2gray(img)
        probs = clf.predict_proba(grayimg.reshape(1, -1))[0]
        tot_probs.append(probs)
    return tot_probs



for itr in random.sample(range(0, len(X_test)), 5):
    print (itr)
    converted = tf.image.rgb_to_grayscale(X_test[itr])
    converted = converted.reshape(X_test[itr].shape[0], X_test[itr].shape[1])
    print (converted.shape)

    explanation = explainer.explain_instance(gray2rgb(converted).
 ↪astype('double'), classifier_fn=custom_pred,
                                             top_labels=3, hide_color=0,␣
 ↪num_samples=1000)

    temp_1, mask_1 = explanation.get_image_and_mask(explanation.top_labels[0],
                                                    positive_only=True,
                                                    num_features=5,
                                                    hide_rest=True)

    temp_2, mask_2 = explanation.get_image_and_mask(explanation.top_labels[0],
                                                    positive_only=False,
                                                    num_features=10,
                                                    hide_rest=False)
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15,15))
    ax1.imshow(mark_boundaries(temp_1, mask_1))
    ax2.imshow(mark_boundaries(temp_2, mask_2))
    ax1.axis('off')
    ax2.axis('off')
    plt.show()
```

```
(50000, 1024)
9932
(32, 32)
```

```
  0%|          | 0/1000 [00:00<?, ?it/s]
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

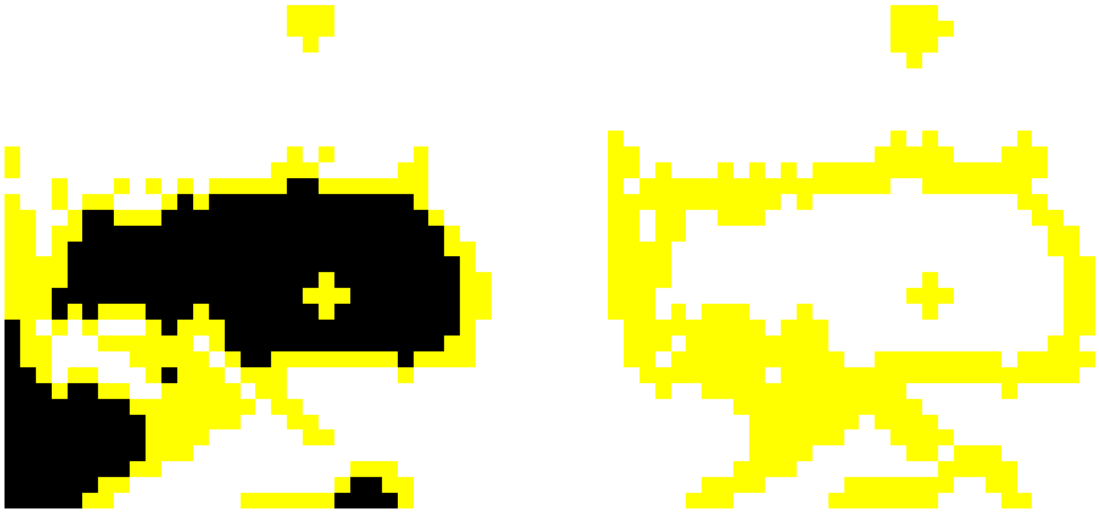
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).



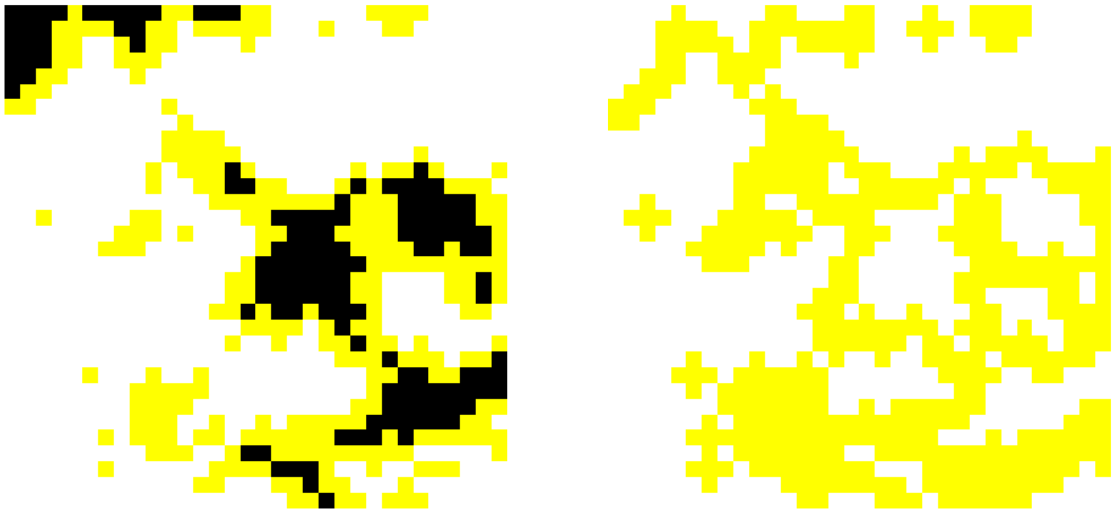
347
(32, 32)

  0%| | 0/1000 [00:00<?, ?it/s]

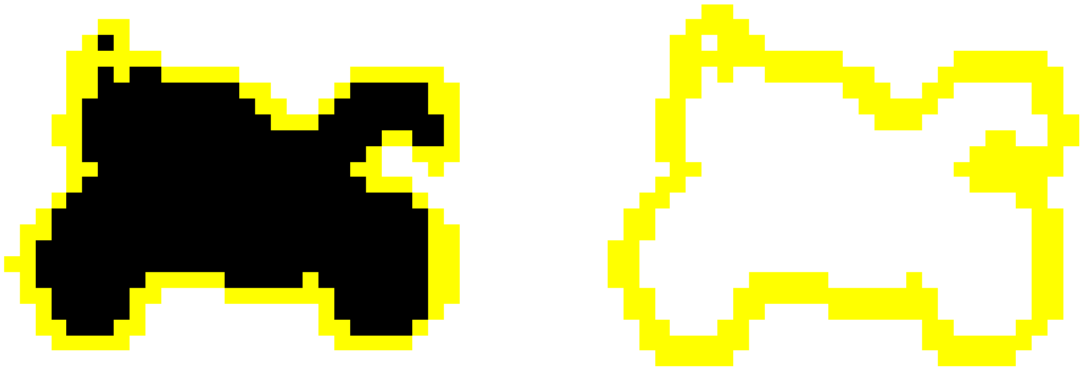
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).



```
4565
(32, 32)
  0%|          | 0/1000 [00:00<?, ?it/s]
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```



```
7109
(32, 32)
  0%|          | 0/1000 [00:00<?, ?it/s]
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```



```
446
(32, 32)
  0%|          | 0/1000 [00:00<?, ?it/s]
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```

15 Congratulations!

You've come to the end of this assignment, and have seen a lot of the ways to explain the predictions given by a classifier.

Congratulations on finishing this notebook!