Creating classification models to predict mental health through machine learning

Module Name: COMP2261
Date: 20/12/2021
Submitted as part of the degree of BSc Natural Sciences to the
Board of Examiners in the Department of Computer Sciences, Durham University

Abstract—We use a data set containing information about people's age and gender, as well as their answers to questions about their own mental health, to create a predictive model on people's self-defeating state. We use three different machine learning classification algorithms, train a model for each, and optimize them by tuning their hyperparameters. We then input some test data into these models to predict people's self-defeating state, both in a discrete and probabilistic form. We then compare the accuracy of our models through several methods, and investigate the predictive results obtained from our models.

Index Terms—Artificial intelligence, classification problems, machine learning, mental health

1 Introduction

Mental health has become an increasingly important issue in recent years. With suicide rates still high in some areas of the world, it is abundantly clear that we must address this issue now more than ever. In this report, we will explore how we can use machine learning to predict an individual's mental health state, based on age, gender, and answers to certain survey questions.

The data set that we will consider is a subset of a larger data set. The large set contains data for 1,071 individuals, including their answers to 32 different questions on their mental health, their age, gender, a scale score for 4 mental health states, and how accurate each individual thought their answers were (on a 0-100 scale).

For each of the questions, individuals were asked to provide an integer between 1 and 5 inclusive, depending on how much they agreed with the statement in the questions. The four scale scores are then calculated from some linear combination of eight answers (so each answer is used exactly once), with each of the scores ranging from 1.0 to 5.0 (each score is rounded to one decimal place).

For this problem, we will investigate the "Self-defeating" scale scores, as this is the state that relates the most to poor mental health. Based on some input data, we want to predict an individual's "Self-defeating" scale score. We will use classification models to do this, but instead of predicting a scale score, we will predict an interval for their scale score, as well as the probabilities of falling into each interval.

Our feature variables will be an individual's age, gender, and their answers to four of the questions used to initially calculate the "Self-defeating" score. This gives us some insight into how age and gender can relate to mental health, and including the answers to the four questions will make our models more accurate.

This problem has a major real-world application, as it could be used by companies, schools, universities etc., to predict an employee's or student's mental health state, simply by asking them four questions, and knowing their age or gender. Appropriate help could then be offered to individuals who are classified as having poor mental health, or likely to have poor mental health.

2 Pre-processing

2.1 Feature selection

The main variables we want to investigate are age and gender, but using only these two variables likely will not lead to very accurate models. So we can also include some of the questions that contribute to the "Self-defeating" scale score in our feature variables. From the data set, questions 4, 8, 12, 16, 20, 24, 28, and 32 contribute to the "Self-defeating" scale score.

Including all of them would be meaningless because we know there is a linear relationship between the eight questions and the scale score. When looking at the questions, we also note that some of them seem quite repetitive. So we will only use four of the eight questions, which will be questions 4, 12, 20, and 28. Note that during our data preparation, we will keep all 8 questions until the very end, as we will need to recalculate certain individuals' scale scores due to missing data.

2.2 Data preparation

The large data set we have been given is not very clean and is clearly corrupted in some places, so before we can start creating our models, there are some alterations to be made on the data. The first unusual thing we notice is that the range of the age column is 44,835, which is obviously a mistake.

So we observe the histogram of all the ages in our data and see that most of the ages lie within a plausible range of 10-100. We also find that the minimum age in the data is 14, and the maximum is 44,849, which explains the large range. Given that there is no way to know the actual age of

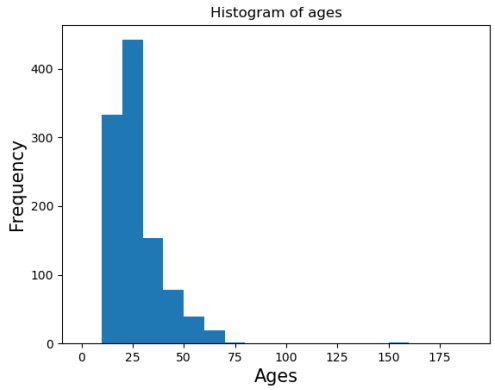

Fig. 1. Histogram of ages in data set

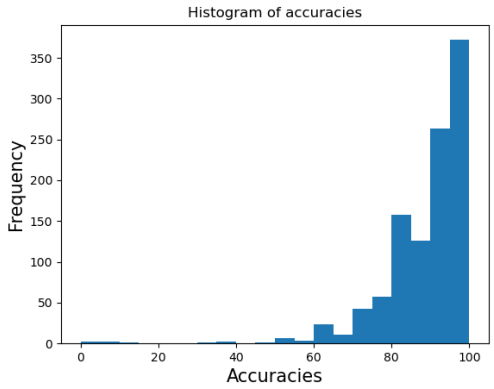

Fig. 2. Histogram of accuracies in the data set

the individuals with the implausible ages, we can just delete individuals whose age is greater than 100 from the data. As shown in Fig. 1, we will retain a large proportion of the data by doing this anyway.

We encounter a similar problem with the genders feature, but again one that is easily fixable. In this data set, gender is stored in a numerical format, with 1 representing male, 2 representing female, and 3 representing other.

TABLE 1
Counts for gender variable

Gender	Count
1	581
2	477
3	8
0	5

As shown in Table 1, there are five occurrences of 0, which is not one of the defined genders. As there is no possible way to determine the gender of these individuals, and the fact that there are only five instances, we can again remove these individuals from the data set.

Another factor we must consider is the accuracy of each individual's answers, which they gave as a number between 0 and 100 (0 indicated they did not want to be included in research). As with the ages, we can take a look at the histogram of the accuracies, shown in Fig. 2.

We see that if we remove all individuals whose accuracy was below 75, we will retain most of the data without being too inaccurate. Obviously, each accuracy in the data is just a rough estimate by the individual and does not represent a "true" accuracy, but if an individual feels as though their answers are at least 75% accurate, that seems like a strong enough confidence not to remove them from the data set.

After removing all of the individuals with an "invalid" age, gender, or accuracy from the data set (which we can do using Python's NumPy package), we see that the number of individuals in our new data set has dropped down to 969, so we have only removed about 10% of the original data set.

There is one more issue with the data set that we must address, and that is the case of missing answers in the questions. As previously mentioned, individuals were asked to submit a 1, 2, 3, 4, or 5 for each question based on the degree to which they agreed with the statement, however some individuals did not select an answer for some questions, which is represented by a -1 in the data. We see this with question 28 in Table 2.

TABLE 2
Counts for answer to question 28

Answer	Count
1	129
2	167
3	215
4	277
5	179
-1	2

The best way to deal with these missing values is to replace them with another value. There are three options here: replace by the mean, median, or mode.. Using the mean does not really work since we could obtain non-integer values which is not consistent with the rest of the answers. Using the median could again cause an issue, as we could obtain a value that appears very few times in the rest of the answers, making it not very likely to be that individual's actual answer.

So the best option would be to replace all missing values with the mode of that question, since that would be the individual's most likely answer. We can do this using scikit-learn's SimpleImputer() object. Then all we do is recalculate each individual's "Self-defeating" scale score using the new answers.

We also need to consider scaling some of our features, since the learning algorithms we will use can be quite sensitive to this. Two of the three algorithms will perform better with all of the features standardized (zero mean and unit variance), whereas the other will perform better with just the ages standardized. This means we will essentially create two "identical" data sets, using scikit-learn's StandardScaler() object.

To get our target variables, we sort the scale scores into classes so that class 0 contains scores in the interval [1, 2], class 1 contains scores in the interval (2, 3], class 2 contains

scores in the interval (3, 4], and class 3 contains scores in the interval (4, 5]. Every scale score now belongs to a class that we can try to predict using classification models.

Finally, we split the data sets into a training set and a test set. We can do this easily using the train_test_split() method from scikit-learn. We will use a 90/10 split on the data, and we will make it constant, so that each model fits the same training data. An example of an individual from our test sets (fully scaled and only age scaled) is shown in Table 3.

TABLE 3
Example individual from test sets

Q4	Q12	Q20	Q28	Age	Gender	Class
-1.58	-0.797	-0.988	-0.169	0.767	1.07	0
1	2	1	3	0.767	2	0

2.3 Learning algorithm selection

We want to be able to predict a scale score interval given our inputs, so the most appropriate type of machine learning algorithm to use, given that this is a supervised learning task, would be a classification algorithm. There are several algorithms we could use, but given that we also want to be able to predict probabilities for an individual falling into each class, we are restricted in what algorithms we can choose.

Fortunately, there are three algorithms that will do exactly what we want, and are easy to implement in Python using sci-kit learn. These algorithms are the K-Nearest Neighbors (KNN) model, Logistic Regression, and the Gaussian Process Classifier (GPC). Note that the KNN and Logistic Regression use the fully-scaled data set, whereas GPC uses the data set with only ages scaled.

3 Model Training and Evaluation

Using scikit-learn, we define three models, each corresponding to one of the three learning algorithms we selected. These models will be the KNeighborsClassifier(), LogisticRegression(), and GaussianProcessClassifier() objects. Each of these objects has a built-in fit() method, which will create a model based on training data that will be able to predict classes and class probabilities for test data.

3.1 Hyperparameter tuning

Firstly, however, we must tune each model's hyperparameters. A hyperparameter is a user-defined value that helps control the training process. We want to find the optimal set of hyperparameters for each model, i.e. the one that gives us the highest accuracy.

Scikit-learn again provides us with a fast and easy method to do this, using the GridSearchCV() object. This will take in a classifier and a set of possible hyperparameters for that classifier, and consider every possible combination of these hyperparameters in a fitting process on our data set. It then returns the combination which outputs the highest accuracy. For our classification models, we define accuracy as

$$\text{Accuracy} = \frac{\text{Correct Predictions Made}}{\text{Total Predictions Made}}.$$

This is the most common metric used for classification models, and gives us a good idea of how well each model can predict classes for new data.

The KNN and Logistic Regression models have several hyperparameters that we need to tune, but the GPC does not have any hyperparameters. Table 4 and Table 5 show the results of the KNN and Logistic Regression hyperparameter tuning.

TABLE 4
KNN optimal hyperparameters

n_neighbors	algorithm	metric
9	ball_tree	euclidean

TABLE 5
Logistic Regression optimal hyperparameters

penalty	solver
l2	saga

For KNN, we chose to tune three hyperparameters. The first, "n_neighbors", which we ranged over the values 1-19, is essentially equivalent to K in the name KNN. THe optimal value for our data is 9, which means every new data point seen will be compared to its nine nearest neighbors, determining its class.

A value that is too small can lead to overfitting, but a value of 9 will mean we won't encounter this issue. The "algorithm" hyperparameter is the algorithm used to compute the nearest neighbors, while "metric" is how the distance is calculated between points. For these hypeparameters, we ranged over several possible values given in the documentation [1], [2].

For Logistic Regression, we tuned 2 hyperparameters. "Penalty" concerns which type of penalty to implement in the cost function. We ranged over "l1", "l2", and "elastic-net", and did not consider "none", because adding a penalty will reduce overfitting. "Solver" concerns the algorithm to use in the optimization problem. We ranged over several possible values in the documentation [3].

3.2 Fitting and model comparison

Each of our model objects comes with a built-in fit() method in scikit-learn, so all we need to do is define the hyperparameter values for each model, and call the fit() method on the training set. Before we do this however, we will perform cross-validation on the entire data set for each of our models. We can do this using scikit-learn's cross_validate() method, and it will give us a better insight into how accurate each model is. From Table 6, we see that all three models have fairly good accuracy, but the GPC model performs slightly better than the other two.

We can now fit our models to the training set, using the fit() method, and observe the results when we use the test set as input. To do this, we call the predict() and predict_proba() methods, which are also built-in to each model in scikit-learn. Table 7 shows the results of the predict_proba() method for each model, for the individual in Table 3.

TABLE 6
Cross-validation model accuracies

Model	Accuracy
Logistic Regression	0.814
KNN	0.812
GPC	0.809

TABLE 7
Predicted class probabilities for individual in Table 3

Model	Class 0	Class 1	Class 2	Class 3
KNN	0	0.556	0.444	0
Logistic Regression	0.000429	0.695	0.305	0.000238
GPC	0.206	0.330	0.257	0.207

We see that there is a semblance of similarity in the outputs from the KNN and Logistic Regression model, however the GPC model seems as though it is able to predict more realistic probabilities, as it does not predict values so close to 0. It is noteworthy that the KNN model is restricted in its predictions, and can only predict probabilities that are a multiple of 1/k, where k is equal to the "n_neighbors" parameter. Making use of the predict() methods on the test set, which predicts the class value instead of class probabilities for a certain input, we can compare each model's confusion matrix too. Figures 3, 4, and 5 display these.

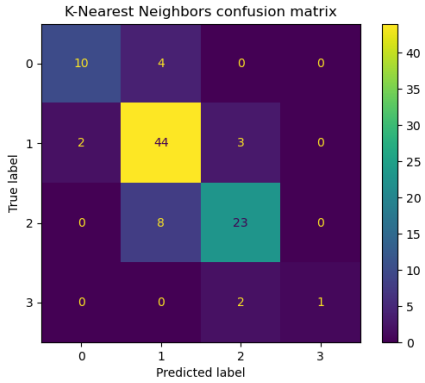

Fig. 3. KNN confusion matrix

We see that all three models do a very good job in predicting the first three classes, however the GPC and KNN models are actually able to predict an individual falling in class 3, which is the class representing the worst mental health in our problem, and is therefore arguably the most important class to be able to predict. Given the more realistic nature of the probabilities predicted by the GPC model, it's ability to predict higher classes, and the fact that it's accuracy in Table 6 was only marginally worse than that of the other models, we could say that the GPC model is the best one for our problem.

4 Conclusion

We created and compared three different machine learning models to predict a class and class probabilities for an individual's "Self-defeating" mental state, that could be used by various institutions to identify people who may need support with their mental health.

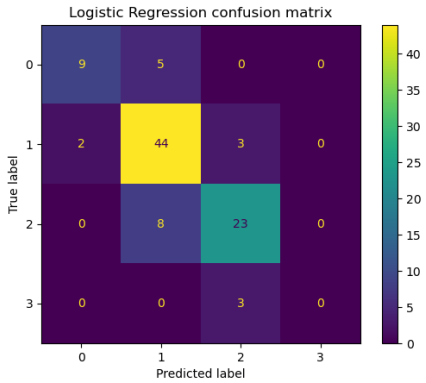

Fig. 4. Logistic Regression confusion matrix

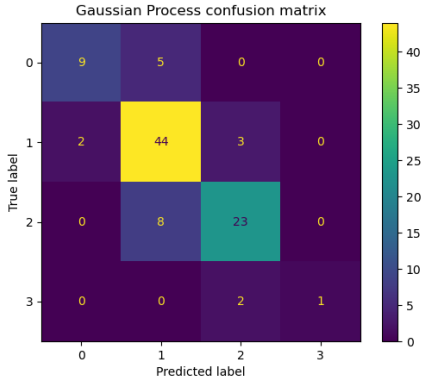

Fig. 5. GPC confusion matrix

The models take in a small number of features, including an individual's age, gender, and their answers to four mental health related questions. This simplifies the data collection process for the user of the model.

We found that our models each had a high level of accuracy, however the GPC model stood out in it's ability to more accurately predict individuals having worse mental health, leading to the conclusion that it may be the better model to use for this problem.

Our models could have been more accurate had we just used all eight questions relating to the "Self-defeating" score in our features, however asking an individual eight questions that are somewhat repetitive could be time-consuming and may cause them to not answer honestly. This would also be redundant since there is a known linear relationship between the scale score and the eight questions.

References

[1] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
[2] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.DistanceMetric.html
[3] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html