Classification on Immunotherapy data set

Munkhsukh Munkhtsetseg

Eötvös Loránd University, H-1053 Budapest, Egyetem tér 1-3
https://www.elte.hu/

Abstract. The goal of this paper is to contribute to a clear understanding of a classification task on Immunotherapy data set. We argue that the classification task can be considered different machine learning algorithms and techniques to improve our result of ML algorithms.

Keywords: Machine learning · Immunotherapy data set · Classification.

1 Introduction to Data set

1.1 Description

Immunotherapy dataset [1] comes from the UCI Machine Learning repository, and the dataset contains information about wart treatment results of 90 patients using immunotherapy. For number of records, It is too low number to train ML model though, We will examine to create a back-bone.

Table 1. Immunotherapy data set description table

Data Set Characteristics:	Univariate	Number of Instances:	90
Attribute Characteristics:	Integer, Real	Number of Attributes:	8
Associated Tasks:	Classification	Missing Values?	N/A
Area:	Life	Date Donated	2018-01-04

Table 2. Immunotherapy data table description

	count	mean	std	min	25%	50%	75%	max
sex	90.0	1.544444	0.500811	1.0	1.00	2.00	2.0000	2.0
age	90.0	31.044444	12.235435	15.0	20.25	28.50	41.7500	56.0
Time	90.0	7.230556	3.098166	1.0	5.00	7.75	9.9375	12.0
Number_of_Warts	90.0	6.144444	4.212238	1.0	2.00	6.00	8.7500	19.0
Type	90.0	1.711111	0.824409	1.0	1.00	1.00	2.0000	3.0
Area	90.0	95.700000	136.614643	6.0	35.50	53.00	80.7500	900.0
induration_diameter	90.0	14.333333	17.217707	2.0	5.00	7.00	9.0000	70.0
Result_of_Treatment	90.0	0.788889	0.410383	0.0	1.00	1.00	1.0000	1.0

It includes 8 fields and sex, type and result of treatment fields are categorical.Other age, time, number of warts, area and induration diameter fields are numerical. In sex field, 1 and 2 represents male and female, and in result of treatment field, 1 means treatment succeeded, and 0 means opposite. Last one, there are 3 types of treatment.

1.2 Data exploration

This section we will explore the results.

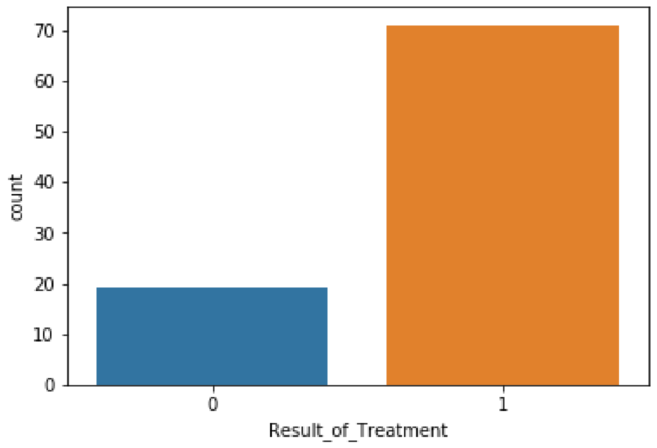


Fig. 1. In our data set, 71 treatment has passed, and 19 ones are failed.

The ratio is 71:19, which means our classes are significantly unbalanced.

Table 3. Categorical means by result of treatment and difference between them.

	0	1	Difference
sex	1.526316	1.549296	0.022980
age	35.473684	29.859155	-5.614529
Time	9.381579	6.654930	-2.726649
Number_of_Warts	6.526316	6.042254	-0.484062
Type	1.578947	1.746479	0.167532
Area	84.315789	98.746479	14.430689
induration_diameter	15.368421	14.056338	-1.312083

Observation Table 3 gives following observations. Younger people tend to pass treatment, while less time and bigger area can give us better results.

1.3 Visualization

In order to explore more, lets see the distribution of data and the correlations.

