ML w/ PYTHON
Winter Holiday 2022
LinkedIn

29/12/22

- supervised, unsupervised *(input, model, output)* and reinforcement learning *(agent, environment)* [(primary entities)]
    - Agent figures the best way to accomplish a task through a series of cycles in which the agent takes an action and receives an immediate positive/negative feedback on the action from the environment
    - After a no. of cycles, agent figures the optimal sequence of actions to take in order to accomplish the task
- ML - interested in predicting data vs Statistics - interested in the relationship btw variables (using inference)
    - *Statistical learning:* the overlap of Ml and statistics
- Supervised Learning: Experience, E ; Class of Tasks, T; Performance measure, P
    - Experience is the training data
    - Performance measure is predictive accuracy
- Reinforcement Learning, 2 learning objectives
    - 1. Finding previously unknown sol to a problem
    - 2. Finding online sols to problems that arise due to unforeseen circumstances
    - Table of states and rewards: policy table
    - Agent looks at each action and reward and chooses the action with the highest reward: exploitation
    - Terminal state: state when the episode is over [the coin cycle of learning has ended]

    - At the end, the reward associated with the action that lead the agent to victory is updated by the environment to its respective value in the policy table: backup
    - Reward associated with the agents second best move is also backed up in the policy table
    - In the next game, if same move is encountered, agent with the pursue the same action assuming the action helps to win
    - Exploration vs exploitation trade-off: if left unchecked an agent will always prefer to take actions it has tried in the past
        - Tho need exploration to find a new sequence of actions with potentially higher reward, agent needs to try actions not selected before/ actions that don't initially appear to maximize reward.
            - Exploration: choosing actions with no consideration of reward
- Steps of ML:
    - Data collection
    - Data exploration: understanding data by describing and visualizing it
        - Look for missing, inconsistent, duplicates or outliers in data
    - Data preparation: modifying data so it works for the type of ML u intent to do
        - Resolving data quality issues such as missing data, noisy data, outlier data and class imbalance
        - Also modify or transform structure of data to make it easier to work w/
            - Normalizing data, reducing no. of rows and columns in data
    - Modeling: applying a ML approach to data
    - Evaluation: assess how well ML approach worked
        - Iterate btw modelling and evaluation to find best approach
    - Actionable insight: identify what to do based on the results of the NL approach u chose
- Things to consider when collecting data:
    - Ensuring data is accurate
        - Ground truth data; ground truth refers to the correct or "true" answer to a specific problem or question i.e. the reality you want to model with your supervised ML.

            - Ground truth data can either come w an existing label based on a prior event or can require that a label be initially assigned to it by domain experts
    - Relevance: type of data collected to describe an observation should be relevant in explaining the label or the response associated with the observation
    - Quantity: amount of data required depends on the type of ML approach chosen
    - Variability (of data)
    - Ethics
        - Avoid bias
- PANDAS
    - Import pandas as pd
    - Pandas series
        - pd.Series(list)
        - one-dimensional array. It holds any data type supported in Python and uses labels to locate each data value for retrieval. These labels form the index, and they can be strings or integers. A Series is the main data structure in the pandas framework for storing one-dimensional data
    - Pandas dataframe
        - pd.DataFrame(list)
        - 2D array, heterogenous, labeled rows and columns; collection of pandas series all sharing same index
        - pd.DataFrame(list, columns = list2)
    - Import from csv file
        - pd.read_csv("brics.csv")
    - Import from excel file
        - pd.read_excel("brics.xlsx")
        - pd.read_excel("brics.xlsx", sheet_name = "Summits")

DESCRIBING DATA

- Instance: each individual example of a target concept i.e. each row in the table

