Detecting regular drinkers by body signals

Aalto University

CS-C3240 Machine Learning D
Project stage 1

Content:

1. Introduction

In the medical field, sometimes, it is important to understand a patient's drinking history, in order to better prescribe them with the most suitable treatments. However, this can be a sensitive topic to some people, especially if they have struggled with alcohol addiction. Therefore, some patients may not disclose the truth to healthcare professionals regarding their drinking habits, which can impede treatment and endanger the patients themselves. This paper helps prevent that situation by studying how to predict if a person is a regular drinker through numerous body signals.

The paper is structured as follow: section 2 discusses problem formulation and features of the dataset; section 3 discusses data preparation, model selection and model validation; section 4 discusses the results of the study in this report; and lastly, section 5 concludes the report and reflect on the limitations of the methods with possible ways to improve them.

2. Problem Formulation

In this paper, we are trying to predict whether or not a person is a habitual drinker via numerous different body signals, such as waistline, sight, hearing, amount of HDL cholesterol in their blood, etc - as presented in more details in table number 1 under "Appendix". This is a large dataset with close to a million entries collected by the Korean National Health Insurance Service, and made available on Kaggle [1]. Each of the data entries give insight into a person's body signals and whether or not they drink regularly.

Briefly, regarding the features in the data set (with more details given under the Appendix):

- 3 of the features, being gender, hearing ability on left and right ear respectively, are binary;
- The feature on the amount of protein in the subject's urine is categorical;
- The rest are continuous data.

This project utilises supervised learning, with the label concerned being the binary outcome of whether or not a person drinks regularly.

3. Methods

3.1. Preparing data

The number of data points is the number of entries in the dataset, which is 991346. In other words, there are 991346 subjects with their respective body signals being examined. In the dataset, some data points are missing. Out of the 21 values as shown in Table 1 under the Appendix, the "Drinker or Not" column is transformed into the label, while the rest of the columns will be the features.

To make data processing easier, the binary data of the “sex” feature is changed from “male” and “female” to “0” and “1” respectively, as strings cannot be interpreted like numbers. Similarly, the binary data of the label (“Drinker or Not”) is changed from “yes” and “no” to “1” and “0” respectively. Due to the fact that there are missing data points, data rows with blank entries are excluded.

Our data set has a total of 20 features (Table 1, Appendix), hence there is a need to perform data reduction to eliminate the noise and select the few features that really matter. In other words, we have to look at each feature’s correlation to to label. To do this, the correlation matrix for each column is calculated, with both the label column and the features columns included. The columns that correlate with the label above a certain threshold are then selected for training. For the purpose of experimentation, we have selected 5 thresholds, being: 0.05, 0.10, 0.15, 0.20, and 0.25. This means that we train our data in thresholds to judge how different features contribute to the result. The visualisation of the correlation matrix is included in the “Appendix” as “Figure 1”.

3. 2. Model selection

3.2.1. Logistic Regression and Logistic Loss

Our first choice of machine learning model is logistic regression. This model uses a linear hypothesis space and works by measuring the correlation between the label and the features [2]. Logistic regression is suited for problems with 2 response classes, such as in our case of whether or not the subject is a habitual drinker. This model is highly comprehensible with easy implementation and good training efficiency. This model uses a logistic loss function, which is a continuous function.

Because the dataset has a large number of features with no clear and simple relation between features and labels, we decided to use logistic loss to predict the labels of data points based on the probability. The logistic loss function assigns to each point a probability so that the data points can be classified to one of the binary outcomes of the label (0 or 1 in our case). Another reason why this function is chosen is that it comes with a well-fitted library for logistic regression.

3.2.2. K-Nearest Neighbours (KNN) and 0/1 Loss Function

Another machine learning model that we are taking into consideration is the KNN model. In our project, this model estimates the likelihood that the subject belongs to one group (drinker/ non drinker) based on the group that the subject’s nearest data point belongs to. This means that by looking at the data points around a certain data point, this model classifies that data point [2]. As we have a large amount of data points available, this method would produce good accuracy even with the larger number of dimensions. For the KNN model, we choose 0/1 loss as we want to assign the points in a binary way, based on the neighbours of the datapoint. This way, the 0/1 loss function shows clearly how well our model works. When assessing the accuracy of the model, this loss function directly measures the number of instances that are classified incorrectly, hence, providing a clear and easily-interpreted measure of classification error.

3.3. Model validation

We split our data into the training, validation and test sets by first shuffling them using the pandas.DataFrame.sample function [3], which will give results in a random sample to ensure that the order of the data would not influence our model. After that, we split the data into sets with numpy.split [4], with 60% of the data being used for training, 20% being used for testing, and the last 20% being used for validating. Since our data set is very large with 991346 data points, it is not necessary to utilise methods such as k-fold cross-validation to avoid over or under-fitting. Therefore, using a single split proves sufficient in terms of model accuracy, while avoiding the lengthy time it would require if methods like k-fold cross-validation is used instead.

4. Result

After testing with various feature matrices and hyperparameters, we decided that for logistic regression, we will use the features whose correlation with the label is bigger than 0.05, with the hyperparameter C = 12, which is the inverse of regularisation strength. Amongst the values for hyperparameter tested, C = 12 yields the best results. Similarly, after various testings, for KNN, we have decided to use 50 closest neighbours, as well as features whose correlation with the label is bigger than 0.15.

The training error of logistic regression is equal to 10.195, while the training error for KNN is 0.276. The validation error of logistic regression is equal to 10.226 while the validation error for KNN is 0.288. This means that both the training error and the validation error of KNN is much smaller than that of logistic regression.

As we use different loss functions in our machine learning methods, direct comparison might be difficult to make. This is due to the fact that the logistic loss is not the best one for this model as misclassifications can result in very high losses. When using logistic loss, our test error for logistic regression is 10.194. When using 0/1 loss, the test error for the KNN method is 0.287. Therefore, direct comparison would suggest that KNN is a better method for this problem. However, when we use 0/1 loss on logistic regression, the test error is then 0.283 - better than that of KNN. Meaning that logistic regression would be slightly better if we considered only this metric.

At the end, we select logistic regression as our final method. It has slightly better accuracy than KNN. The high logistic loss is due to possible outliers: some people who are not drinkers can have very similar bodily signals to drinkers, and vice versa. These outliers in body signals can be caused by underlying medical issues. If we could deal with outliers better, this method could improve the accuracy of the model. In general, logistic regression seems to have much room for improvement.

As mentioned above, the test set is constructed by first shuffling the data using the pandas.DataFrame.sample function [3], then split it using numpy.split [4], with the test set being 20% of the dataset.

Hence, our final chosen method is logistic regression with the test error of 10.194.

5. Conclusion

From this report, as we achieved an accuracy of 71.72% on the test set, hence, detecting regular drinkers through body signals is a problem that can be solved with machine learning. Judging by various metrics above, logistic regression proved to be a better method to be utilised in this machine learning problem. There is room for improvement of this result, taking into account that there are drastic outliers in the dataset, as well as certain features that may not be causal to drinking habits, and thus, may not contribute to the prediction. From our working, the features that correlate the most with the label are sex, age, height, and weight. Amongst them, height and weight strongly correlates with sex, suggesting that these features can be more demographic, and may not be conclusive for habitual intoxication prediction. Therefore, a way to improve our studies is to know which body signals are affected by drinking, and use the values for those body signals instead for our studies. This can be achieved by looking at other research on drinking and bodily signals for reference.

References

[1] Soo.Y (n.d.). *Smoking and Drinking Dataset with body signal*. [online] Available at: https://www.kaggle.com/datasets/sooyoungher/smoking-drinking-dataset
[2] A. Jung, “Machine Learning: The Basics”, Springer, Singapore, 2022
[3] pandas.pydata.org. (n.d.). *pandas.DataFrame.sample — pandas 2.1.1 documentation*. [online] Available at: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.sample.html
[4] numpy.org. (n.d.). *numpy.split — NumPy v1.26 Manual*. [online] Available at: https://numpy.org/doc/stable/reference/generated/numpy.split.html

Appendix

No.	Column	Description	Data type
1	sex	[male/ female]	Binary
2	age	round up to 5 years [year]	Continuous
3	height	round up to 5 cm [kg]	Continuous
4	weight	[kg]	Continuous
5	sight_left	eyesight (left eye)	Continuous
6	sight_right	eyesight (right eye)	Continuous
7	hear_left	hearing left, 1 (normal), 2 (abnormal)	Binary
8	hear_right	hearing right, 1 (normal), 2 (abnormal)	Binary
9	SBP	Systolic blood pressure [mmHg]	Continuous
10	SDP	Diastolic blood pressure [mmHg]	Continuous
11	tot_chole	total cholesterol [mg/dL]	Continuous
12	HDL_chole	HDL cholesterol [mg/dL]	Continuous
13	LDL_chole	LDL cholesterol [mg/dL]	Continuous
14	triglyceride	triglyceride [mg/dL]	Continuous
15	hemoglobin	hemoglobin [g/dL]	Continuous
16	urine_protein	protein in urine, 1(-), 2(+/-), 3(+1), 4(+2), 5(+3), 6(+4)	Categorical
17	serum_creatinine	serum(blood) creatinine [mg/dL]	Continuous
18	SGOT_AST	SGOT(Glutamate-oxaloacetate transaminase) AST(Aspartate transaminase) [IU/L]	Continuous
19	SGOT_ALT	ALT(Alanine transaminase) [IU/L]	Continuous
20	gamma_GTP	y-glutamyl transpeptidase [IU/L]	Continuous
21	Drinker or Not	Drinker or Not [Yes/No]	Binary **(Label)**

Table 1: Details on the dataset's columns (20 features and 1 label)