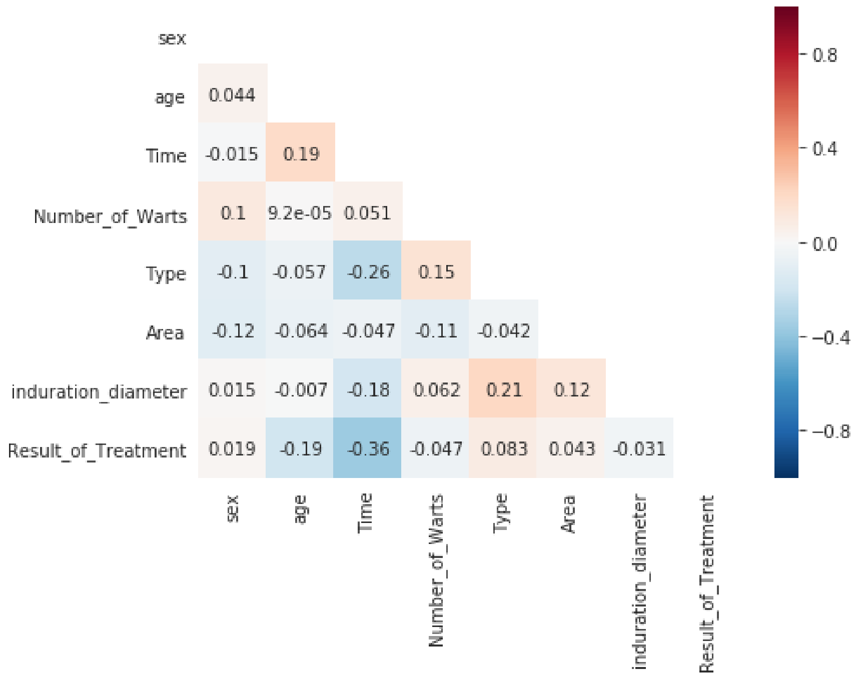


Fig. 2. Correlation diagram of data table

Observation Data columns with very similar trends are also likely to carry very similar information. Here we calculate the correlation coefficient between numerical and nominal columns as the Coefficient. Here we can Time field has high negative correlations with result and type fields. We may use this diagram to reduce features later.

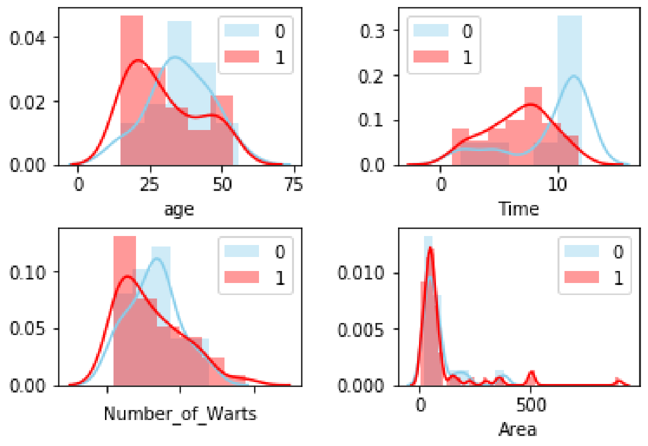


Fig. 3. Histogram diagrams in age, time, number of warts and area by the result of treatment. Age, time and number of warts might be strong predictors.

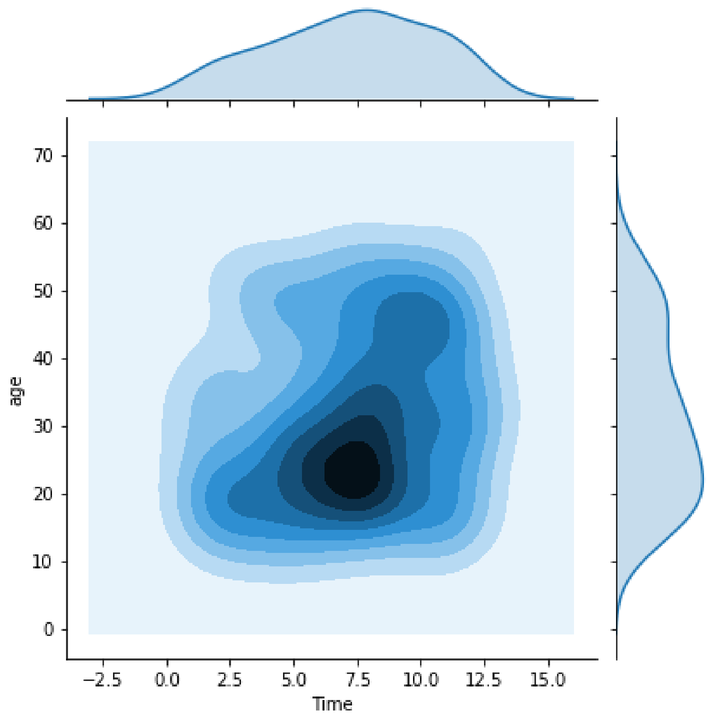


Fig. 4. Histogram diagrams in age, time and their relationship. As we see, around 7.5 and 20 is the most frequent time and age.