    - A dataset contains several instances
    - A.k.a a record or observation
- Feature: property or characteristic of an instance i.e. a column in the table
    - A.k.a a variable
    - Types: categorical features (holds discrete data), Continuous features (integer or real no.)
- Class/Response: attribute or feature that is described by the other features within an instance
    - Response if continuous, class if discrete
- Dimensionality: no. of features in a dataset
    - Increase dimensionality = increased detail about each instance
- Sparsity and Density: degree to which data exists in a dataset
    - Sparcity: ow much data is missing or undefined
    - Density is the complement of sparsity
- Summarizing data in Python
    - .info()
    - .head() - Head method: to get sneak peak of data in the dataframe
    - .describe() - returns a statistical summary for each of the columns in a DataFrame
        - Statics depends on the data type of the column
        - file_stored_in_variable[‘[column_name’]].describe()
        - file_stored_in_variable[[‘column_name’]].value_counts()
            - Gives count of each of the instances
            - file_stored_in_variable[['column_name']].value_counts(normalize = True) : to get it as a percentage count
        - file_stored_in_variable[['column_name']].mean()
    - Group-level Aggregations
        - washers.groupby('BrandName')[['Volume']].mean() : mean vol of each brand
        - washers.groupby('BrandName')[['Volume']].mean().sort_values(by = 'Volume')
        - washers.groupby('BrandName')[['Volume']].agg(['mean', 'median', 'min', 'max'])

VISUALIZING DATA

- Comparison
    - Box plots
    - Need to first create a pivot table
        - vehicles.pivot(columns = 'drive', values = 'co2emissions') // values
    - To plot box plot: vehicles.pivot(columns = 'drive', values = 'co2emissions').plot(kind = 'box', figsize = (10,6)) // for a 10x6 size figure
- Relationship: shows correlation btw one or more variables
    - Scatter graphs
    - vehicles.plot(kind = 'scatter', x= 'citympg', y = 'co2emissions')
- Distribution: shows statistical distribution of values of a feature
    - histograms
    - vehicles['co2emissions'].plot(kind='hist')
- Composition: shows component markup of data
    - Stack bar chart
    - vehicles.groupby('year')['drive'].value_counts()
    - vehicles.groupby('year')['drive'].value_counts().unstack()
    - vehicles.groupby('year')['drive'].value_counts().unstack().plot(kind = 'bar', stacked = True, figsize = (10,6))
- %matplotlib inline //plot method of pandas provides an abstraction of the matplotlib functions //use this method to ensure illustrations appear after the code

DATA QUALITY ISSUES

- Either remove missing data or enter -1/NA in their instances
- Else, Imputation, use of systematic approach to fill in missing data by using the most probable substitute vales
    - Median Imputation: use median of non-missing values
    - mask = students['State'].isnull() - to check for missing values
    - students[mask] - contains the table of missing values

    - students.dropna() - remove all instances w/ missing values
    - students = students.dropna(subset=['State', 'Zip'], how = 'all') - remove all instances which miss State and Zip values
    - students.dropna(axis =1) - remove all column which contain missing data
    - students = students.dropna(axis =1, thresh = 10) - remove columns w/ more than 10 missing values
    - students = students.fillna({'Gender':'Female'}) - replace na values in a column // inside fillna add a dictionary with the key as the column name and the value is the item to be replaced w/
    - students = students.fillna({'Age':students['Age'].median()}) - replace with median of values in column
    - To replace a specific value :
        - mask = (students['City'] == 'Granger') & (students['State'] == 'IN') // specify the value in a variable
        - students.loc[mask, :] //locate in students the row mask and colon represents all columns
        - students.loc[mask, 'Zip'] = 46799 // to replace a value with a certain value
- Outliers, need to decide if they are useful for the ML algo
    - Proportion of instances that belong to each class labels are, class distribution
    - Class imbalance: when distribution of values for the class is not uniform
        - If not well accounted for can lead to misleading predictions
        - To account for this, one approach: “under sample the majority class” - remove some of the majority class instances to even the class distribution

NORMALIZING DATA

- Normalization/Standardization: ensure values share a common property
- Involves scaling the data to fall within a small or specified range
- Reduces complexity of models and makes results easier to interpret

