Random Forest

Group 4: Chris Ng, Jakob Thunen, Alex Mora, Nick Kartschoke

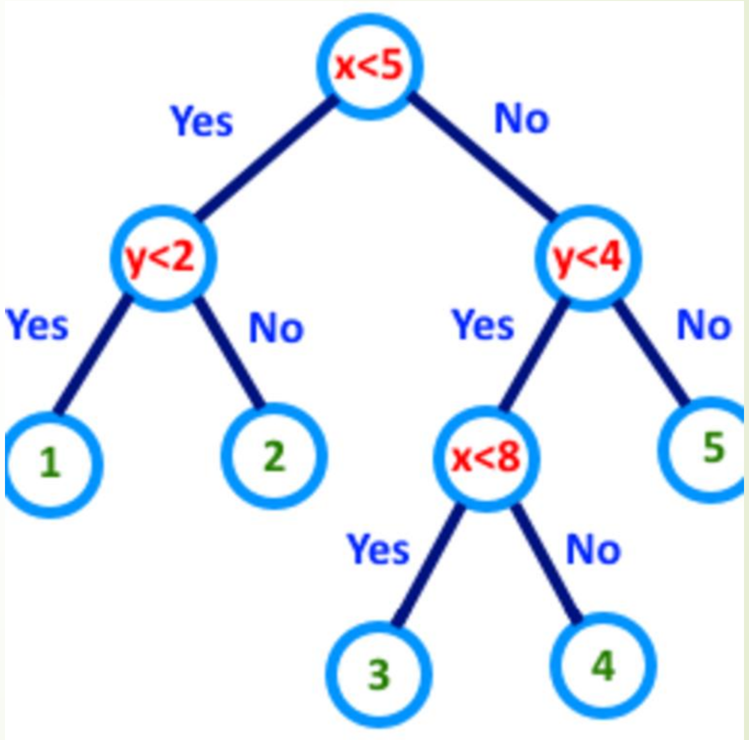
Outline

1. Random Forest Description
2. Explain Data
3. Data Processing
4. Default Random Forest Model
5. Adjusted Random Forest
6. Logistic Regression
7. Comparison (Random Forest vs Logistic)

Random Forest

Random Forest is a machine learning algorithm that is used to solve regression and classification problems

The algorithm creates multiple decision trees during training and makes a prediction for those decision trees



Data

We used the Pima Indians Diabetes Database, available on Kaggle.

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

Data Processing

Changing all 0 values in the dataset to NaN excluding pregnancies and the outcome

```
for i in df.columns:
    df[i] = pd.to_numeric(df[i], downcast='integer', errors='coerce')
df.Glucose = df.Glucose.replace(0, np.nan)
df.BloodPressure = df.BloodPressure.replace(0, np.nan)
df.SkinThickness = df.SkinThickness.replace(0, np.nan)
df.Insulin = df.Insulin.replace(0, np.nan)
df.BMI = df.BMI.replace(0, np.nan)
df.DiabetesPedigreeFunction = df.DiabetesPedigreeFunction.replace(0, np.nan)
df.Age = df.Age.replace(0, np.nan)
```

Changing all NaN values to the mean of the column

```
mode_impute = SimpleImputer(missing_values = np.nan, strategy='mean')
mode_impute.fit(df['Insulin'].values.reshape(-1,1))
df[['Insulin']] = mode_impute.transform(df[['Insulin']])

mode_impute.fit(df['Glucose'].values.reshape(-1,1))
df[['Glucose']] = mode_impute.transform(df[['Glucose']])

mode_impute.fit(df['BloodPressure'].values.reshape(-1,1))
df[['BloodPressure']] = mode_impute.transform(df[['BloodPressure']])

mode_impute.fit(df['SkinThickness'].values.reshape(-1,1))
df[['SkinThickness']] = mode_impute.transform(df[['SkinThickness']])

mode_impute.fit(df['BMI'].values.reshape(-1,1))
df[['BMI']] = mode_impute.transform(df[['BMI']])
```

Updated Data Frame

Note: 0 entries are gone and replaced with the mean.