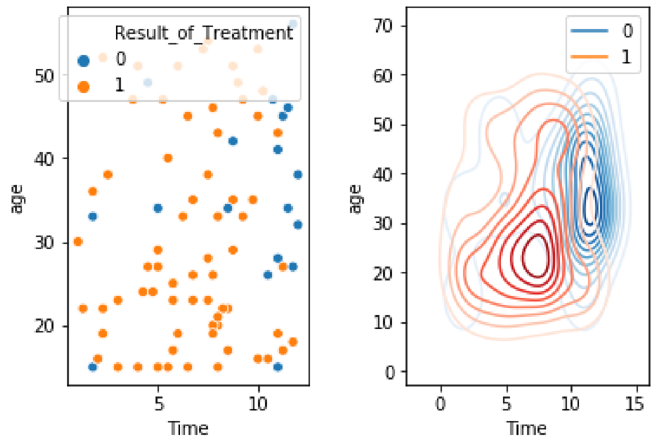


Fig. 5. Here we can see their relationship on the result of treatment. Age and time fields can be strongest predictors.

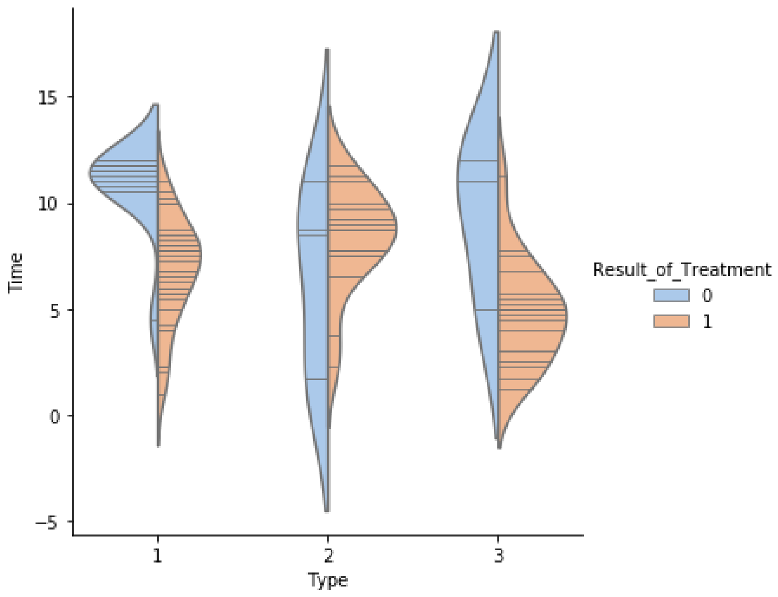


Fig. 6. As the correlation diagram, Time and type had a high score. Here, we can see their relation.

2 Model

Train/Test Split First of all, the data we use is usually split into training data and test data. The training set contains a known output and the model learns on this data in order to be generalized to other data later on. We have the test dataset (or subset) in order to test our model's prediction on this subset.
Here, we split our samples into 20/80 ratio test and training samples, respectively.

2.1 Decision tree

Given a data of attributes together with its classes, a decision tree produces a sequence of rules that can be used to classify the data.

Description: Decision Tree [2], as it name says, makes decision with tree-like model. It splits the sample into two or more homogeneous sets (leaves) based on the most significant differentiators in your input variables. To choose a differentiator (predictor), the algorithm considers all features and does a binary split on them (for categorical data, split by cat; for continuous, pick a cut-off threshold). It will then choose the one with the least cost (i.e. highest accuracy), and repeats recursively, until it successfully splits the data in all leaves (or reaches the maximum depth).
Another reason I chose decision tree model is easy to visualize and examine model. Following figure show us the result.

