Using Logistic Regression and Decision Trees for water quality prediction

Introduction:

Access to potable water is very important for health and a basic human right. Potable water means that it is safe for human to consume. It is very important health and development issue that everyone has access to drinkable water everywhere in the world.

In this report I am using and comparing Logistic Regression and Decision Tree models to predict water quality from several features. Problem formulation section I discuss about my problem as a machine learning problem and my data set. After that in Method section I am discussing about methods I was using. Then in Results and Conclusion sections I am discussing about results obtained from my models and summary of the problem. Lastly in the end of the report you can find references and Python code I used to obtain my results.

Problem formulation:

For my machine learning project, I am trying to predict if the water is safe for human consumption. I am using a dataset from Kaggle(link), which has 10 columns (pH, hardness, solids (total dissolved solids), chloramines, sulfate, conductivity, organic carbon, trihalomethanes, turbidity, potability).

There are 3276 data points in the data set. Data points are different water bodies each having own pH, hardness, etc. I am going to use potability as a label which would have either value 0 (not safe to drink) or value 1 (safe to drink). Features are all the other columns of the data set. Different features of the data set are:

- pH
- hardness
- solids
- chloramines
- sulfate
- conductivity
- organic carbon
- trihalomethanes

- turbidity

All features have numerical values.

Methods:

As I mentioned earlier, there are 3276 data points in my data set, however some rows contained null values in some of the features, so I removed those rows and 2011 data points remained. I checked the amount of 0 and 1 values of the label 'Potability' to see if the data is balanced. Figure shows that there is about 1200 datapoints with value of the label being 0 and 811 data points with label value 1. This shows that data is mildly imbalanced which we should keep in mind.
I also checked correlation between the features using heatmap. It shows that there is not really a correlation between any features and thus I can't remove any columns and will use all the features in the dataset.

I treat this problem as a classification problem because data set is multivariate with 0/1 binary label. First classification model that I'm going to try is Logistic Regression which uses Logistic loss as a loss function to asses the quality of the hypothesis. Hypothesis space in Logistic Regression is defined as:

$H^{(n)} := \{h^{(w)} : R^n \rightarrow R : h^{(w)}(x)=w^T x$ with some parameter vector $w \in R^n\}$

Logistic regression tries to minimize average logistic loss. I used log_loss function in scikit-learn:

$$L_{log}(y,p) = -(ylog(p) + (1-y)\log(1-p))$$

I used log_loss function for both training and validation error. I also used accuracy_score function

$$accuracy(y, \hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} 1(\hat{y}_i = y_i)$$

and F1-score to evaluate my models performance. I used accuracy_score because it's a good way to measure how many of the predictions the model got right. F1-score combines recall and precision into a single quantity. Goal would be to find hypothesis with big recall and big precision. For my problem precision is probably more important than recall because it would react to false positives while recall reacts to false negatives. I'm predicting water quality so it is really important that my method would minimize false positives because drinking not potable water could be dangerous. However, I also want to make my model to be good at minimizing false positives too, so I decided to use F1-score.

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

I split the data into training, validation and test sets using train_test_split function. Sizes of the sets are 60%, 20% and 20% respectively. I used these set sizes because this ratio is widely used in machine learning problems, and it was suitable ratio for the size of my data set (2011 data points). I used this split for both models.

Another classification model I am using is Decision Tree. It's a tree consisting of true/false questions. Nodes are connected with edges and model chooses the edge where it proceeds depending on if value of the node is true or false. Every leaf node gives a value to the label.

The loss function used in Decision Tree is Gini impurity. Gini impurity of a data set is the likelihood of a new data to be misclassified.

$$Gini = 1 - \sum_j p_j^2 \ , where\ p_j\ is\ the\ probability\ of\ class\ j$$

I also used accuracy_score function and F1-score to evaluate models performance. I again decided to use accuracy_score and F1-score for the same reasons as in logistic regression.

Results:

Logistic regression:

Training error, validation error, test error and F1-score are seen in the table below.