- Z-Score Normalization: transform data so that it has a mean of 0 and std of 1. Normalized value v' is

$$v' = \frac{v - \bar{F}}{\sigma_F}$$

    - Original value, v; feature F; F bar, feature mean; std of F, sigma F
    - from sklearn.preprocessing import StandardScaler
    - co2emissions_zm = StandardScaler().fit_transform(vehicles[['co2emissions']])
    - co2emissions_zm = pd.DataFrame(co2emissions_zm, columns = ['co2emissions'])
    - co2emissions_zm.describe()
    - co2emissions_zm.plot(kind = 'hist', bins = 20, figsize = (10, 6))
- Min-Max Normalization: transform data from measured units to a new interval, which goes from $lower_F$ to $upper_F$

$$v' = \frac{v - min_F}{max_F - min_F}(upper_F - lower_F) + lower_F$$

- v is the current value of feature F and v' is the normalized value

    - Used when no outliers
    - scikit-learn package provides several methods for transforming data in python for min-max normalization
    - First import minmax scalar object from the sk learn pre-processing sub package
    - from sklearn.preprocessing import MinMaxScaler
    - The use fit transform object to normalize data
    - co2emissions_mm = MinMaxScaler().fit_transform(vehicles[['co2emissions']])
    - co2emissions_mm = pd.DataFrame(co2emissions_mm, columns = ['co2emissions'])
    - co2emissions_mm.describe()

    - co2emissions_mm.plot(kind = 'hist', bins = 20, figsize = (10, 6))
- Log Transformation: transform data by replacing original values of data with its logarithm

$$v' = log(v)$$

- v is the original value and v' is the normalized value.

    - Usually log 2 or 19
    - Log transformation only works for values that are positive
    - This approach minimizes distance btw outliers and rest of data
- To plot histogram: vehicles[['co2emissions']].plot(kind = 'hist', bins = 20, figsize = (10, 6)) // bins are the height of the histogram
    - The towers or bars of a histogram are called bins. The height of each bin shows how many values from that data fall into that range.
    - We can pass an integer in bins stating how many bins/towers to be created in the histogram and the width of each bin is then changed accordingly.

SAMPLING DATA

- Process of selecting a subset of instances in a dataset as a proxy for the whole
- population : original dataset; sample: subset
- Simple random sampling
    - Random sampling w/o replacement
    - Random sampling w/ replacement
        - Bootstrapping: used to evaluate and estimate performance of a supervised ML model when have v little data
- Stratified sampling: ensures distribution of values for a particular feature within the sample matches the distribution of values for the same feature in the overall population
    - Instances in the original data population are divided into homogenous subgroups known as strata

First need to separate the dependent variable from the independent variables

Decide ‘co2emissions’ is the dependent variable column and create a data frame called y based on that column alone

response = 'co2emissions' // create string variable response for the name of the dependent variable column co2 emission

y = vehicles[[response]] //create dataframe by subsetting based on the response variable

y.head() //preview dataframe

predictors = list(vehicles.columns) // to create dataframe of independent variables, first create list predictors to contain all the columns in the vehicles dataframe

predictors

predictors.remove(response) // remove co2emissions column from the list

x = vehicles[predictors] //create dataframe

x.head()

- SIMPLE RANDOM SAMPLING

//Train Test Split function from sk learn model selection subpackage

from sklearn.model_selection import train_test_split

//call data sets x_train, x_test, y_train, y_test = train_test_split(x, y)

//x_train = independent variables of the training set, x_test = of the test set, y_train = dependent variables of the training set, y_test = train_test_split(x, y)

.shape() to get the size

x_train.shape , etc

//by default train test split function allocated 25% of the original data to the test set.