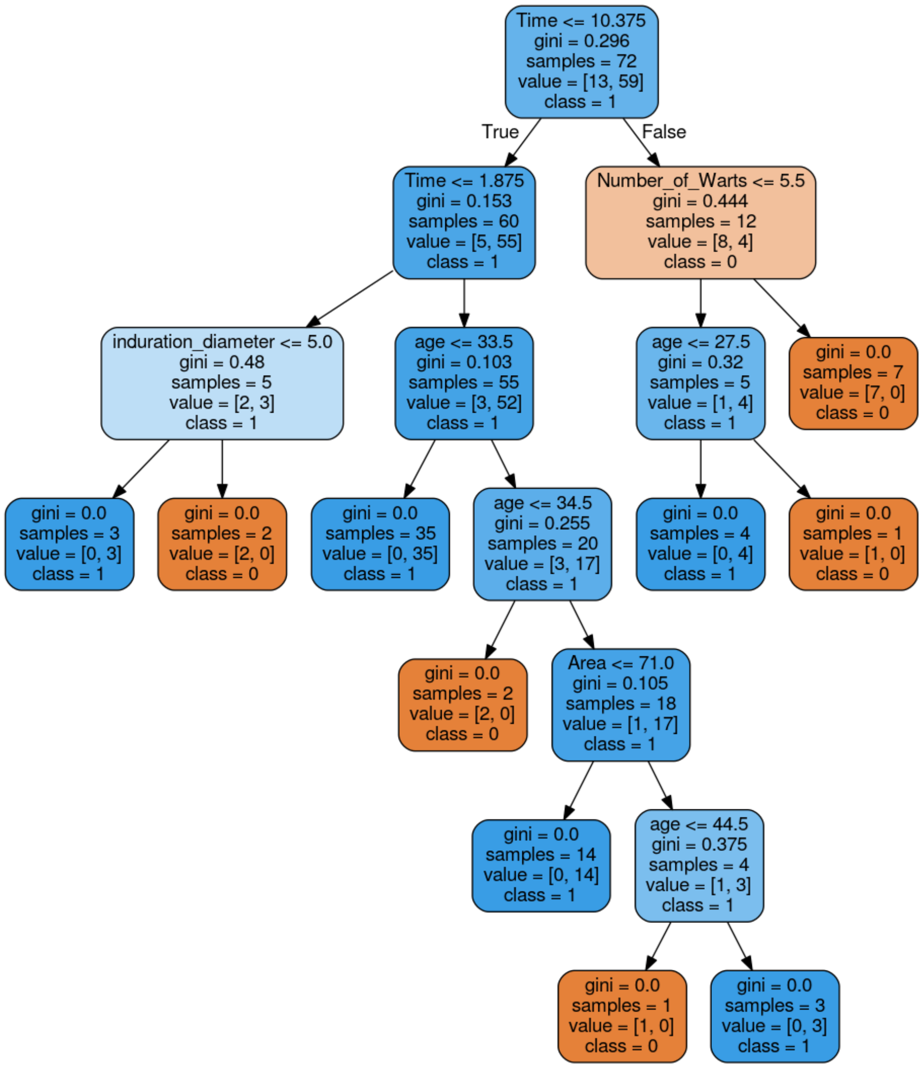


Fig. 7. Decision tree model after trained with our training data.

2.2 Logistic regression

Logistic regression is a classification algorithm used to assign observations to a discrete set of classes. We will use binary logistic regression. In order to map predicted values to probabilities, we use the sigmoid function. The function maps any real value into another value between 0 and 1. In machine learning, we use sigmoid to map predictions to probabilities.

$$S(z) = \frac{1}{1+e^{-z}} \tag{1}$$

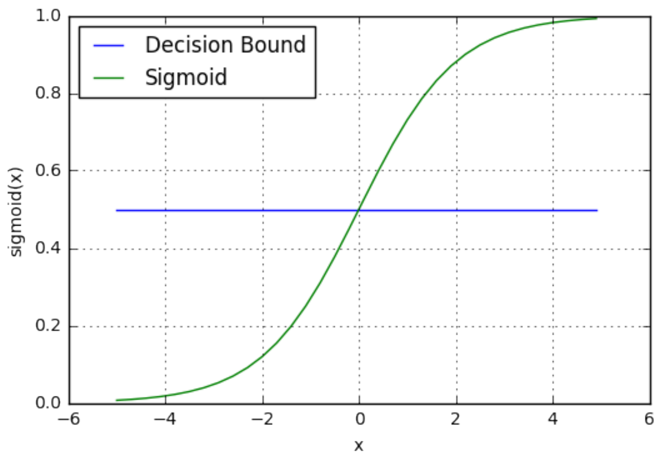


Fig. 8. Sigmoid function

Apply Sigmoid function on linear regression:

$$S(z) = \frac{1}{1+e^{-b_0+b_1x_1+b_2x_2+..b_nx_n}} \tag{2}$$

If the weighted sum of inputs is greater than zero, the predicted class is 1 and vice-versa. So the decision boundary separating both the classes can be found by setting the weighted sum of inputs to 0.

Cost function https://www.overleaf.com/project/5e308d9f1394c60001211bc5 The cost function for a single training example can be given by:

$$cost = \begin{cases} -log(h(x)) & \text{if } y = 1, \\ -log(1-h(x)) & \text{if } y = 0 \end{cases}$$

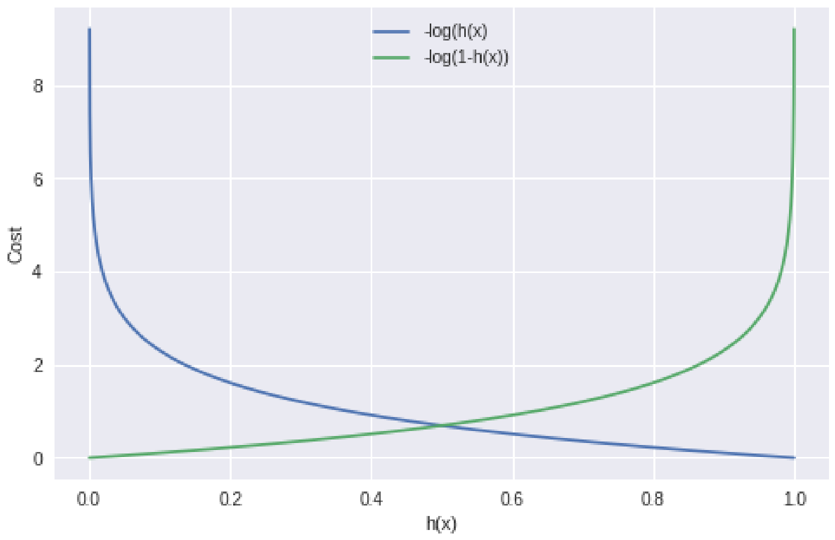


Fig. 9. Cost function

If the actual class is 1 and the model predicts 0, we should highly penalize it and vice-versa. We can combine both of the equations using:

$$cost(h(x), y) = -ylog(h(x)) - (1 - y)log(1 - h(x))$$

The cost for all the training examples denoted by J() can be computed by taking the average over the cost of all the training samples

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^i log(h(x^i)) + (1 - y^i)log(1 - h(x^i))]$$

where m is the number training samples. We will use gradient descent to minimize the cost function.

```
    Optimization terminated successfully.
         Current function value: 0.473337
         Iterations 6
                          Results: Logit
==================================================================
Model:               Logit              Pseudo R-squared: -0.002
Dependent Variable:  Result_of_Treatment AIC:              82.1605
Date:                2020-01-30 13:18   BIC:              98.0972
No. Observations:    72                 Log-Likelihood:   -34.080
Df Model:            6                  LL-Null:          -34.001
Df Residuals:        65                 LLR p-value:      1.0000
Converged:           1.0000             Scale:            1.0000
```

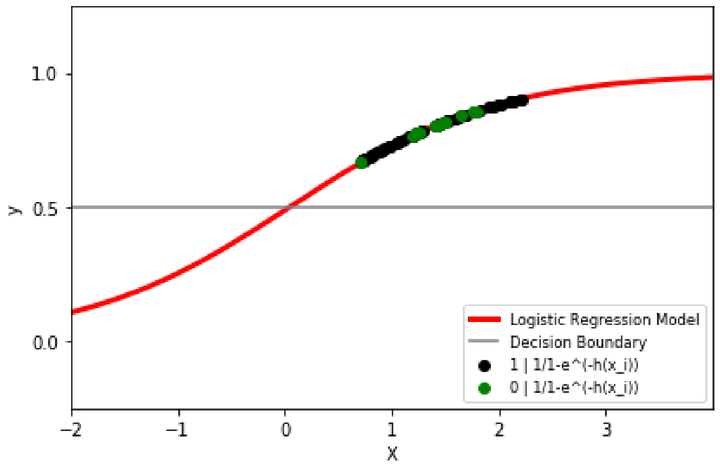


Fig. 10. Sigmoid function on on our training samples

```
No. Iterations:      6.0000
------------------------------------------------------------------
                     Coef.  Std.Err.    z    P>|z|   [0.025 0.975]
------------------------------------------------------------------
sex                  0.9095   0.5196  1.7504 0.0800 -0.1089 1.9279
age                  0.0210   0.0270  0.7771 0.4371 -0.0320 0.0740
Time                -0.1318   0.0956 -1.3794 0.1678 -0.3192 0.0555
Number_of_Warts     -0.0218   0.0741 -0.2938 0.7689 -0.1669 0.1234
Type                 0.2155   0.3432  0.6279 0.5300 -0.4572 0.8882
Area                 0.0007   0.0024  0.2703 0.7869 -0.0041 0.0054
induration_diameter  0.0023   0.0213  0.1078 0.9142 -0.0394 0.0440
==================================================================
```

Now we tried our own code to predict logistic regression with Immunotherapy data set. Later we will use scikit-learn library and compare other algorithms.

```
iter: 0  cost: 1.8732674100249116
iter: 100  cost: 0.6522403704435823
iter: 200  cost: 0.609406517463262
iter: 300  cost: 0.5777195777911466
iter: 400  cost: 0.5551677781320082
iter: 500  cost: 0.5395701581670956
iter: 600  cost: 0.5289746327726486
iter: 700  cost: 0.5218382544817948
iter: 800  cost: 0.517035287249884
iter: 900  cost: 0.5137841005838157
```