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.0	35.00000	155.548223	33.6	0.627	50	1
1	1	85.0	66.0	29.00000	155.548223	26.6	0.351	31	0
2	8	183.0	64.0	29.15342	155.548223	23.3	0.672	32	1
3	1	89.0	66.0	23.00000	94.000000	28.1	0.167	21	0
4	0	137.0	40.0	35.00000	168.000000	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101.0	76.0	48.00000	180.000000	32.9	0.171	63	0
764	2	122.0	70.0	27.00000	155.548223	36.8	0.340	27	0
765	5	121.0	72.0	23.00000	112.000000	26.2	0.245	30	0
766	1	126.0	60.0	29.15342	155.548223	30.1	0.349	47	1
767	1	93.0	70.0	31.00000	155.548223	30.4	0.315	23	0

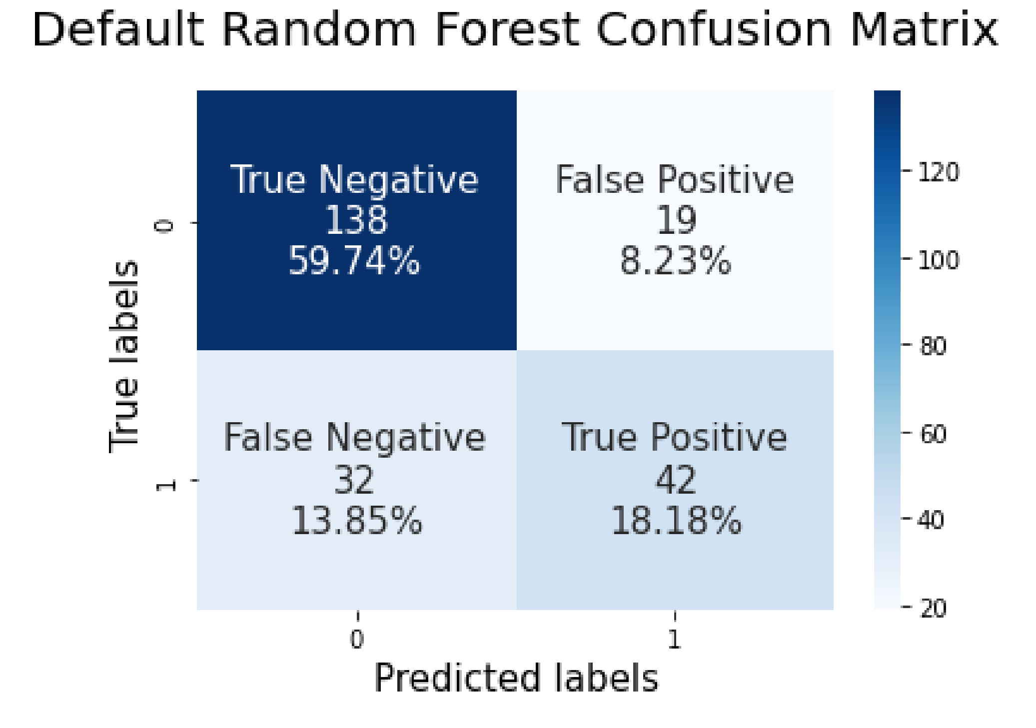
Random Forest (Default Parameters)