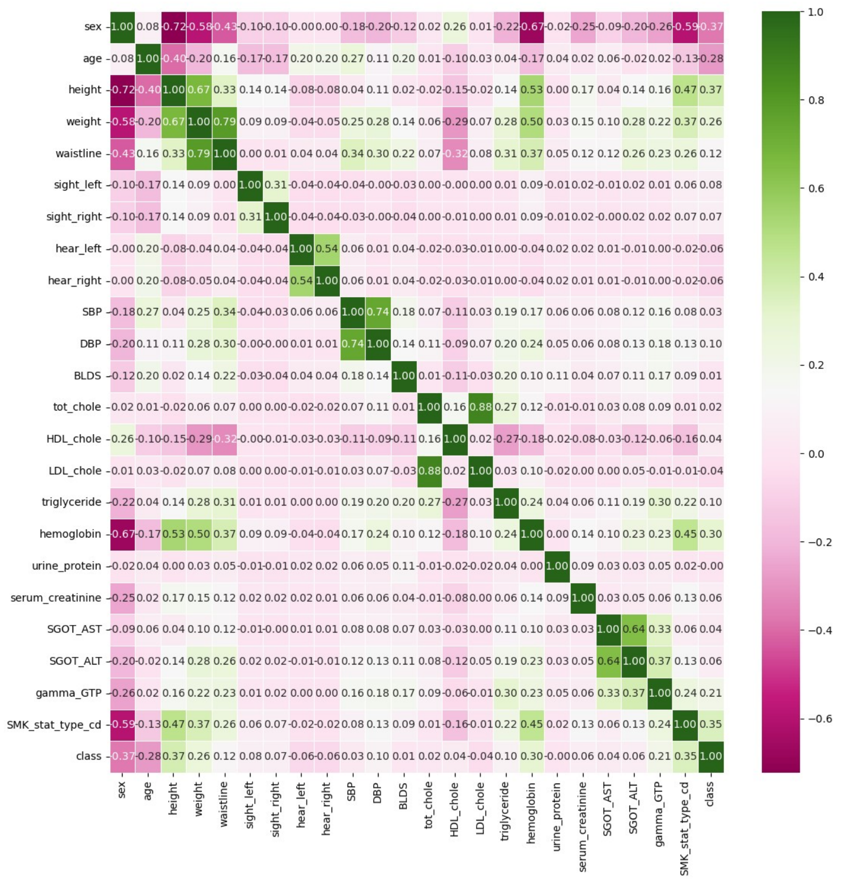

Figure 1: Correlation matrix

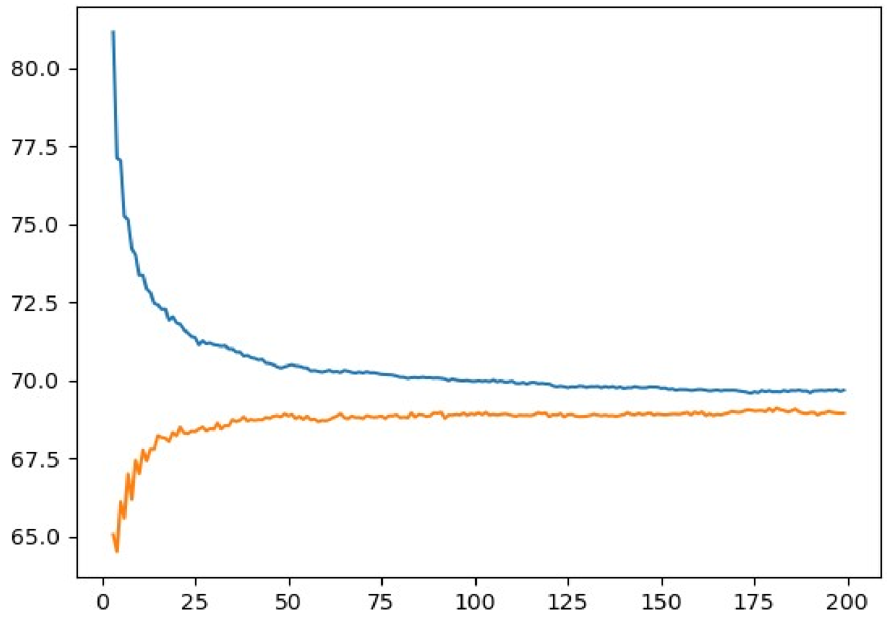

Figure 2: Accuracy for the KNN method varies with the value of k
blue = training accuracy, orange = validation accuracy
(horizontal axis = value of k, vertical axis = accuracy)

From the next page onwards, there will be codes of what we have done so far in our project

code

October 8, 2023

```
[ ]: import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
     from sklearn.linear_model import LogisticRegression
     from sklearn.metrics import accuracy_score, confusion_matrix,␣
      ↪classification_report, log_loss, zero_one_loss
     from sklearn.model_selection import cross_val_score, KFold
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.model_selection import cross_validate


     df = pd.read_csv('smoking_driking_dataset.csv', sep = ';')
     print(df.shape[0])
     df.head(5)
```

```
[ ]: df = df[df['waistline'] < 200]
     print(df.shape[0])
     df['DRK_YN'] = df['DRK_YN'].replace({'Y': 1, 'N': 0})
     df['sex'] = df['sex'].replace({'Female': 1, 'Male': 0})
     for i in df.columns[5:-1]:
         mu = df[i].mean()
         sd = df[i].std()
         df[i] = df[i].apply(lambda x: (x - mu)/sd)
     df.head(5)
     cols = df.columns
```

```
[ ]: for label in cols[:-1]:
       plt.hist(df[df["DRK_YN"]==1][label], color='blue', label='Drinks', alpha=0.7,␣
      ↪density=True)
       plt.hist(df[df["DRK_YN"]==0][label], color='red', label='not', alpha=0.7,␣
      ↪density=True)
       plt.title(label)
       plt.ylabel("Probability")
       plt.xlabel(label)
       plt.legend()
       plt.show()
```

```
[ ]: df = df.rename(columns= {'DRK_YN' : 'class'})
     cor = df.corr()
     fig, ax = plt.subplots(figsize=(13,13))
     sns.heatmap(cor, linewidth=0.5, annot = True, cmap = 'PiYG', fmt = '.2f', ax =␣
      ↪ax)
     plt.show()
```

```
[ ]: feature_matrix = []
     for i in [0.05, 0.1, 0.15, 0.2, 0.25]:
         pom = []
         for j in df.columns[:-1]:
             if cor['class'][j] > i or cor['class'][j] < -i:
                 pom.append(j)
         feature_matrix.append(pom)
     for i in feature_matrix:
         print(i)
```

```
[ ]: def generate_confusion_matrix(y_true, y_pred):
         ax = plt.subplot()
         c_mat = confusion_matrix(y_true, y_pred)
         sns.heatmap(c_mat, 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', fontsize=15)

     df_new = df.sample(n = 100000) # for checking different training methods and␣
      ↪parameters

     for j in range(10):
         print('-'*40)
         train, valid, test = np.split(df_new.sample(frac=1), [int(0.6*len(df_new)),␣
      ↪int(0.8*len(df_new))])
         for i in range(len(feature_matrix)):
             X_train = train[feature_matrix[i]]
             y_train = train['class']
             X_valid = valid[feature_matrix[i]]
             y_valid = valid['class']
             X_test = test[feature_matrix[i]]
             y_test = test['class']
             clf_1 = LogisticRegression(max_iter = 4000, solver = 'newton-cholesky',␣
      ↪class_weight = 'balanced', C = 12)
             clf_1.fit(X_train, y_train)
             y_pred = clf_1.predict(X_train)
             multi_accuracy = accuracy_score(y_train, y_pred)
             #print(classification_report(y_train, y_pred))

             #y_test_pred = clf_1.predict(X_test)
```

```
        #generate_confusion_matrix(y_test, y_test_pred)
        #multi_accuracy = accuracy_score(y_test, y_test_pred)

        print(f"Prediction accuracy on training set: {100*multi_accuracy:.2f}%")
        #print(classification_report(y_test, y_test_pred))
        loss = log_loss(y_train, y_pred)
        print(('validation loss', loss))
        y_valid_pred = clf_1.predict(X_valid)
        multi_accuracy = accuracy_score(y_valid, y_valid_pred)
        print(f"Prediction accuracy on validation set: {100*multi_accuracy:.
 ↪2f}%")
        loss = log_loss(y_valid, y_valid_pred)
        print(('validation loss', loss))
        plt.show()
```

```
[ ]: x_line = []
     y_line = []
     y_w = []
     df_new = df.sample(n = 100000)
     train, valid, test = np.split(df_new.sample(frac=1), [int(0.
      ↪6*len(df_new)),int(0.8*len(df_new))])
     for j in range(len(feature_matrix)):
         print('-'*40)
         for i in range(3, 200):
             X_train = train[feature_matrix[j]]
             y_train = train['class']
             X_valid = valid[feature_matrix[j]]
             y_valid = valid['class']
             X_test = test[feature_matrix[j]]
             y_test = test['class']
             knn_model = KNeighborsClassifier(n_neighbors=i)
             knn_model.fit(X_train, y_train)
             y_pred = knn_model.predict(X_train)
             multi_accuracy = accuracy_score(y_train, y_pred)
             print(i)
             print(f"Prediction accuracy on training set: {100*multi_accuracy:.2f}%")
             print('Training loss: %.3f' % zero_one_loss(y_train, y_pred))
             y_line.append(100*multi_accuracy)
             y_valid_pred = knn_model.predict(X_valid)
             multi_accuracy = accuracy_score(y_valid, y_valid_pred)
             print(f"Prediction accuracy on validation set: {100*multi_accuracy:.
      ↪2f}%")
             print('Validation loss: %.3f' % zero_one_loss(y_valid, y_valid_pred))
             print("")
             x_line.append(i)
             y_w.append(100*multi_accuracy)
```

```
        #knn_model_2 = KNeighborsClassifier(n_neighbors=i, weights = 'distance')
        #knn_model_2.fit(X_train, y_train)
        #y_pred_2 = knn_model_2.predict(X_test)
        #multi_accuracy_2 = accuracy_score(y_test, y_pred_2)
        #y_w.append(100*multi_accuracy_2)


    plt.plot(x_line, y_line, x_line, y_w)
    plt.savefig('KNN.png')
    plt.show()
```

```
[ ]: x_line = []
     y_line = []
     y_w = []
     df_new = df.sample(n = 100000)
     train, valid, test = np.split(df_new.sample(frac=1), [int(0.
      ↪6*len(df_new)),int(0.8*len(df_new))])
     for i in range(3, 200):
         X_train = train[feature_matrix[j]]
         y_train = train['class']
         X_valid = valid[feature_matrix[j]]
         y_valid = valid['class']
         X_test = test[feature_matrix[j]]
         y_test = test['class']
         knn_model = KNeighborsClassifier(n_neighbors=i)
         knn_model.fit(X_train, y_train)
         y_pred = knn_model.predict(X_train)
         multi_accuracy = accuracy_score(y_train, y_pred)
         print(i)
         print(f"Prediction accuracy on training set: {100*multi_accuracy:.2f}%")
         print('Training loss: %.3f' % zero_one_loss(y_train, y_pred))
         y_line.append(100*multi_accuracy)
         y_valid_pred = knn_model.predict(X_valid)
         multi_accuracy = accuracy_score(y_valid, y_valid_pred)
         print(f"Prediction accuracy on validation set: {100*multi_accuracy:.2f}%")
         print('Validation loss: %.3f' % zero_one_loss(y_valid, y_valid_pred))
         print("")
         x_line.append(i)
         y_w.append(100*multi_accuracy)

         #knn_model_2 = KNeighborsClassifier(n_neighbors=i, weights = 'distance')
         #knn_model_2.fit(X_train, y_train)
         #y_pred_2 = knn_model_2.predict(X_test)
         #multi_accuracy_2 = accuracy_score(y_test, y_pred_2)
         #y_w.append(100*multi_accuracy_2)
```

```
plt.plot(x_line, y_line, x_line, y_w)
plt.savefig('KNN.png')
plt.show()
```