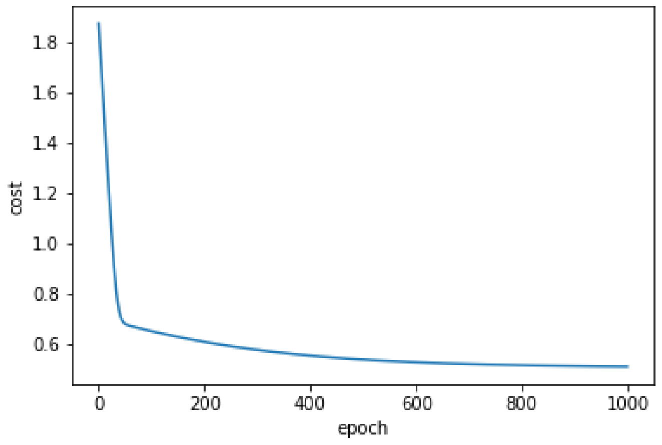
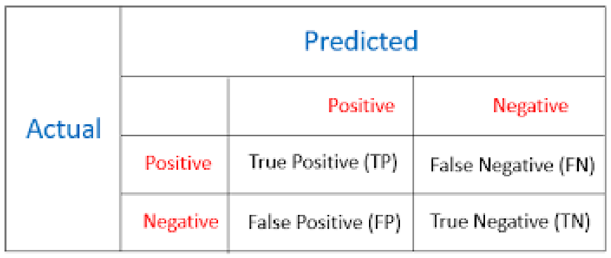


Fig. 11. Training process

2.3 Evaluation of Model

Confusion matrix A confusion matrix is a table that is often used to describe the performance of a classification model on a set of test data for which the true values are known.

– true positives (TP): These are cases in which we predicted yes and are actually yes.
– true negatives (TN): We predicted no, and no in actual.
– false positives (FP): We predicted yes, but actual is no. (Type I error)
– false negatives (FN): We predicted no, yes in actual. (Type II error)



		Predicted	
Actual		Positive	Negative
	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

Fig. 12. Confusion matrix

– Accuracy: Overall, how often is the classifier correct?

$$Accuracy = (TP + TN)/total$$

- Misclassification Rate(Error Rate): Overall, how often is it wrong?

$$MisclassificationRate = (FP + FN)/total$$

- True Positive Rate(Sensitivity or Recall): When it's actually yes, how often does it predict yes?

$$TruePositiveRate = TP/(actual_yes)$$

- False Positive Rate: When it's actually no, how often does it predict yes?

$$FalsePositiveRate = FP/(actual_no)$$

- True Negative Rate(Specificity): When it's actually no, how often does it predict no?

$$TrueNegativeRate = TN/actual_no$$

- Precision: When it predicts yes, how often is it correct?

$$Precision = TP/predictedyes$$

- Prevalence: How often does the yes condition actually occur in our sample?

$$Prevalence = actualyes/total$$

Cross Validation In k-fold cross-validation, the original sample is randomly partitioned into k equal sized subsamples. Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining k 1 subsamples are used as training data. The cross-validation process is then repeated k times, with each of the k subsamples used exactly once as the validation data. The k results can then be averaged to produce a single estimation. The advantage of this method over repeated random sub-sampling (see below) is that all observations are used for both training and validation, and each observation is used for validation exactly once.

```python
from sklearn.model_selection import cross_validate
from sklearn import metrics

def scoringModel(model, train, y):
    scores = cross_validate(model, train, y, scoring=scoring,
     cv=5)
    return scores

scoreTreeModel = scoringModel(decisionTreeModel,train_X,
    train_y)
scoreLR = scoringModel(logisticRegressionModel,train_X,
    train_y)

```

```
data = {'Model' : ['Decision tree', 'Logistic regression']}

for metric_name in scoreTreeModel.keys():
    data[metric_name] = [scoreTreeModel[metric_name].mean(),
    scoreLR[metric_name].mean()]

df = pd.DataFrame(data)
df.T
```

Listing 1.1. Scoring model function code

Because of low amount of data, we used Cross validation [4].

Table 4. Evaluation of models.

Model	Decision tree	Logistic regression
fit_time	0.00423088	0.00406737
score_time	0.0128297	0.0135069
test_accuracy	0.845275	0.790989
test_precision_macro	0.660598	0.407949
test_recall_macro	0.665909	0.482576
test_f1_weighted	0.82632	0.724681
test_roc_auc	0.665909	0.291919

3 Improvement

3.1 Normalization

Standardize features by removing the mean and scaling to unit variance

The standard score of a sample x is calculated as:

$$z = (x - u)/s$$

where u is the mean of the training samples or zero if $with_mean = False$, and s is the standard deviation of the training samples or one if $with_std = False$. After normalization:

After normalization, fitting and scoring time increased and other performances are increased by little.

3.2 Feature reduction

There are lot of techniques to reduce features. But we mentioned before, High correlated features are selected now along the correlation diagram such as age, time, type and induration$_diameter$.

Table 5. Evaluation of models after Normalization

Model	Decision tree	Logistic regression
fit_time	0.00378752	0.0033298
score_time	0.0120449	0.0117829
test_accuracy	0.845275	0.776703
test_precision_macro	0.660598	0.406667
test_recall_macro	0.665909	0.474242
test_f1_weighted	0.82632	0.716681
test_roc_auc	0.665909	0.502273

Table 6. Evaluation of models after Normalization and feature reduction

Model	Decision tree	Logistic regression
fit_time	0.00363851	0.003368
score_time	0.0122293	0.0125299
test_accuracy	0.860659	0.805275
test_precision_macro	0.66188	0.409048
test_recall_macro	0.675	0.490909
test_f1_weighted	0.834291	0.732066
test_roc_auc	0.675	0.622222

Conclusion and Future work Even though we have too less number of samples for this work, We analyzed data and learned classification task. Moreover, Data over-sampling, finding outliers, feature engineering and hyper-parameter tuning can be helped to improve accuracy and performance. Most importantly, we should gather more samples for this case.

References

1. UCI Immunotherapy data set, https://archive.ics.uci.edu/ml/datasets/Immunotherapy+Dataset. Last accessed 4 Jan 2020
2. Classification Algorithms in Machine Learning, https://medium.com/datadriveninvestor/classification-algorithms-in-machine-learning-85c0ab65ff4. Last accessed 4 Jan 2020
3. Logistic regression, https://ml-cheatsheet.readthedocs.io/en/latest/logistic$_regression.html.Lastaccessed4Jan$2020
4. Cross Validation https://en.wikipedia.org/wiki/Cross-validation$_(statistics).Lastaccessed4Jan$2020
5. Building A Logistic Regression in Python, https://towardsdatascience.com/building-a-logistic-regression-in-python-step-by-step-becd4d56c9c8. Last accessed 4 Jan 2020

Appendix A

```
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import mean_absolute_error
import graphviz
from IPython.display import Image

decisionTreeModel = DecisionTreeClassifier(random_state=1)
decisionTreeModel.fit(train_X, train_y)

dot_data = tree.export_graphviz(decisionTreeModel, out_file=
    None,
     feature_names = train_X.columns.values.tolist(),
     class_names= ['0','1'],
     filled=True,
     rounded=True
     )
graph = graphviz.Source(dot_data)
from IPython.display import SVG
Image(graph.pipe(format='png'))
```

Listing 1.2. Decision tree model visualization

```
from sklearn.model_selection import train_test_split

X = data.loc[:, data.columns != 'Result_of_Treatment']
y = data.loc[:, 'Result_of_Treatment']
train_X, test_X, train_y, test_y = train_test_split(X, y,
    test_size=0.2, random_state=1)
#train_X, val_X, train_y, val_y = train_test_split(train_X,
    train_y, test_size=0.2, random_state=1)

from sklearn.linear_model import LogisticRegression

LR = LogisticRegression()
scoreLR = scoring(LR)

data = {'Model' : ['Decision tree', 'Logistic regression']}

for metric_name in scoreTreeModel.keys():
     data[metric_name] = [scoreTreeModel[metric_name].mean(),
    scoreLR[metric_name].mean()]

df = pd.DataFrame(data)
df.T
```

Listing 1.3. Training Logistic regression and Decision tree model

```

#check null values
cols_with_missing = [col for col in data.columns
                     if data[col].isnull().any()]

pd.plotting.register_matplotlib_converters()
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

sns.countplot(data['Result_of_Treatment'],label="Sum")
data['Result_of_Treatment'].value_counts()
#TODO We can use SMOTE algorithm for oversampling

pd.concat([data.groupby('Result_of_Treatment').mean(), data.
    groupby('Result_of_Treatment').mean().diff().dropna()]).T

ax = sns.distplot( data[data['Result_of_Treatment'] == 0]['
    age'] , color="skyblue", label="0")
ax = sns.distplot( data[data['Result_of_Treatment'] == 1]['
    age'] , color="red", label="1")
ax.legend()

ax = sns.distplot( data[data['Result_of_Treatment'] == 0]['
    Time'] , color="skyblue", label="0")
ax = sns.distplot( data[data['Result_of_Treatment'] == 1]['
    Time'] , color="red", label="1")
ax.legend()

sns.catplot(x="Result_of_Treatment", y="Time", kind = 'violin
    ', palette="pastel", inner="stick", data=data);
sns.catplot(x="Result_of_Treatment", y="age", kind = 'violin'
    , palette="pastel", inner="stick", data=data);
sns.catplot(x="Type", y="Time", hue= 'Result_of_Treatment',
    kind = 'violin', palette="pastel", inner="stick", split=
    True, data=data);

corr = data.corr()
corr.T

mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask)] = True
with sns.axes_style("white"):
    f, ax = plt.subplots(figsize=(7, 5))
    ax = sns.heatmap(corr, mask=mask, annot = True,
                     vmin =-1, vmax =1, center = 0,
                     xticklabels=corr.columns, cmap="RdBu_r")

sns.scatterplot(x=data['Time'], y=data['age'], hue=data['
    Result_of_Treatment'])
```

```

# 2D KDE plot
sns.jointplot(x=data['Time'], y=data['age'], kind="kde")

fig = plt.figure()
fig.subplots_adjust(hspace=1, wspace=.4)
ax = fig.add_subplot(1, 2, 1)
ax = sns.scatterplot(x=data['Time'], y=data['age'], hue=data[
    'Result_of_Treatment'], ax = ax)
ax = fig.add_subplot(1, 2, 2)
successed = data.loc[data.Result_of_Treatment == 1]
failed = data.loc[data.Result_of_Treatment == 0]
ax = sns.kdeplot(failed.Time, failed.age,label ="0",cmap="
    Blues", ax=ax)
ax = sns.kdeplot(successed.Time, successed.age,label ="1",
    cmap="Reds", ax = ax)
ax.legend()
plt.show()

successed = data.loc[data.Result_of_Treatment == 1]
failed = data.loc[data.Result_of_Treatment == 0]
ax = sns.kdeplot(failed.Time, failed.age,label ="0",
                 cmap="Blues")
ax = sns.kdeplot(successed.Time, successed.age,label ="1",
                 cmap="Reds")
ax.legend()

c = sns.FacetGrid(data, col="Type", hue="Result_of_Treatment"
    )
c.map(plt.scatter, "Time", "age", alpha=.7)
c.add_legend();
```

Listing 1.4. Data exploration and visualization code

```
from sklearn import preprocessing

std_scale = preprocessing.StandardScaler().fit(train_X)
x_train_norm = std_scale.transform(train_X)
training_norm_X = pd.DataFrame(x_train_norm, index=train_X.
    index, columns=train_X.columns)
print(training_norm_X.head())

x_test_norm = std_scale.transform(test_X)
testing_norm_X = pd.DataFrame(x_test_norm, index=test_X.index
    , columns=test_X.columns)
print(testing_norm_X.head())
```

Listing 1.5. Normalization

```python
def sigmoid(z):
  return 1.0 / (1 + np.exp(-z))

def predict(features, weights):
  '''
  Returns 1D array of probabilities
  that the class label == 1
  '''
  z = np.dot(features, weights)
  return sigmoid(z)

def cost_function(features, labels, weights):
    '''
    Using Mean Absolute Error
    Features:(100,3)
    Labels: (100,1)
    Weights:(3,1)
    Returns 1D matrix of predictions
    Cost = (labels*log(predictions) + (1-labels)*log(1-
   predictions) ) / len(labels)
    '''
    observations = len(labels)
    predictions = predict(features, weights)
    #Take the error when label=1
    class1_cost = -labels*np.log(predictions)
    #Take the error when label=0
    class2_cost = (1-labels)*np.log(1-predictions)
    #Take the sum of both costs
    cost = class1_cost - class2_cost
    #Take the average cost
    cost = cost.sum() / observations

    return cost

def update_weights(features, labels, weights, lr):
    '''
    Vectorized Gradient Descent

    Features:(200, 3)
    Labels: (200, 1)
    Weights:(3, 1)
    '''
    N = len(features)
    #1 - Get Predictions
    predictions = predict(features, weights)
    #2 Transpose features from (200, 3) to (3, 200)
    gradient = np.dot(features.T,  predictions - labels)
    #3 Take the average cost derivative for each feature
    gradient /= N
    #4 - Multiply the gradient by our learning rate
    gradient *= lr
    #5 - Subtract from our weights to minimize cost
    weights -= gradient

    return weights
def decision_boundary(prob):
  return 1 if prob >= .5 else 0

def classify(predictions):
  '''
  input  - N element array of predictions between 0 and 1
  output - N element array of 0s (False) and 1s (True)
  '''
  _decision_boundary = np.vectorize(decision_boundary)
  return _decision_boundary(predictions).flatten()

def train(features, labels, weights, lr, iters):
    cost_history = []

    for i in range(iters):
        #print (features)
        #print (weights)

        weights = update_weights(features, labels, weights,
   lr)

        #Calculate error for auditing purposes
        cost = cost_function(features, labels, weights)
        cost_history.append(cost)

        # Log Progress
        if i % 100 == 0:
            print ("iter: {}  cost: {}".format(i, cost))

    return weights, cost_history

#CALCULATION
np.random.seed(0)
weights, cost_history = train(np.hstack((np.ones((train_y.
    count(),1)), train_X[['Time','age']].to_numpy())),
      train_y.to_numpy(),
      np.random.uniform(low=-1.0, high=1.0, size=3),
      0.001, 1000)
possiblities = predict(np.hstack((np.ones((test_y.count(),1))
    ,test_X[['Time','age']].to_numpy())), weights)
predictions = classify(possiblities)
possiblities
```

Listing 1.6. Logistic regression from scratch