Training	Validation	Test	F1-score
13.51766	13.14539	15.25535	0.01111

Training, validation and test errors calculated with log_loss function are very high and F1-Score is low. This means that our model is not very precise. From the confusion matrix we can see that the number of false negatives is really high. This means that this model predicts that the label is 0 even though its real value is 1.

Decision tree:

I tried my decision tree model with different depths. Best maximum depth seems to be depth 4 because test accuracy for that maximum depth was the best and also training accuracy and validation accuracy were close to test accuracy. Maximum depths bigger than 4, test and validation accuracies didn’t really increase but training accuracy started to converge to 1. This tells us that max depths bigger than 4 would have caused overfitting.
Table for training accuracy, validation accuracy, test accuracy and F1-score for max depth 4:

Training	Validation	Test	F1-score
0.66998	0.65920	0.61290	0.30973

Accuracy for training, validation and test sets are pretty good and F1-score is much higher than in Logistic Regression. F1-score should still be higher and problem for this model too is that it predicts a lot of false negatives. This can be seen from confusion matrix.

Final chosen method:
I chose Decision Tree as my final method. For Logistic Regression the errors were really high and it basically predicted almost every label to value 0. Accuracy for the test set was pretty similar for both models but it’s not good way to compare these two methods because Logistic Regression model got around 0.6 accuracy just for predicting almost every label to 0.
F1-score was much better in Decision Tree model than in Logistic Regression model. In the Decision Tree model the problem was again that there were a lot of false negatives but the amount of true positives was significantly better for Decision Tree model (36) than for Logistic Regression model (1).

Conclusion:

Low F1-score tells us that there is a lot of room for improvement. Model had low precision which means that it predicted a lot of false negatives. I was predicting if a water is potable or not so predicting false negatives is better than predicting false positives but the results should still be a lot better.
To better my models performance I would consider balancing the data so there would be equal amount of 0 and 1 values for the labels. This would prevent the model being biased towards one class. Another thing I could try is to gather more data.

References:

https://www.kaggle.com/adityakadiwal/water-potability

http://mlbook.cs.aalto.fi
https://en.wikipedia.org/wiki/Drinking_water

Appendices:

Untitled

March 28, 2022

[1]:
```
import numpy as np                    # import numpy package under shorthand "np"
import pandas as pd                   # import pandas package under shorthand␣
 ↪"pd"
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix  # evaluation␣
 ↪metrics
from sklearn.model_selection import train_test_split, KFold
from sklearn.preprocessing import PolynomialFeatures    # function to generate␣
 ↪polynomial and interaction features
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.tree import DecisionTreeClassifier
```

[2]:
```
import os
```

[3]:
```
os.chdir('/notebooks/ml_project/')
```

[27]:
```
df = pd.read_csv('water_potability.csv')
df.head(5)
```

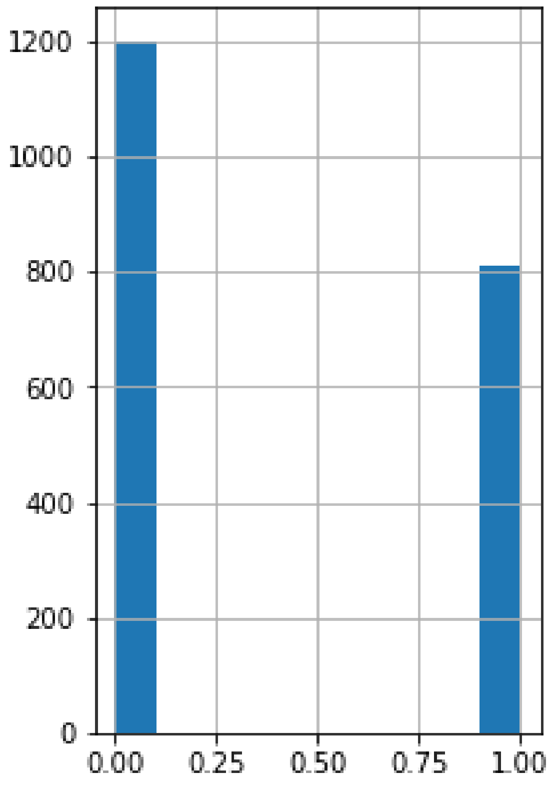
[27]:
```
         ph    Hardness        Solids  Chloramines     Sulfate  Conductivity  \
0       NaN  204.890455  20791.318981     7.300212  368.516441    564.308654
1  3.716080  129.422921  18630.057858     6.635246         NaN    592.885359
2  8.099124  224.236259  19909.541732     9.275884         NaN    418.606213
3  8.316766  214.373394  22018.417441     8.059332  356.886136    363.266516
4  9.092223  181.101509  17978.986339     6.546600  310.135738    398.410813

   Organic_carbon  Trihalomethanes  Turbidity  Potability
0       10.379783        86.990970   2.963135           0
1       15.180013        56.329076   4.500656           0
2       16.868637        66.420093   3.055934           0
3       18.436524       100.341674   4.628771           0
4       11.558279        31.997993   4.075075           0
```