The mean accuracy is about 77.92%

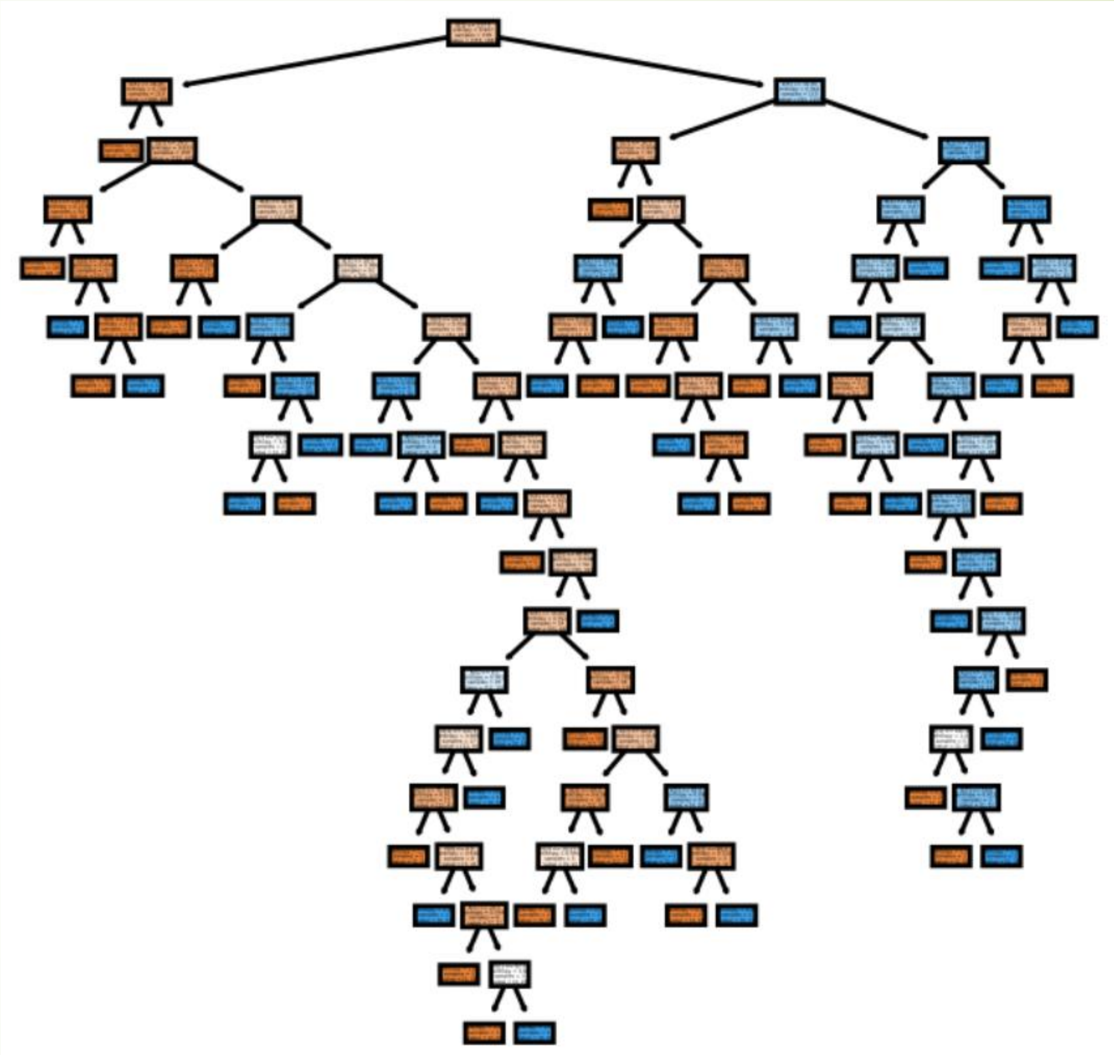
Default Parameters:

- N_estimators =100
- Max_depth = None
- ETC

```
#First Random Forest with default parameters
clf = RandomForestClassifier(criterion='entropy',random_state=0)
```



- Lots of Nodes
- Complex

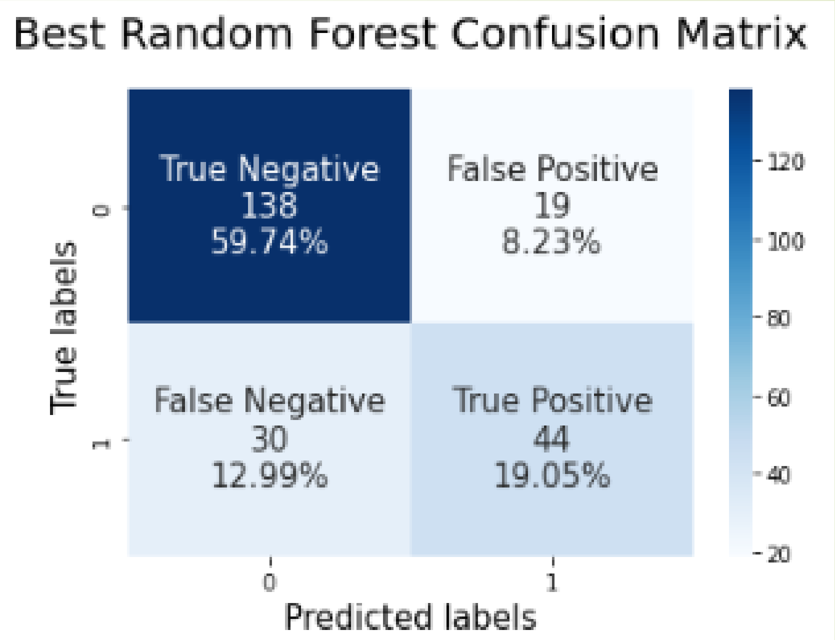


Random Forest (Adjusted Parameters)

Adjusting the hyper parameters allowed for greater accuracy. With an accuracy of **78.79%** this model proved to be the best outcome.