[ ]:
```
train, valid, test = np.split(df.sample(frac=1), [int(0.6*len(df)), int(0.
 ↪8*len(df))])
X_train = train[feature_matrix[0]]
y_train = train['class']
X_valid = valid[feature_matrix[0]]
y_valid = valid['class']
X_test = test[feature_matrix[0]]
y_test = test['class']
clf_1 = LogisticRegression(max_iter = 4000, solver = 'newton-cholesky',␣
 ↪class_weight = 'balanced', C = 12)
clf_1.fit(X_train, y_train)


y_pred = clf_1.predict(X_train)
multi_accuracy = accuracy_score(y_train, y_pred)
print(f"Prediction accuracy on training set: {100*multi_accuracy:.2f}%")
print('Training loss: %.3f' % log_loss(y_train, y_pred))
print("")

y_valid_pred = clf_1.predict(X_valid)
multi_accuracy = accuracy_score(y_valid, y_valid_pred)
print(f"Prediction accuracy on validation set: {100*multi_accuracy:.2f}%")
print('Validation loss: %.3f' % log_loss(y_valid, y_valid_pred))
print("")

y_test_pred = clf_1.predict(X_test)
generate_confusion_matrix(y_test, y_test_pred)
multi_accuracy = accuracy_score(y_test, y_test_pred)
print(f"Prediction accuracy on test set: {100*multi_accuracy:.2f}%")
print('Test loss: %.3f' % log_loss(y_test, y_test_pred))
```

[ ]:
```
train, valid, test = np.split(df.sample(frac=1), [int(0.6*len(df)), int(0.
 ↪8*len(df))])
X_train = train[feature_matrix[2]]
y_train = train['class']
X_valid = valid[feature_matrix[2]]
y_valid = valid['class']
X_test = test[feature_matrix[2]]
y_test = test['class']
knn_model = KNeighborsClassifier(n_neighbors=50)
knn_model.fit(X_train, y_train)

y_pred = knn_model.predict(X_train)
```

```python
multi_accuracy = accuracy_score(y_train, y_pred)
print(f"Prediction accuracy on training set: {100*multi_accuracy:.2f}%")
print('Training loss: %.3f' % zero_one_loss(y_train, y_pred))

y_valid_pred = knn_model.predict(X_valid)
multi_accuracy = accuracy_score(y_valid, y_valid_pred)
print(f"Prediction accuracy on training set: {100*multi_accuracy:.2f}%")
print('Validation loss: %.3f' % zero_one_loss(y_valid, y_valid_pred))

y_test_pred = knn_model.predict(X_test)
generate_confusion_matrix(y_test, y_test_pred)
multi_accuracy = accuracy_score(y_test, y_test_pred)
print(f"Prediction accuracy on test set: {100*multi_accuracy:.2f}%")
print('Test loss: %.3f' % zero_one_loss(y_test, y_test_pred))
```