[5]:
```
df = df.dropna(axis=0)
```

```
[6]: df.info()
```

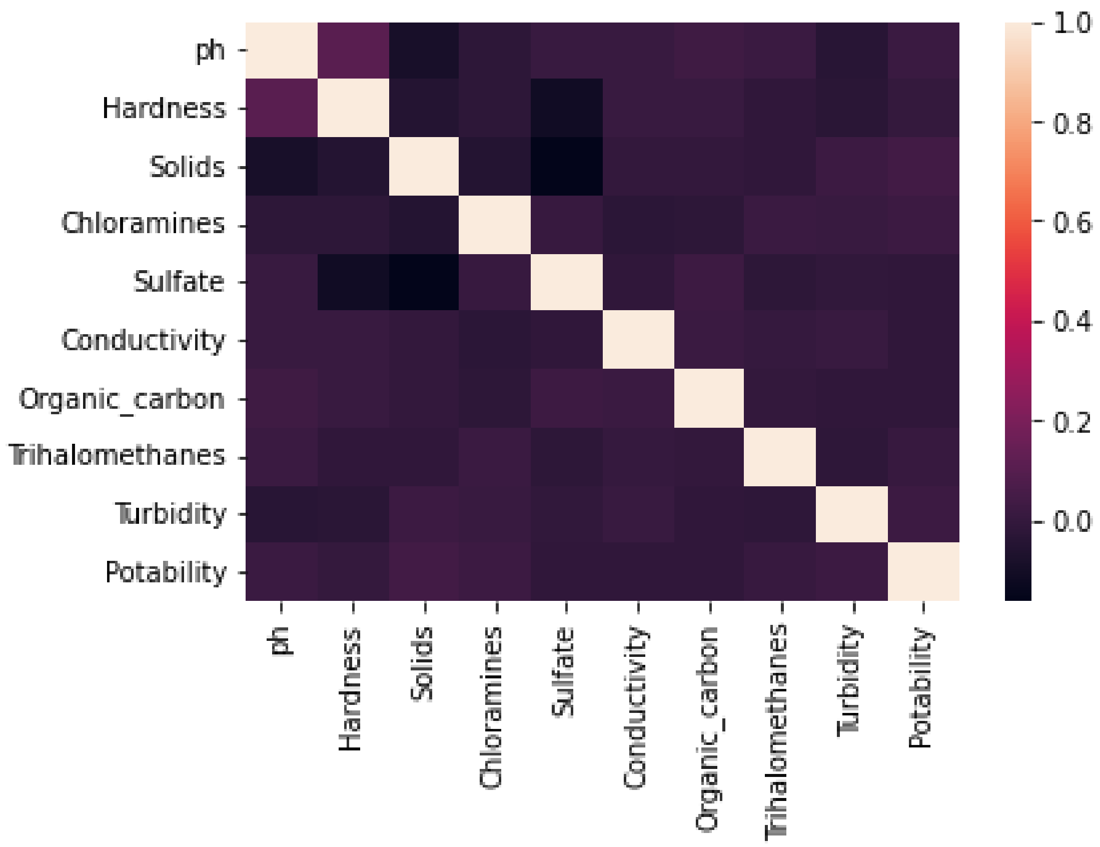
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2011 entries, 3 to 3271
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ph               2011 non-null   float64
 1   Hardness         2011 non-null   float64
 2   Solids           2011 non-null   float64
 3   Chloramines      2011 non-null   float64
 4   Sulfate          2011 non-null   float64
 5   Conductivity     2011 non-null   float64
 6   Organic_carbon   2011 non-null   float64
 7   Trihalomethanes  2011 non-null   float64
 8   Turbidity        2011 non-null   float64
 9   Potability       2011 non-null   int64  
dtypes: float64(9), int64(1)
memory usage: 172.8 KB
```