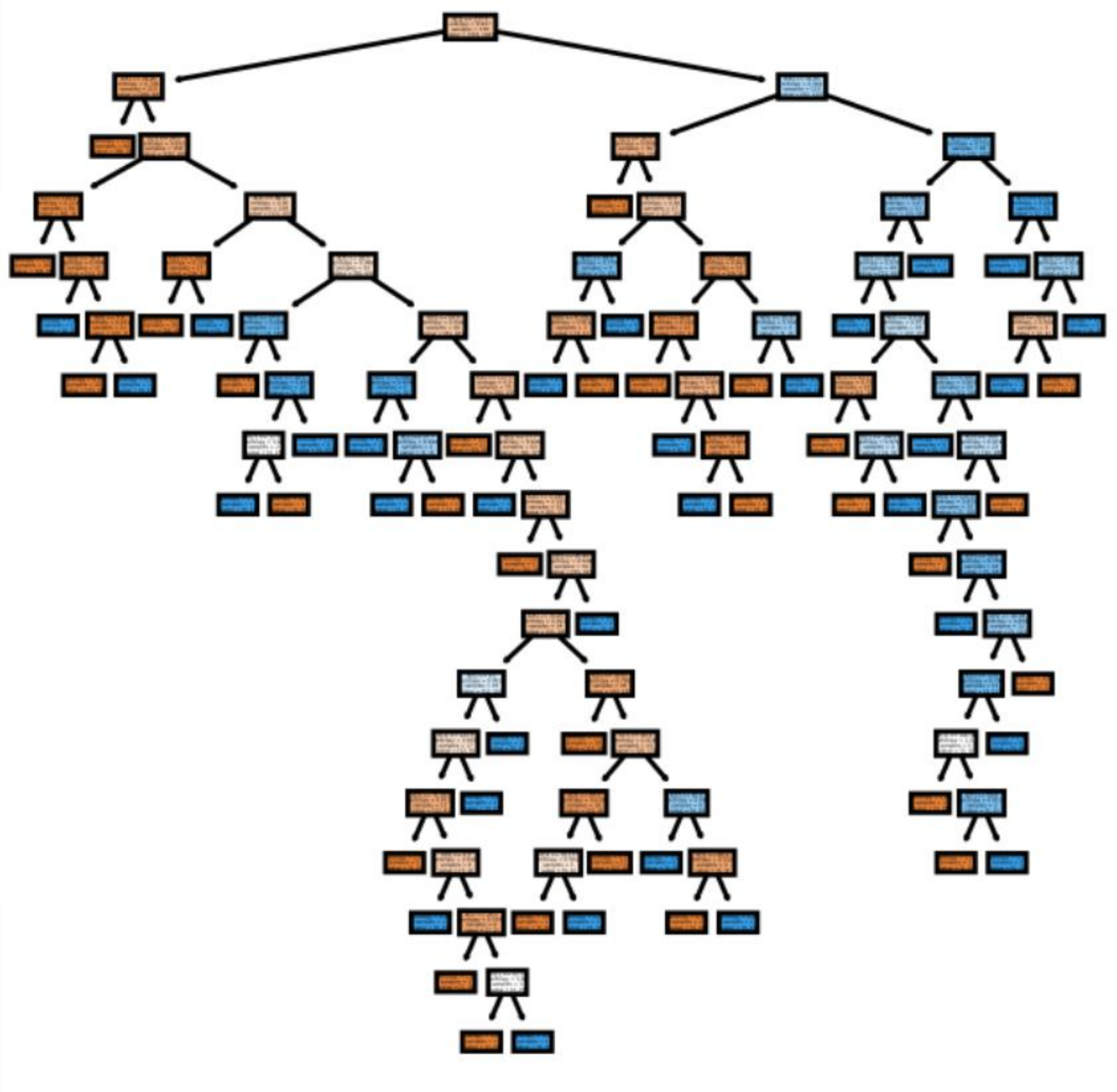
Attempted Parameters include:

- Booststrap
- max_depth
- min_samples_leaf
- max_leaf_nodes
- n_estimators



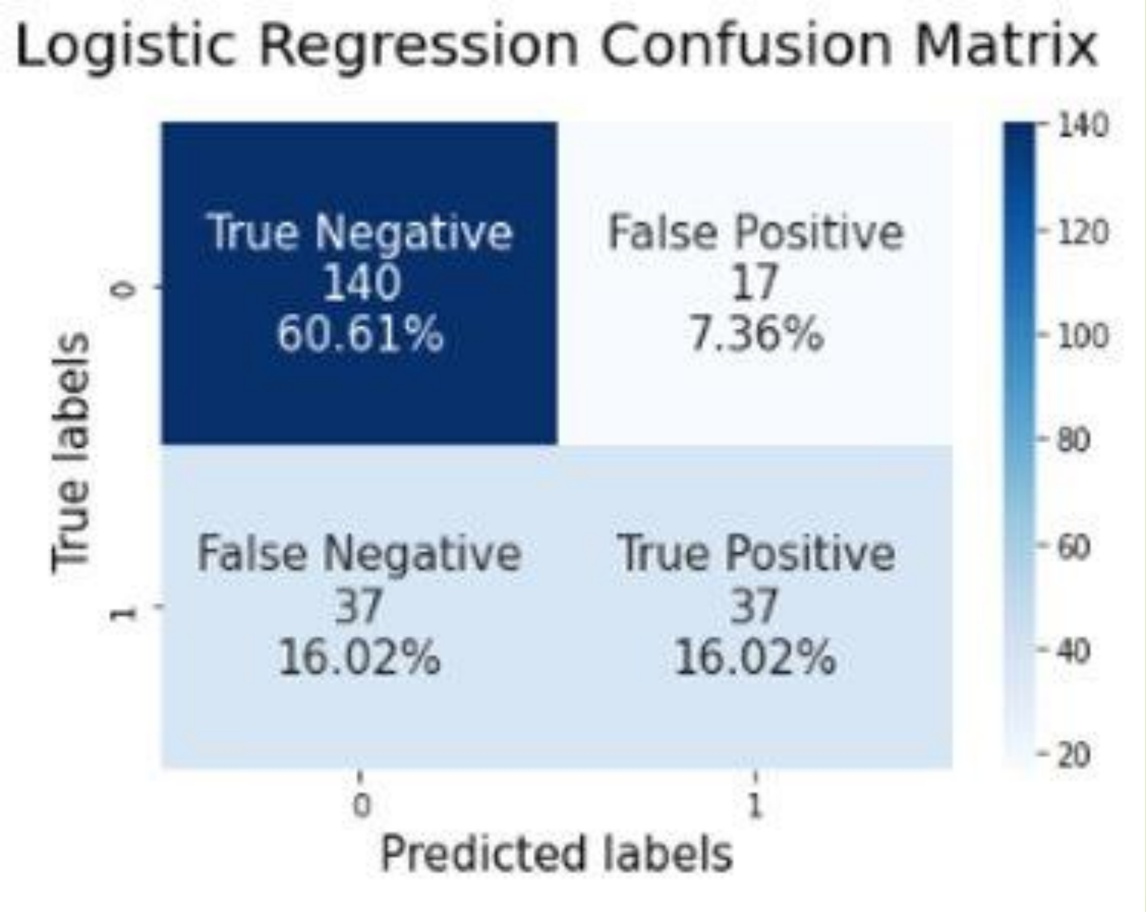

```
#Testing diff Param
#Best Model for random forest
clf2 = RandomForestClassifier(criterion='entropy',random_state=0,max_features=8)
```

- Pregnancies attribute is at the top
- Complex



Logistic Regression

This CM represents the results of an optimized Logistic Regression model run on the same dataset, prepared the same way. It achieved an average score of 76.62%, erring on the side of positive results.




```
#Logistic Regression
from sklearn.linear_model import LogisticRegressionCV
clf3 = LogisticRegressionCV(cv=5, random_state=0,max_iter=500,Cs=8).fit(X_train,y_train)
```

```
#Logistic Regression Coefficients
clf3.coef_
```

```
array([[ 0.09310639,  0.03785255, -0.01796133, -0.00796755, -0.00068314,
         0.10657938,  0.47945333,  0.02442719]])
```

```
clf3.intercept_
```

```
array([-8.68191115])
```

Comparison

- Overall adjusting the hyper-parameter max_features in the Random Forest model increased the accuracy by 0.87%, which isn’t much but nonetheless an improvement.
- When comparing the Adjusted Random Forest to the logistic regression we can see that the Random Forest has a higher accuracy by 2.17%
- Logistic Regression is less complex than Random Forest