ReachSplitter Classifier Write-Up

Version Last Updated: July 27, 2022

Emily Nguyen

July 28, 2022

Contents

1 Introduction

This document is a software manual of ReachSplitter's classifier created using LaTeX and Overleaf. It provides a general overview for processing data and producing results.

2 Setup

2.1 Software

- Jupyter Notebook (local or nonlocal setup with Berkeley Datahub[1]
- Python
- PyCharm IDE

2.2 Files

- Processes Labels: label_explorer.ipynb
- Manipulates data and labels into ML ready format: EDA_SampleReaches3.ipynb
- Performs classification training and visualization: Classifier2_BN_edit.ipynb
- Robot and 3D Kinematics from ReachMaster experimental setup: Sample_Reaches.pkl

2.3 Directory

- All data file types (csv, pkl, pickle) are in a "DataFrames" folder in the current working directory for organizational purposes.
- All image types (jpeg, pdf, png) are in a "Plots" folder in the current working directory. This folder is created automatically by running label_explorer.ipynb if one does not already exist.

[1] https://datahub.berkeley.edu/

3 Descriptions

3.1 Label Processing

In label_explorer.ipynb , labels are processed into ML ready formats and are saved as csv files.

1. Vectorize all manually collected DLC video labels.

```
def make_vectorized_labels(blist):
 """Vectorizes list of DLC video trial labels for use in
    ML-standard format
    Converts labels which hand and tug vs no tug string
       labels into numbers.

 Attributes
 -------------
 blist: list, of trial labels for a specific
    rat,date,session
       For more robust description please see github

 Returns
 ----------
 new_list: array, of lists of numeric labels.
       One list corresponds to one labeled trial.
 ind_total: array of lists of reaching indices .
       Currently all empty.
 """
```

2. Converte vectorized labels into a single dataframe.

```
def make_vectorized_labels_to_df(labels):
   """Convert return value from make_vectorized_labels
      into a pandas df

   Args:
      labels (arr of lists): return value from
         make_vectorized_labels

   Returns:
      newdf(df)

   Examples:
      >>> l18l, ev18 = CU.make_vectorized_labels(l18)
```

```
        >>> make_vectorized_labels_to_df(l18l)
    """
```

3. One-Hot all relevant columns. For example, for the null vs not null label column, all trials with a null label (labeled with a 0 according to the labeling key) will be assigned a value of 1 while successful reaches will be assigned a value of 0.

```
def onehot_nulls(type_labels_):
    # kwargs: n_f_fr_s_st: Trial type (null, failed,
        failed_rew,s ,succ_tug), label key [0, 1, 2, 3, 4]
    null_labels = np.zeros((type_labels_.shape[0]))
    null_labels[np.where(type_labels_ == 0)] = 1 # 1 if
        null, 0 if real trial
    return null_labels

def onehot_num_reaches(num_labels_):
    num_r_labels = np.zeros((num_labels_.shape[0])) # 0
        vector
    num_r_labels[np.where(num_labels_ > 1)] = 1 # 0 if <=1,
        1 if > 1 reaches
    return num_r_labels

def hand_type_onehot(hand_labels_):
    hand_type_label = np.zeros((hand_labels_.shape[0]))
    hand_type_label[np.where(hand_labels_ > 1)] = 1 #
        classify all non r,l reaches as 1
    return hand_type_label

def multiple_reaches_onehot(labels):
    new_labels = np.zeros((labels.shape[0]))
    new_labels[np.where(labels >= 4)] = 2 # classify 4+, 2,
        3 reaches
    new_labels[np.where(labels == 3)] = 1
    new_labels[np.where(labels == 2)] = 0
    return new_labels
```

4. Save label dataframe as LABELS.csv which contains the trial type, num reaches, and which hand as binary float fields and some nonbin fields in the “DataFrames” folder in current working directory.

5. Perform summary statistics and visualizations of class balances, then saved bar plots as PDFs.

3.2 Exploratory Data Analysis + More Processing

In EDA_SampleReaches3.ipynb , labels from the previous section and the experimental data (Sample_Reaches.pkl) are processed together into ML ready formats, then saved as csv files.

1. Load (Sample_Reaches.pkl) and LABELS.csv as Pandas DataFrames.

2. Convert all numeric data to types float or int, such as the “Date” column. Remove columns containing NaN values and unnecessary columns (e.g. drop column "Unnamed: 0").

```
df["Date"] = df["Date"].astype(int)
labels = labels.drop(["Unnamed: 0"], axis=1)
```

3. Match each trial to its time series and label.

   a) filterdf: Define function that takes in trial IDs (rat, date, session, trial number) and returns a DataFrame of the time series (number of rows that match the trial IDs).

   b) Iterate through each trial ID in the label df. Find the corresponding time series in the feature df using filterdf. Store each time series in a list. Result is a list of DataFrames where each dataframe corresponds to one trial in the label df (order matters).

   CHECKS: Make sure “filterdf” returns type DataFrame. “filterdf” should error when it cannot find the matching data. When comparing trial IDs from the label df to the feature df, make sure the format and types of the trial IDs are the same (e.g. rat in label df is str ‘RM16’ but rat in feature df is int 16).

4. Convert from continuous to Discrete - get 5, 25, 50, 75, 95 percentiles for each time series.

   a) Make a new df. Populate with new feature columns and each row represents a trial.

   b) If the feature is a time series: take the sum stats for each df. store as new feature columns and rows. Else, store value in new df. Result is a new df with reduced dimensionality (trials by features). i.e. gets rid of time component.

5. Save resulting Pandas DataFrame as DISCRETE_DF.csv. Then, appended the labels dataframe columns to it. Saved as DISCRETE_andLABELS.csv, a file with no PCA standardized discrete features with labels.

6. Taking the resulting data from the previous steps, standardize it for PCA.

```
   ## Standardizing the features
"""
Each feature of your data should be normally distributed
such that it will scale the distribution to a mean of zero
   and a standard deviation of one.
"""
discrete_df2 = StandardScaler().fit_transform(discrete_df)
discrete_df = pd.DataFrame(discrete_df2, columns =
   discrete_df.columns)
```

7. Visualize[2] results from PCA and save plots in "Plots" folder in current working directory. Some example visualization are principal component 1 vs principal component 2 scatter plots colored by class type and the first 10 principal components vs percentage of their explained variance bar plots.

3.3 PCA + Training Classifiers

In Classifier2_BN_edit.ipynb , PCA is performed on the processed data and labels from the previous section. Then, the classifiers are trained according to the classification hierarchy, accompanied by visualizations.

1. Load (DISCRETE_DF.csv) and LABELS.csv as Pandas DataFrames.
2. Check for and remove unnecessary columns. For instance, remove the 50th percentile for xyz_rob and sensor because it is just the handle position and does not provide any new information. Standardize the features.
3. Define PCA function to perform on the features. Save the PCA models using the "joblib.dump" function.

```
def doPCA(df, n, concat = True):
   """
   Performs PCA on df.
   Returns dataframe with PCA columns and pca object. """
   pca = PCA(n_components=n)
   components = pca.fit_transform(df)
```

[2] https://plotly.com/python/pca-visualization/

```
    PCA_col_names = ["PC" + str(x) for x in range(1,
        pca.n_components_+1)]
    pca_df = pd.DataFrame(components, columns=PCA_col_names)
    if concat:
        pca_df = pd.concat([pca_df, new_labels], axis=1)

    return pca_df, pca
```

4. Define a "classifier" class which contains all relevant functions for a classifier such as a "split" function which splits data into training, validation, and test sets.

```
class classifier:
    def __init__(self, data, y_name):
        self.data = data
        self.y_name = y_name

    ### Define X and y from df ###
    def filter_y(df, y_name):
        """
        Separates X and y in df
        df: Df
        y_name: str col name
        Returns X and y separted
        """
        new_df = df.copy()
        for col in new_df.columns:
            if (not "PC" in col) and (col!=y_name):
                new_df= new_df.drop([col], axis=1)
        X = new_df.drop([y_name], axis=1)
        y = new_df[y_name]#.to_frame()
        return new_df, X, y

    ### Adjust Class Imbalances ###
    ...
    ### Visualize Class Balances ###
    ...
    ### Split Data ###
    ...
    ### Determine Chance by Shuffling Y ###
    ...
    ### Hyperparameter Tuning ###
    ...
    ### Model Scoring ###
```

```
   def main(...)
```

5. Execute the classifier object in the previous step on each class in the classification hierarchy by calling the main function. This produces class balance bar plots, 2D and 3D PCA scatter plots of the first two and three PCS colored by class respectively, accuracy scores, and ROC-AUC curves. An example is given below.

```
# class: null v not null
null_classifier = classifier(data, 'Trial Type')
convert_dict = { 0:"not null", 1:"null"} # for labeling
   plot axis.
null_classifier.main(convert_dict, sn_k=False)
```

6. Visualize optimum number of PCs to use to train the model by plotting number of PCs vs accuracy and roc-auc scores.

```
def plotPCAROC(data, y_name, maximum):
  """
  data: training data
  y_name: str of class type e.g. ``Num Reaches''
  maximum: 1, 2...maximum-1, maximum number of PCs to use
     to train the base random forest model
  Produces plots of # of given PCs to a model
  vs the model's training, validation, and test
     accuracies and roc-auc scores.
  Saves plots to plots folder.
  """
```

7. Visualize comparing different models. The current implementations explores six classifiers: Random Forest, Logistic Regression, SVM, LDA, KNN, and AdaBoost classifiers.

```
    def plotModelsvAcc(data, y_name, models):
    """
     data: training data
     y_name: str of class type e.g. ``Num Reaches''
     models: list of different models to compare
    Plots models vs their corresponding chance, train,
       val, and roc-auc scores.
    """
```

8. Save the desired trained models, plots, and performance scores.

4 Appendix

Documentation on how to collect DLC video labels can be found here[3] (login with UC Berkeley email).

A technical report of the classifier was written: Emily Nguyen. Machine Learning Methods for Supervised Classification of Behavioral Time Series Data, B.A. Thesis; University of California, Berkeley, May, 2022.

For the number of reaches classification, an ensemble method was found to be more accurate. The documentation and code to do this is not included in this write up.

[3] https://docs.google.com/document/d/1RX-hwpxBa_DLCVeY3CWWe2lQpKUCy_vXPdUgoHNAguE/edit?usp=sharing

Figure 1: Thank you.