```
[7]: df['Potability'].hist(figsize=(3,5))
     plt.show()
```



```
[8]: sns.heatmap(df.corr())
     plt.show
```

```
[8]: <function matplotlib.pyplot.show(close=None, block=None)>
```



```
[9]: X = df.drop('Potability', axis=1)
     y = df['Potability']

     ## Split the dataset into a training set and a remaining set with␣
      ↪train_test_split.

     X_train, X_rem, y_train, y_rem = train_test_split(X, y, test_size = 0.4,␣
      ↪random_state=42)
```

```
[10]: ## Split the remaining dataset into a validation set and test set with␣
       ↪train_test_split.

      X_val, X_test, y_val, y_test = train_test_split(X_rem, y_rem, test_size=0.5,␣
       ↪random_state=42)
```

[11]:
```python
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
y_pred_train = logreg.predict(X_train)
acc_train = accuracy_score(y_train, y_pred_train)

y_pred_val = logreg.predict(X_val)
acc_val = accuracy_score(y_val, y_pred_val)

y_pred_test = logreg.predict(X_test)
acc_test = accuracy_score(y_test, y_pred_test)

print("accuracy of train : ", acc_train)
print("accuracy of val : ", acc_val)
print("accuracy of test : ", acc_test)
```

```
accuracy of train :  0.6086235489220564
accuracy of val :  0.6194029850746269
accuracy of test :  0.5583126550868487
```

[12]:
```python
from sklearn.metrics import log_loss
logloss_train = log_loss(y_train, y_pred_train)
logloss_val = log_loss(y_val, y_pred_val)
logloss_test = log_loss(y_test, y_pred_test)

print("Logistic loss train : ", logloss_train)
print("Logistic loss val : ", logloss_val)
print("Logistic loss test : ", logloss_test)
```

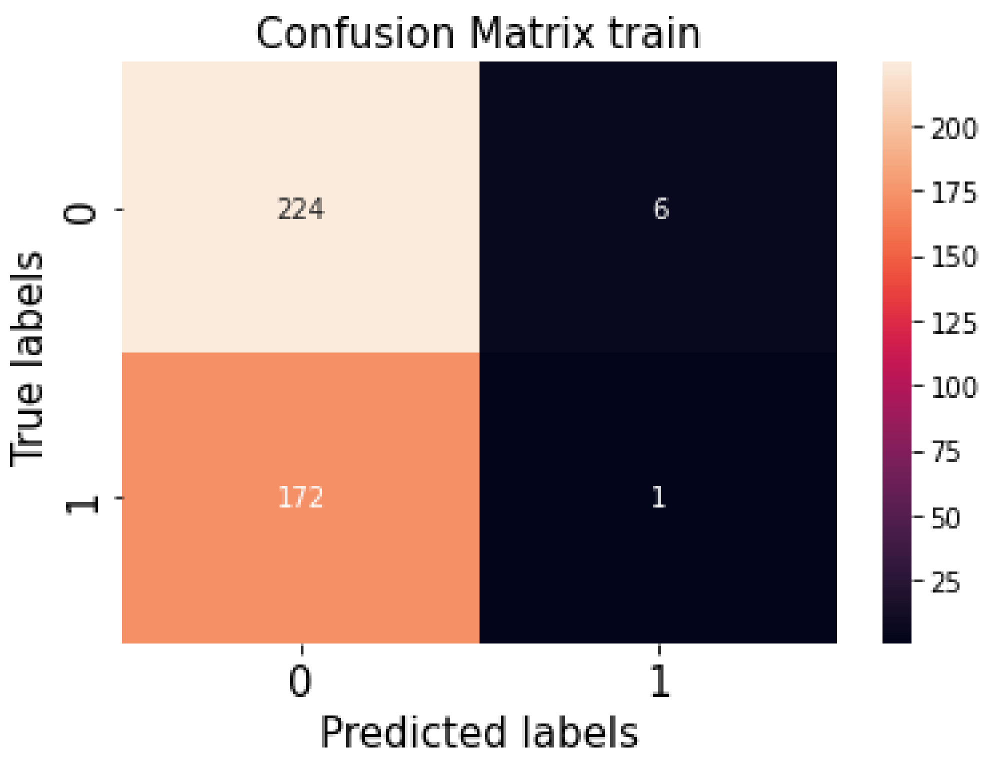
```
Logistic loss train :  13.517667045095353
Logistic loss val :  13.145359173174613
Logistic loss test :  15.255352347093506
```

[13]:
```python
conf_mat_test = confusion_matrix(y_test, y_pred_test)


