In Class Competition #3: DrugUse Probabilities

Team 1

Mehak Juneja, Srija Pinnamaneni, and Victor Suhendra

Overview:

For this project, we are working with a dataset that contains information on homeless youth and their drug use. The dataset contains many features indicating their demographics, education, sleep patterns, rent payments, and other profiling variables to help determine which drug they prefer. Using this data we will be forecasting the probability of a youth using Cocaine, Crack, Heroin, Meth, Ecstasy, and Marijuna making it a multi classification task. At first glance, this dataset contains a structured table of columns as our predictions and rows as individual youth.

Data Preprocessing:

We housed our data in a shared github repository for each access that can be efficiently read into a Google collaboratory notebook. This allowed us to maintain a consistent work space between all the members of our group. We knew this dataset included people who were not homeless and so we wanted to subset only the homeless youth. To start off we extracted the homeless data from the original data by filtering on the variable “selfhomeless” that had the label 2, which meant that the person declared themselves as homeless. We did this to ensure we only wanted to look at drug use within the population of homeless youth. We set this new data dataframe as the master. Our first approach, that we did not end up using was to add up all the dummy variables called “a12monthhomeless” and adding the values into a new column and removing any rows that had 0 in the new column, however we realized the “selfhomeless” variable would be a more accurate representation of the homeless population.

Next, we had to focus on feature engineering. We chose a few features that we believed impacted drug use for the baseline: “education,” “inschool,” and “fostercare.” Since there were only three variables, we chose to use dropna() as there were few rows that didn’t have all three. As we adjusted the features, we realized that most of these features, such as education, were numerical, but should have been categorical. For example, in education, ‘one to three semesters of college” was assigned a value of 7, while a “bachelor's degree” was assigned a score of 5. This is obviously a problem as a bachelor's degree is better than one to three semesters of college, and it makes much more sense for it to be encoded as categorical variables. So we created dummy variables instead since the numbers weren’t representative of the categories using one hot encoding This also created a dummy variable for each category with na values, so we didn’t need to remove the na values anymore. However to get a score which reached the baseline, we needed more features than simply these 3. We looked through the documentation to find features which interested us, before then checking the dataset to see how many of its values were missing(as some of the columns had a significant amount of NA values). In the end we added ‘birthsex,’ ‘sexualorientation,’ and ‘ethnic’ as we felt that biological gender and sexual orientation likely played a role in terms of the communities a person blended in with and thus what types of drugs

they would consume in general. Moreover we added ‘screen1sleep’ (where a person slept the previous night) as we felt this would be quite reflective of their current situation and thus a good predictor. Adding these features, we increased our score by about 0.05 to 0.56.

However after this, when we tried adding in several more columns, our score consistently decreased. Although maybe we were just unlucky with the columns we picked, we realised the inadequacies of this manual selection of features approach we used, as compared to the other projects, this dataset had significantly more columns (>1000).

We wanted an approach that was more generalizable. Looking through the documentation, we realized that most of the numerical features were better suited as categorical features(except the features related to age). We thus tried an extremely naive approach in which we worked with all the features in the dataset and transformed all of their values into strings, before doing one hot encoding, in this way every column would be encoded. There were some obvious flaws in this approach, such as some of the features had all NA values, some of the features should be numerical and should not be encoded, and since we encoded all the columns, the number of columns skyrocketed to almost 10,000, taking our model almost an hour to finish running as we did 10-fold cross validation as well. However using this naive approach, we received a much better score of 0.71.

We decided it would be better to try to improve on this naive approach as it was significantly better than our original approach. Keeping all columns, especially considering that some columns had most if not all NA values was obviously wrong. We decided to drop columns if they were above a certain threshold of NA values. We also decided to try different models. After several attempts, we found that using DecisionTreeClassifier with a threshold of 0.5 gave us a score of 0.99.

The variables that were used to identify whether a person uses a drug or not were the “druguse_30” variables. They ranged from “druguse_30_1” to “druguse_30_6”. However these numbers ranged from 1-6, depending on how much they used the drug. As such, we changed these features to binary to make it easier to work with. Although this caused it to lose some information, we could not figure out a better way to do it. In these druguse columns, we changed the values of 1 to 0 and 2-6 to 1. We also needed to predict whether a person used drugs or not. To create this new column, we set it to 1 if druguse 1-6 were all 0 (That is if a person did not use any of the drugs) and 0 otherwise.

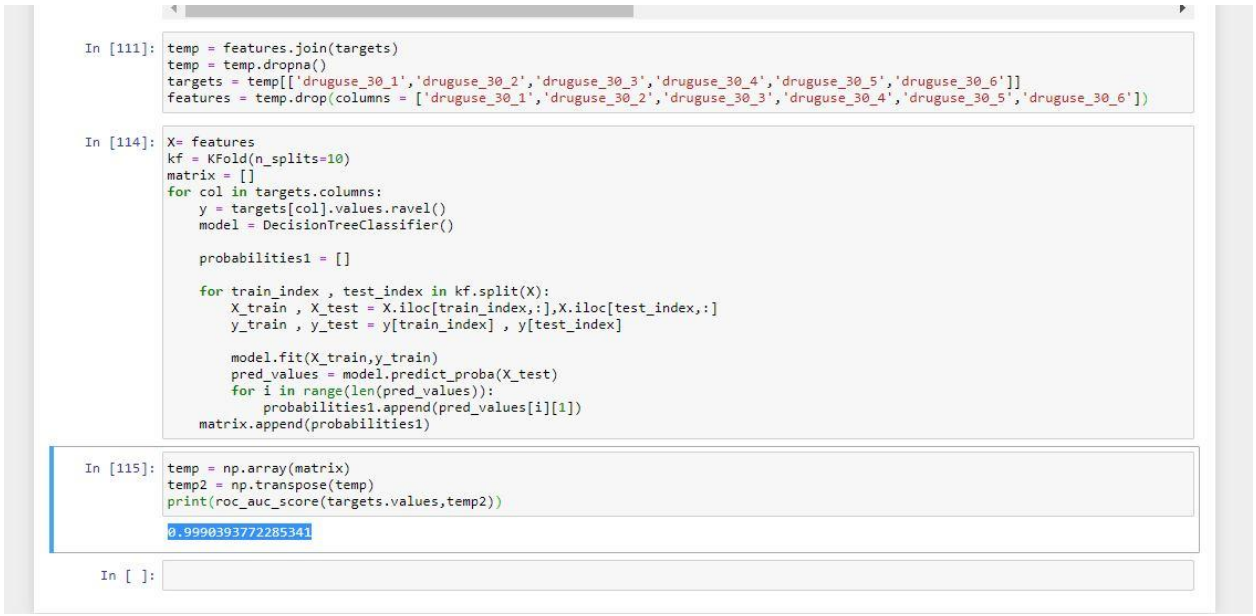
Fitting and Modeling:

To divide the data into train and test data, to determine the effectiveness of the model we first used k-fold with 10 splits on the feature selected homeless data. The first model we tried

was logistic regression with multiple outputs. This was a good model to start off with, but we ended up with a really low score of ~0.45. We then tried GaussianNB, which helped us get a better score of 0.56. We tried to add more features to the Gaussian model, but in the end, the score just got worse. This is why we moved on to a KNeighbors Classifier. We manually adjusted the number of neighbors and found that 15 worked relatively well. For our naive approach, and using KNeighboursClassifier, we set this threshold to remove NA values as 0.5 (if the column has more than 50% NA values it is removed), which resulted in an improved score of 0.73. We tried a different threshold of 0.8 but received a lower score of 0.72.

```
In [86]: X= features
         kf = KFold(n_splits=10)
         matrix = []
         for col in targets.columns:
             y = targets[col].values.ravel()
             model = KNeighborsClassifier(n_neighbors=15)

             probabilities1 = []

             for train_index , test_index in kf.split(X):
                 X_train , X_test = X.iloc[train_index,:],X.iloc[test_index,:]
                 y_train , y_test = y[train_index] , y[test_index]

                 model.fit(X_train,y_train)
                 pred_values = model.predict_proba(X_test)
                 for i in range(len(pred_values)):
                     probabilities1.append(pred_values[i][1])
             matrix.append(probabilities1)

In [87]: temp = np.array(matrix)
         temp2 = np.transpose(temp)
         print(roc_auc_score(targets.values,temp2))

         0.7341487483320446
```

We decided to try DecisionTreeClassifier instead. Using this model and set the threshold to 0.5 as that is what worked best earlier, we received a score of 0.999



```
In [111]: temp = features.join(targets)
          temp = temp.dropna()
          targets = temp[['druguse_30_1','druguse_30_2','druguse_30_3','druguse_30_4','druguse_30_5','druguse_30_6']]
          features = temp.drop(columns = ['druguse_30_1','druguse_30_2','druguse_30_3','druguse_30_4','druguse_30_5','druguse_30_6'])

In [114]: X= features
          kf = KFold(n_splits=10)
          matrix = []
          for col in targets.columns:
              y = targets[col].values.ravel()
              model = DecisionTreeClassifier()

              probabilities1 = []

              for train_index , test_index in kf.split(X):
                  X_train , X_test = X.iloc[train_index,:],X.iloc[test_index,:]
                  y_train , y_test = y[train_index] , y[test_index]

                  model.fit(X_train,y_train)
                  pred_values = model.predict_proba(X_test)
                  for i in range(len(pred_values)):
                      probabilities1.append(pred_values[i][1])
              matrix.append(probabilities1)

In [115]: temp = np.array(matrix)
          temp2 = np.transpose(temp)
          print(roc_auc_score(targets.values,temp2))

          0.9990393772285341

In [ ]:
```

Methods We Considered:

While working through this project, we considered many approaches that evidently didn't make it into the final submission. During feature selections we took a simple approach of encoding all the numerical values to characters and then creating dummy variables from there. With this we noticed there were many columns and even though it improved our accuracy, there was a high risk of noise and linear dependency between a lot of our features, in turn it probably affecting our accuracy. We would have liked to plot a correlation matrix and extract the values that were above a certain threshold to see which brainless could be removed since that information was already provided within another. This would have reduced the dimensionality of your data while keeping the predictors necessary to make accurate predictions.

Another approach we would have implemented given more time would be a python feature selection method, both forward and backward iterations. This could have optimized our features set as well and we could have manually removed and added features the function might have ignored. Overall, this project was a challenging one as it was multi class with many target columns which created much confusion for us on where to start. After figuring out cross validation was the right way to implement all the classes we were able to reach a decent accuracy score.