//To override it set either train size argument or the test size argument of the function

```
x_train, x_test, y_train, y_test = train_test_split(x, y,
                                                    test_size = 0.4)
```

- STRATIFIED RANDOM SAMPLING

```
x_train, x_test, y_train, y_test = train_test_split(x, y,
                                                    test_size = 0.01,
                                                    random_state = 1234)
```

//also use random_state argument to ensure that the result in the tutorial stay uniform for distribution i.e. not actually needed

If objective for using stratified random sampling is to maintain the same distribution of values for the drive column between the original training and test sets, then get the distribution for the drive column in the original data, here x

```
x['drive'].value_counts(normalize = True) // get normalized values of column drive
```

```
x_test['drive'].value_counts(normalize = True) //get normalized values of the test column
```

Both distributions will be similar but with noticeable differences

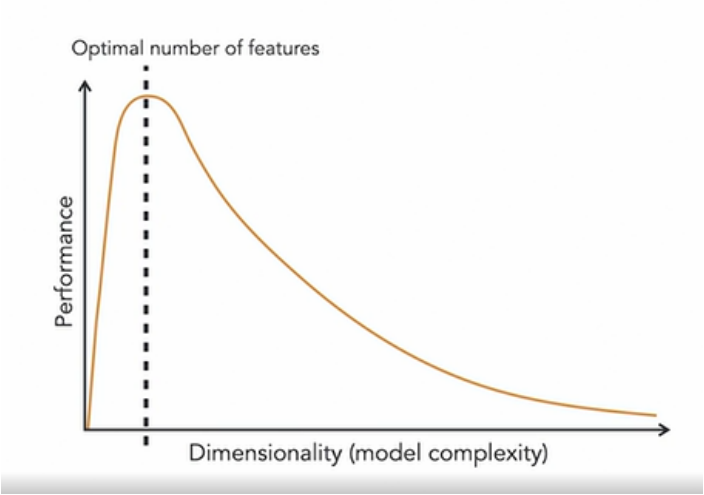
To do stratified sampling:

```
x_train, x_test, y_train, y_test = train_test_split(x, y,
                                                    test_size = 0.01,
                                                    random_state = 1234,
                                                    stratify = x['drive'])
```

REDUCING THE DIMENSIONALITY OF DATA

- To reduce the size/complexity of data
    - Sampling one approach, dimensionality reduction is another
- Process of reducing the number of features in a dataset prior to modeling

- Helps avoid the curse of dimensionality
    - Phenomenon in ML that describes the eventual reduction in the performance of a model as the dimensionality of the training data increases
    - As complexity (p) increases, amount of data (n) needed to generalize accurately grows exponentially
    - As increase features in the model, will also have increase instances in the training data else performance will degrade
    - If n is held constant, performance will eventua;;y diminish as p increases



- 2 ways of reducing dimensionality: feature selection & feature extraction
- Feature selection: process of identifying the minimal set of features needed to build a good model (reasonably close to optimal model)
    - Assumption: some features/independent variables are either redundant or irrelevant and can be removed w/o much impact on the model performance
    - A.k.a variable subset selection
- Feature extraction: use of math func to transform high-dimensional data unto lower dimensions
    - Final set of features completely diff from original set
    - New features a projection of original features
    - A.k.a feature projection

- New features may be complicated and may not make sense to humans

TRAIN THE MODEL

- For linear regression

from sklearn.linear_model import LinearRegression

model = LinearRegression().fit(x_train, y_train)

model.intercept_ // to get the model intercept

model.coef_ // to get the model's coefficients in order in which they are listed in the training data

- equation for the fitted regression line can be written as:

$$y=3800.68+80.35\times temperature-4665.74\times humidity-196.22\times windspeed$$

MODEL EVALUATION

- One way to evaluate a linear regression model is by calculating coefficient of determination or R^2. closer the metric is to 1, the better the model
- Call the score method of the model and pass it to x_test and y_test

model.score(x_test, y_test) // e.g. value is 0.98, means model is able to explain 98% of variability in response values

- Another way of evaluation, see how accurate it is by comparing predicted values against actual values

y_pred = model.predict(x_test) //first predict using the model

from sklearn.metrics import mean_absolute_error

mean_absolute_error(y_test, y_pred) //check predicted against test values, e.g. value is 194, means values can be off the mark by an average of plus or minus 194 units

- Mean Absolute Error (MAE) is a useful metric for evaluating the performance of a regression model.