print(conf_mat_test)
```

```
[[224   6]
 [172   1]]
```

[14]:
```python
ax= plt.subplot()

sns.heatmap(conf_mat_test, annot=True, fmt='g', ax=ax)

ax.set_xlabel('Predicted labels',fontsize=15)
ax.set_ylabel('True labels',fontsize=15)
ax.set_title('Confusion Matrix train',fontsize=15)
ax.xaxis.set_ticklabels(['0', '1'],fontsize=15)
ax.yaxis.set_ticklabels(['0', '1'],fontsize=15)
```

```
[14]: [Text(0, 0.5, '0'), Text(0, 1.5, '1')]
```



```
[15]: precision = conf_mat_test[1][1]/(conf_mat_test[1][1] + conf_mat_test[0][1])
recall = conf_mat_test[1][1]/(conf_mat_test[1][1] + conf_mat_test[1][0])
f1 = 2 * ((precision*recall)/(precision+recall))
print("Precision : ", precision)
print("Recall : ", recall)
print("F1-score : ", f1)
```

```
Precision :  0.14285714285714285
Recall :  0.005780346820809248
F1-score :  0.011111111111111112
```

```
[16]: ##Decision tree

tree = DecisionTreeClassifier()
tree.fit(X_train, y_train)
```

```
[16]: DecisionTreeClassifier()
```

[17]:
```python
accuracies_train = []
accuracies_val = []
accuracies_test = []
for i in range(1, 8):
    tree = DecisionTreeClassifier(criterion="gini", max_depth=i)
    tree.fit(X_train, y_train)
    y_pred_train = tree.predict(X_train)
    acc_train = accuracy_score(y_train, y_pred_train)
    accuracies_train.append(acc_train)

    y_pred_val = tree.predict(X_val)
    acc_val = accuracy_score(y_val, y_pred_val)
    accuracies_val.append(acc_val)

    y_pred_test = tree.predict(X_test)
    acc_test = accuracy_score(y_test, y_pred_test)
    accuracies_test.append(acc_test)


print(accuracies_train, "\n")
print(accuracies_val, "\n")
print(accuracies_test, "\n")
```

```
[0.6194029850746269, 0.6227197346600332, 0.6533996683250415, 0.6699834162520729,
0.6998341625207297, 0.724709784411277, 0.7545605306799337]

[0.6144278606965174, 0.6194029850746269, 0.654228855721393, 0.6567164179104478,
0.6293532338308457, 0.6442786069651741, 0.6393034825870647]

[0.5905707196029777, 0.5880893300248139, 0.6054590570719603, 0.6153846153846154,
0.5955334987593052, 0.6178660049627791, 0.6228287841191067]
```

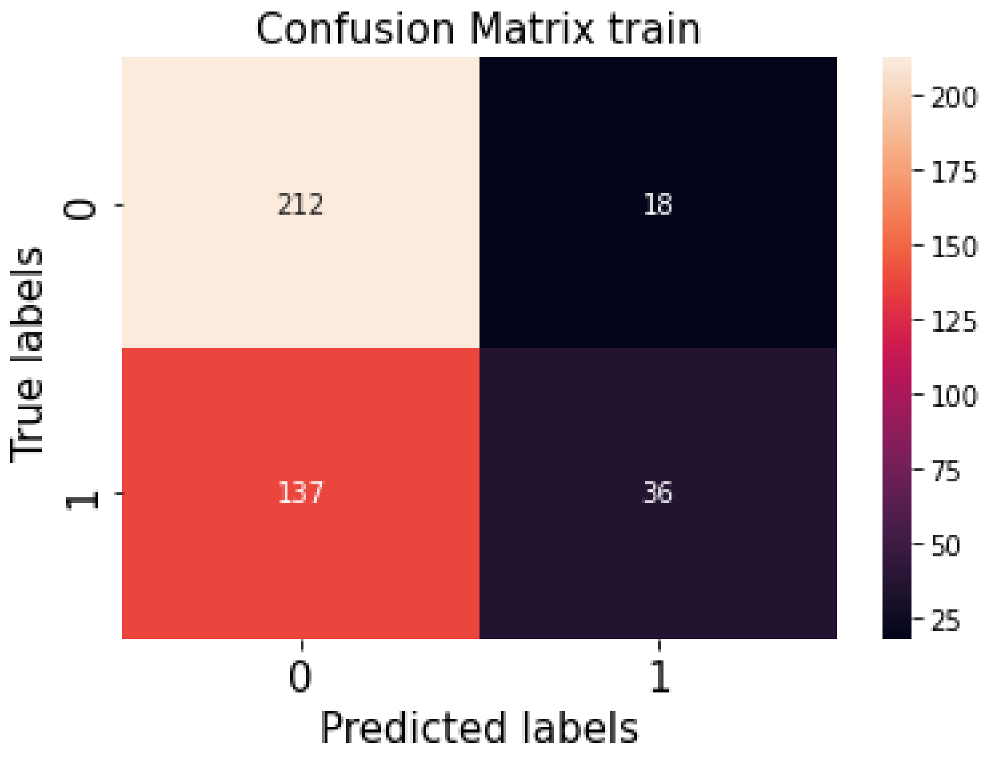
[24]:
```python
tree = DecisionTreeClassifier(criterion="gini", max_depth=4)
tree.fit(X_train, y_train)
y_pred_train = tree.predict(X_train)
acc_train = accuracy_score(y_train, y_pred_train)


y_pred_val = tree.predict(X_val)
acc_val = accuracy_score(y_val, y_pred_val)


y_pred_test = tree.predict(X_test)
acc_test = accuracy_score(y_test, y_pred_test)

conf_mat_test2 = confusion_matrix(y_test, y_pred_test)
```

```
print("Training accuracy : ", acc_train)
print("Validation accuracy : ", acc_val)
print("Test accuracy : ", acc_test, "\n")
print(conf_mat_test2)
```

```
Training accuracy :  0.6699834162520729
Validation accuracy :  0.6592039800995025
Test accuracy :  0.6129032258064516

[[212  18]
 [138  35]]
```

[21]:
```
ax= plt.subplot()

sns.heatmap(conf_mat_test2, annot=True, fmt='g', ax=ax)

ax.set_xlabel('Predicted labels',fontsize=15)
ax.set_ylabel('True labels',fontsize=15)
ax.set_title('Confusion Matrix train',fontsize=15)
ax.xaxis.set_ticklabels(['0', '1'],fontsize=15)
ax.yaxis.set_ticklabels(['0', '1'],fontsize=15)
```

[21]:
```
[Text(0, 0.5, '0'), Text(0, 1.5, '1')]
```



[20]:
```
precision = conf_mat_test2[1][1]/(conf_mat_test2[1][1] + conf_mat_test2[0][1])
recall = conf_mat_test2[1][1]/(conf_mat_test2[1][1] + conf_mat_test2[1][0])
f1 = 2 * ((precision*recall)/(precision+recall))
print("Precision : ", precision)
print("Recall : ", recall)
print("F1-score : ", f1)
```

```
Precision :  0.6666666666666666
Recall :  0.20809248554913296
F1-score :  0.3171806167400881
```

[ ]:

[ ]: