Appendix A: AdvancedAnalytics API

AdvancedAnalytics contains classes that make programming python analytics faster and less error prone. This API summarizes the classes and their parameters, attributes and methods. AdvancedAnalytics is only designed for Python 3.

For Anaconda users, the condal install command is easier to use. It manages package dependencies better. Both installs check for dependent packages, such as NLTK. However, with the pip install, conflicts can arise in versions. For a detailed discuss of dependencies see https://pypi.org/project/AdvancedAnalytics/

- **Anaconda:** conda install -c dr.jones AdvancedAnalytics
- **Python:** pip install AdvancedAnalytics

The following table summarizes the organization and contents of this appendix.

AdvancedAnalytics Structure

Super Class	Description	Classes
Background	Why AdvancedAnalytics	None
ReplaceImputeEncode	Custom Data Type	DT ReplaceImputeEncode
Regression	Linear & Logistic Regression	linreg logreg stepwise
Tree	Decision Trees	tree_regressor tree_classifier
Forest	Random Forests	forest_regressor forest_classifier
NeuralNetwork	Neural Networks	nn_regressor nn_classifier
Text	Text Analytics - NLP	text_analysis text_plot sentiment_analysis
Internet	Web Scraping	scrape
Metrics	Quality of Fit Metrics	metrics

Background

Developing python machine learning applications (MLA) generally involves the following steps.

1. Data Preparation and Preprocessing
2. Model Development
3. Display and Evaluation of the Model

In terms of MLA coding, the time spent in the first and last steps is much higher than the second. AdvancedAnalytics provides easy-to-use tools for the first and last steps in the MLA development process. These are organized to support popular machine learning packages including:

- Pandas
- Sci-Learn
- StatsModels
- NTLK

This API is intended to provide a summary of these classes and their parameters, attributes and methods. A more detailed description is provided in the book *The Art and Science of Data Analytics*.

Class DT: Advanced Analytics Data Types ⇑

There are many data structures and base data types used in python. Both `pandas` and `numpy` have custom data structures such as data frames, data series, arrays and matrices. DT is another custom data structured used to identify statistical data types. Custom data structures and data types are built from eight python base data types :

- Integer
- Floating Point
- Complex
- String

- Boolean
- List
- Tuple
- Dictionary

In addition to these 8 basic data types, python contains many other objects that are considered customized data types. Class `ReplaceImputeEncode.DT` is a custom data type. Data types were first described by S. S. Stevens in a 1946 article in *Science* titled *On the Theory of Scales of Measurement*. He defined four statistical levels: Nominal, Ordinal, Interval and Ratio. The class `DT` provides a way of classifying data by the first 3 of these measurement levels plus an additional 5.

- `DT.Interval`
- `DT.Binary`
- `DT.Nominal`
- `DT.Ordinal`
- `DT.String`
- `DT.Text`
- `DT.ID`
- `DT.Ignore`

The first three data types correspond to those introduced by Stevens. The fourth data type, `DT.Ordinal` is currently treated as `DT.Nominal`. This is scheduled to be introduced in the next release.

Levels five and six are also both treated as `DT.Text`. The last two are ignored. That is the last two are changed during data preparation, preprocessing.

The following is a code snippet illustrates how `DT` is used in a python data dictionary to provide data types. This is a form of meta-data for the data analysis.

Class DT Snippet

```
from AdvancedAnalytics.ReplaceImputeEncode import DT
# Data map constructed using DT data types.
attribute_map = {
          "gender" : [DT.Binary, ('M', 'F')],
          "cust_no": [DT.ID, ("")],
          "age"    : [DT.Interval, (18, 120)],
          "review" : [DT.Text, ("")] }
```

The *keys* in a data dictionary are attribute names. If the data is in a pandas dataframe, the keys are column names. This snippet has four attributes. The data file may contain other attributes, but if they are not in the dictionary, they are excluded from data preprocessing with `ReplaceImputeEncode`.

The *values* in this data dictionary are *lists*, each containing *2* elements. The first is the statistical data type described using `DT`. The data type is used by `ReplaceImputeEncode` to correctly impute and encode attributes.

The second item in the value list is a *tuple*. The tuples describe accepted values for each attribute. For example, the tuple for `gender` is `('M', 'F')`. Since the type for this attribute is `DT.Binary`, it should only have two values, `M` and `F`. An observation with the value `'m'` would be classified as an outlier.

The tuples for `cust_no` and `review` are empty strings, indicating that for these attributes any data type is acceptable. For further details, see the `ReplaceImputeEncode` class for a detailed description of its interpretation.

Finally `age` is an interval attribute with possible values from 18 to 120.

ATTRIBUTES

`DT.Interval` Represents AdvancedAnalytics interval (numeric) data type. `DT.Interval` variables are either integer or floating point values. `DT.Interval` is converted to the character `"I"`.

DT.Binary Represents AdvancedAnalytics binary data type. `DT.Binary` variables have only two values. These can be represented by python strings, integers or floating point values. `DT.Binary` is converted to the character `"B"`.

DT.Nominal Represents AdvancedAnalytics nominal data type. `DT.Nominal` variables have more than two values. These must be represented by python strings, integers or floating point values, but data types cannot be mixed in the data map. The must be all strings, integers or floating point values. `DT.Nominal` is converted to the character `"N"`.

DT.Ordinal Represents AdvancedAnalytics ordinal data type. `DT.Ordinal` variables have more than two values. These must be represented by python strings, integers or floating point values, but data types cannot be mixed in the data map. The must be all strings, integers or floating point values. Currently `DT.Ordinal` is treated as nominal and converted to the character `"N"`.

DT.String Represents AdvancedAnalytics string data type. `DT.String` variables have string values, and they are not intended for use in text analytics. `DT.String` is converted to the character `"S"`.

DT.Text Represents AdvancedAnalytics data type for text analysis. Unlike `DT.String`, `DT.Text` is intended for use in text analytics. `DT.Text` variables have text values, i.e. long strings. `DT.Text` is converted to the character `"T"`.

DT.ID Represents AdvancedAnalytics data type for identification. `DT.ID` variables are used as labels for data, and are not intended for data analysis. `DT.ID` variables can have strings, integers or floating point values. `DT.ID` is converted to the character `"Z"`.

DT.Ignore Represents AdvancedAnalytics data type that should be ignore. `DT.Ignore` data are not read into memory and do not undergo preprocessing or analysis. `DT.Ignore` variables can have strings, integers or floating point values. `DT.Ignore` is converted to the character `"Z"`.

Class ReplaceImputeEncode: Data Preprocessing ⇑

The primary purpose of this class is to prepare data for analysis, referred to as data preprocessing. `ReplaceImputeEncode` conducts data preprocessing using the following three steps.

- Identifying & Correcting Outliers
- Imputing Missing Values, and
- Encoding Binary & Nominal Attributes, and Scaling Interval Attributes

The general API format for `ReplaceImputeEncode` is:

```
from AdvancedAnalytics.ReplaceImputeEncode import DT, \
          ReplaceImputeEncode

ReplaceImputeEncode(data_map=< data  dictionary>,
          Interval_Scale= None, Binary_Encoding= None,
          Nominal_Encoding= None, no_impute= None,
          drop= False, display= False)
```

PARAMETERS

data_map

The accepted parameter values are either `None`, `"filename"` or a python dictionary. If `data_map=None`, a draft map is created by scanning the data to infer attribute types and valid values. This should be considered a draft. It assumes all data values are correct. If outliers are present, the data map descriptions of valid data will likely need correcting.

If `data_map="filename"`, it is assumed the data map dictionary was previously saved into a python pickle file. This is retrieved and used as the data map.

Each key in the data map dictionary is the pandas column name. The value for each key is a list of two elements. the first is one of the recognized data types defined by class `DT`. These are `DT.Interval`, `DT.Binary`, `DT.Nominal`, `DT.Ordinal`, `DT.String`, `DT.Text`, `DT.ID`, and `DT.Ignore`. The second is a tuple describing all valid data values for the attribute.

For data with the data type `DT.Text, DT.ID` or `DT.Ignore`, the tuple contains an empty string (`""`). For interval attributes, the tuple is two numbers, the lower and upper limits, inclusive, for the attribute. For example, an interval attribute `temp` with lower and upper limits of [0, 212], the dictionary entry would be: `"temp": [DT.Interval, (0, 212)]`

For binary and nominal attributes the tuple describes all recognized values. The data map entry for a binary variable named `credit` might be: `"credit:" [DT.Binary, ("no", "yes")]`

Examples for nominal attributes are: `"income" : [DT.Nominal, (1, 2, 3)]` and `{"income" : [DT.Nominal, ("low', "mid", "high")]}`. The categories can be all numeric or string, but not both. The data map entry `"account": [DT.Nominal, (1, 2, "high")]` is considered invalid.

interval_scale

A string with values `"std"`, `"robust"` or the default `None`. Determines the scaling of all interval attributes in `df`. This might include an interval target, if present. `"std"` scales using the standard deviation and mean to standardize the interval attribute. This is also referred to as z-score scaling: $z = (x - \hat{x})/s$. `"robust"` uses the range and median instead: $z = (x - m)/r$.

binary_encoding

A string with values `"SAS"`, `"one-hot"` or the default `None`. If `binary_encoding` is set to the default, then encoding is not applied, and all binary variables are unchanged. This implies that all binary variables with classes described by two strings, will remain strings.

If encoding is set to `"one-hot"`, all binary attributes are encoded into numeric variables with values 0 and 1. A binary attribute like `"gender" : [DT.Binary, ('M', 'F')]` is encoded into 0 and 1. The encoded *event* is the class that is alphabetically or numerically last. In this case, `'F'` is encoded as 0.

If encoding is set to `"SAS"`, all binary attributes are encoded into numeric variables with values -1 and 1. A binary attribute like `"gender" : [DT.Binary, ('M', 'F')]` is encoded into -1 and 1. The encoded *event* is the class that is alphabetically or numerically last. In this case, `'F'` is encoded as -1.

nominal_encoding

The default `None`, `"SAS"`, or `"one-hot"`. Determines the encoding applied to all nominal attributes.

no_impute

The default `None`, or a list of one or more column names in `df`. Identifies one or more attributes that should not be imputed. In most applications this consists of target attributes since in sci-learn models, targets are not encoded.

drop

`True` or the default `False`. Nominal attributes with `k` levels, are encoded by replacing nominal columns with a collection of `k` new columns. The technique used for generating these columns is controlled by `nominal_encoding`. Regardless of which encoding method is used, the sum of these columns equals 1.0. As a result, not all of them can be modeled with a regression equation that includes an intercept. In linear and logistic regression, it is necessary to drop the last of these columns. One exception occurs when using forward stepwise selection to select the best columns. This method automatically excludes one or more of the encoded columns. Other models, such as decision trees and random forests do not exclude one of the `k` columns.

display

`True` or the default `False`. Controls whether information from the data preprocessing is displayed. This information is useful for confirming the preprocess was accomplished as expected. The information displayed includes the number of observations and the names, number missing and number of outliers for each attribute in the `data_map`.

ATTRIBUTES

col

A list containing the column names for the dataframe returned by `fit_transform`, ordered as they appear in this dataframe.

METHODS

fit_transform(df)

df is a pandas dataframe containing only the targets or input attributes. Attributes that are to be ignore should not be included. `fit_transform` returns a new pandas dataframe with imputed attributes, scaled interval attributes and encoded nominal attributes. Attributes listed in the parameter `no_impute` will not be imputed.

draft_data_map(df, max_n=10, max_s=30, display_map=True, out=None, replace=False)

Scans the data to prepare and display a draft **data_map** for df. If a column is numeric and contains **max_n** or more unique values, it is labeled as DT.Interval. Numeric columns with fewer than **max_n** unique values are labeled as DT.Binary or DT.Nominal, depending upon the number of classes.

Similarly, columns containing string attributes with **max_s** or more unique values are labeled as DT.Text. If a column of strings contains fewer than **max_s** unique values, it is labeled as either DT.Binary or DT.Nominal, depending upon the number of classes.

The displayed data dictionary can be copied and pasted into a python program. However, if out is set to a filename such as out="draft_data_map.pk", the dictionary is stored into a pickle file with the filename designated by out. Later this file can be loaded and assigned to the data_map parameter in ReplaceImputeEncode.

Sometimes the columns designated as DT.Text will need to be changed to either DT.ID or DT.Ignore. Likewise, those designated as DT.Interval may need to be changed to DT.Nominal, and visa versa. Sometimes this can be accomplished by increasing or decreasing k_min and k_max.

display_data_map()

Displays the data dictionary currently paired with ReplaceImputeEncode.

get_data_map()

Returns the data map currently paired with ReplaceImputeEncode. The data map is returned as a python dictionary.

load_data_map(file_name)

Loads the data dictionary from **file_name**. The data map is assumed saved to a pickle file using the out parameter in the draft_data_map method, or the save_data_map method.

save_data_map(data_map, file_name)

Saves the data dictionary data_map to file_name. data_map is saved as a pickle file.

The following example is a small example that uses three AdvancedAnalytics classes: `DT`, `ReplaceImportEncode`, and `logreg`. The data are customer issue reports registered with the National Transportation, Highway and Safety Administration. The target is a binary attribute `crash` which indicates whether the vehicle was involved in an accident. The objective is to identify the attributes that are contributing accidents by modeling `crash` against the other attributes recorded with the complaints.

Code lines 1-6 import that packages need to fit the logistic regression model. This includes classes `DT`, `ReplaceImputeEncode` and the AdvancedAnalytics class for logistic regression. The model is estimated using the sci-learn.

Lines 9-15 is an abbreviated data map described using class `DT`. Lines 17-19 preprocess these data using `ReplaceImputeEncode`. This class imputes missing values and outliers, encodes nominal attributes using one-hot encoding, and returns this in the dataframe `encoded_df`.

The actual logistic regression fit occurs in lines 23-24. Lines 25 and 26 use the `logreg` class from AdvancedAnalytics.Regression, which will be described below.

Notice that only 3 of the 29 code lines directly involve fitting the logistic regression model. The remainders are involved in pre and postprocessing of data and analysis.

ReplaceImputeEncode Snippet for Logistic Regression

```
1  # Required Import
2  import pandas as pd
3  from AdvancedAnalytics.ReplaceImputeEncode import DT
4  from AdvancedAnalytics.ReplaceImputeEncode import
       ReplaceImputeEncode
5  from AdvancedAnalytics.Regression import logreg
6  from sklearn.linear_model import LogisticRegression
7  df = pd.read_excel("BinaryTarget.xlsx")
8  attribute_map = {
9      "temperature": [DT.Interval, (-40, 100)],
10     "crash":       [DT.Binary, ("yes", "no")],
11     "cost":        [DT.Nominal, ("low", "mid", "high")],
```

```
    "report":       [DT.Text, ("")],
    "ID":           [DT.ID, ("")],
    "date":         [DT.Ignore, ("")] }

rie = ReplaceImputeEncode(data_map=attribute_map, display=True,
                    Nominal_Encoding="one-hot", drop=True)
encoded_df = rie.fit_transform(df)
y = encoded_df['crash']
X = encoded_df.drop('crash', axis=1)

lr = LogisticRegression()
lr.fit_transform(X, y)
logreg.display_coef(lr, X.shape[1], y, col=X.columns)
logreg.display_metrics(lr, X, y)
```

Class Regression: Linear Regression

⇑

Class `linreg` supports displaying results from fitting sci-learn Linear Regression models. Currently the functions in this class support the sci-learn function `LinearRegression`. This class consists of a series of methods for displaying results from linear regression models. It does not have an instantiation method.

Linear Regression Snippet

```
from AdvancedAnalytics.Regression import linreg
from sklearn.linear_model import LinearRegression
                          . . .
                          . . .
lr = LinearRegression(C=0.5, tol=1e-8)
lr = lr.fit_transform(X,y)
linreg.display_coef(lr, X, y)
linreg.display_metrics(lr, X, y)
```

LINREG METHODS

display_coef(lr, X, y, col=None)

This function displays the linear regression coefficients returned from `sklearn.linear_model.LinearRegression` labeled by the columns in `X`. `X` is the all numeric $n \times k$ dataframe or matrix sent to `sklearn`, where n is the number of observations and k is the number of columns in `X`.

The parameter `y` is the target, a column or array containing the `n` target values. `col` is a list of string labels for the columns in `X`. If `X` is a pandas dataframe then set `col=X.columns`.

display_metrics(lr, X, y, w=None)

This function displays the fit metrics for the `sklearn` linear regression model `lr`. The parameters `lr, X, y` are described in the `display_coef` method. The parameter `w` is equal to `None` if `lr` is an unweighted regression model. Otherwise `w` is a column or array of length n containing the regression weights. In `sklearn` the weights correspond to the optional parameter `sample_weights` used with `fit_transform(X, y, sample_weights=w)`

return_metrics(lr, X, y, w=None)

This function returns a list with four fit metrics: `[Adj-R2, AIC, AICc, BIC]`. The parameters to this function are identical to those described for `display_metrics`.

display_split_metrics(lr,Xt,yt,Xv,yv,
wt=None, wv=None)

The sci-learn model selection method `model_selection.train_test_split` randomly divides data described by `X` and `y` into two subsets, referred to as training and validation data. Typically the training subset is 60% to 70% of the data, and the test subset is the remaining data. Users have control over the proportion of data allocated to training and test, and also the randomization.

`lr` is the instantiation of the linear regression model, `lr=LinearRegression()`.

For a 70%/30% split, the parameters are obtained from `sklearn.model_selection`.

```
Xt, Xv, yt, yv = train_test_split(X, y, train_size=0.7)
```

The parameters `wt` and `wv` are linear regression weights, if used.

Class Regression: Logistic Regression ⇑

Class `logreg` supports displaying results from fitting sci-learn Logistic Regression models. Currently the functions in this class support the sci-learn function `LogisticRegression`. This class consists of a series of methods for displaying results from logistic regression models. It does not have an instantiation method.

Logistic Regression Snippet

```
from AdvancedAnalytics.Regression import logreg
from sklearn.linear_model import LogisticRegression
                          . . .
                          . . .
lr = LogisticRegression(C=0.5, tol=1e-8)
lr = lr.fit_transform(X,y)
logreg.display_coef(lr, X, y, col=X.columns)
logreg.display_metrics(lr, X, y)
```

Notice that the above snippet for logistic regression closely resembles the snippet for linear regression. Both include a regularization parameter `C` and others to control the iteration process.

Because of this, the outcomes from the sci-learn regression classes is very different from most packages that do not include regularization. In python, classes for linear and logistic regression that do not include regularization are provided by StatsModels.

LOGREG METHODS

display_coef(lr, X, y, col=None)

This function displays the logistic regression coefficients returned from `sklearn.linear_model.LogisticRegression` labeled by the list of column names in `col`, if given.

`lr` is the logistic regression fit `lr=lr.fit(y, X)`, which follows the instantiation `lr = sklearn.linear_model.LogisticRegression()`. See the sci-learn documentation for logistic regression for the details on using `sklearn` logistic regression.

The parameter `nx` is the number of columns in the encoded features matrix `X`. As with linear regression, this is an all numeric dataframe or matrix of n observations and k columns. The parameter `y` is a column or array of n values for the target. `y` is not encoded into a one-hot matrix. It is a column or series containing $n_classes$ different values.

display_confusion(conf_mat)

This function calculates and displays the fit metrics for a binary classification model with a 2×2 confusion matrix: `conf_mat = [[TN, FP], [FN, TP]]`, where `TN` and `TP` are the true negatives and positive, and `FN` and `FP` are the false negatives and positives obtained from the model.

display_metrics(lr, X, y)

This function displays the fit metrics for the `sklearn` logistic regression model `lr`. The parameters `lr, X, y` are described in the `display_coef` method.

display_split_metrics(lr, Xt, yt, Xv, yv, target_names=None)

The sci-learn model selection method `model_selection.train_test_split` randomly divides data described by `X` and `y` into two subsets, referred to as training and validation subsets. Typically the training subset is 60% to 70% of the data, and the test subset is the remaining data. Users have control over the proportion of data allocated to training and test, and also the the randomization.

`lr` is the instantiation of the logistic regression model, `lr=LogisticRegression()`.

For a 70%/30% split, the parameters are obtained from `sklearn.model_selection`.

```
Xt, Xv, yt, yv = train_test_split(X, y, train_size=0.7)
```

If the target is nominal, `target_names` can be used to label the target classes. By default they are labeled `class0, class1, ...`. These generic labels can be replaced with more identifiable labels by setting `target_name` to a list of labels.

Class Regression: Stepwise

Class `stepwise` contains three versions of the stepwise algorithm for features selection: `forward`, `backward`, and `stepwise`. Currently these work for both linear regression and logistic regression where the target is binary.

Stepwise Logistic Regression Snippet

```
from AdvancedAnalytics.Regression import logreg
from AdvancedAnalytics.Regression import stepwise
from sklearn.linear_model import LogisticRegression
. . .
. . .
sw = stepwise(encoded_df, target, reg="logistic", \
              verbose= True)
selected = sw.fit_transform()
X  = encoded_df[selected]
lr = LogisticRegression(C=0.5, tol=1e-8)
lr = lr.fit_transform(X,y)
logreg.display_coef(lr, X, y, col=selected)
logreg.display_metrics(lr, X, y)
```

Notice that the above snippet for Stepwise logistic regression closely resembles the snippet for linear regression. Both include a regularization parameter `C` and others to control the iteration process.

STEPWISE METHODS

stepwise(encoded_df, target, reg="stepwise", method="stepwise", crit_in=0.1, crit_out=0.1, xnames=None, x_force=None, verbose=False, deep=True)

This method instantiates the `stepwise` class with the parameters for the stepwise algorithm.

`encoded_df` is a Pandas dataframe containing the data, including the target. `encoded_df` contains selection candidates, and possibly non-candidate features.

`target` is the name of the target column in `encoded_df`.

`reg` selects the algorithm used for feature selection. By default `reg` is set to the `stepwise` method. Instead, it can be set to `forward` or `backward`.

`method` selects the stepwise method. Recognized options are `stepwise`, `forward`, and `backward`. By default `method` is set to `stepwise`.

`crit_in` is upper limit for the p-value of a candidate feature to be added to the model. The candidate feature with the smallest p-value is added to the model, provided it is not already in the model and its p-value is less than `crit_in`. This is ignored by `method="backward"`. By default `crit_in=0.1`.

`crit_out` is lower limit for the p-value of a candidate feature to remain in the model. Any features in the model with a p-value greater than `crit_out` are removed from the list of model features. This is ignored by `method="forward"`. By default `crit_out=0.1`.

`xnames` is a list of names in `encoded_df` that are candidates for selection. By default `xnames=None` which indicates that all features, other than the target, are candidates for selection.

`x_force` is a list of features in `encoded_df` that must be included in the model. By default this is set to `None` indicating that none of the candidate features in `encoded_df` should be forced into the model.

`verbose` this controls whether the stepwise method should display the selection at each step. By default `verbose=False` which means that the selection process is not displayed.

fit_transform() executes the feature selection algorithm specified in the `stepwise` method, and returns a list of names for the features selected.

Class Tree: Decision Trees

Classes `tree_regressor` and `tree_classifier` display results from sci-learn decision trees with interval and categorical targets, respectively. Currently the methods in these classes support the sci-learn modules `DecisionTree.Regressor` and `DecisionTree.Classifier`.

Since the fit metrics are different, the two classes use different code calculations. However, as far as the structure of their methods are concerned, they are similar. Calls to both `tree_regressor` and `tree_classifier` are made directly to the corresponding method in `tree_regressor` or `tree_classifier`.

Decision Tree Snippet

```
from AdvancedAnalytics.Tree import tree_regressor
from AdvancedAnalytics.Tree import tree_classifier
from sklearn.tree import DecisionTreeClassifier
                          . . .
                          . . .
dt = DecisionTreeClassifier(max_depth=6)
df = df.fit(X,y)
tree_classifier.display_importance(dt, X.columns)
tree_classifier.display_metrics(dt, X, y)
```

These methods are available in **tree_regressor** and**tree_classifier** classes.

METHODS

display_metrics(dt, X, y)

This function displays the fit metrics for the `sklearn` decision tree `dt` after the tree is trained using the `fit_transform(X,y)` method.

dt is the decision tree object for `sklearn.tree.DecisionTreeRegressor` or `sklearn.tree.DecisionTreeClassifier` after training the tree with the method: `dt = dt.fit_transform(X,y)`

X is the decision tree feature matrix used in `df=fit_transform(X,y)`. Sci-Learn accepts either pandas arrays or numpy matrices for `X`.

y is the decision tree target used in `df=fit_transform(X,y)`. Sci-Learn accepts either pandas or numpy series for `y`.

display_importance(dt, col, top="all", plot=False)

This function displays the importance metrics for a sci-learn decision tree or random forest built for an interval or categorical target. It does not return an object, and it accepts two required and two optional parameters.

dt is the sci-learn decision tree object created by the method `fit_transform(X,y)` from the `sklearn.tree.DecisionTreeRegressor` or `sklearn.tree.DecisionTreeClassifier` module.

col is a list of attribute names in order of the column names in the tree inputs `X`. If `X` is a pandas dataframe, `col=X.columns`. `col` is used to label the importance values.

top is optional. By default it is set to `"all"` which indicates that the importance of all features should be reported. Alternatively, you can request to display only the top `m` features by setting `top=m`.

plot is optional. By default it is set to `False` which suppresses plotting the decision tree. Setting `plot=True` displays a `png` bar-chart of the importance values.

display_split_metrics(dt,Xt,yt,Xv,yv,
target_names=None)

The sci-learn model selection method `model_selection.train_test_split` randomly divides data described by `X` and `y` into two subsets, referred to as training and validation subsets. Typically the training subset is 60% to 70% of the data, and the test subset is the remaining data. Users have control over the proportion of data allocated to training and test, and also the randomization.

dt is the state of the DecisionTreeRegressor model after the `fit_transform` method is applied to `Xt` and `yt`. Normally this corresponds to the code `dt=dt.fit_transform(Xt, yt)`. `Xv` and `yv` are not used in fitting `dt`.

Xt and **yt** are the features and target values for the training data.

Xv and **yv** are the features and target values for the validation data.

`Xt, Xv, yt` and `yv` are obtained from from `sklearn.model_selection`. For a 70%/30% split, the call to this method is:

```
Xt, Xv, yt, yv = train_test_split(X, y, train_size=0.7)
```

target_names is only used with nominal targets. For nominal targets, `target_names` is a list of labels for each nominal class, and is used to label the summary metrics returned by the `classification_report` method in the sci-learn package `sklearn.metrics`.

Class Forest: Random Forests

⇑

Classes `forest_regressor` and `forest_classifier` display results from sci-learn random forests with interval and categorical targets, respectively. Currently the methods in these classes support the sci-learn modules `RandomForest.Regressor` and `RandomForest.Classifier`.

Since the fit metrics are different, the two classes use different code calculations. However, as far as the structure of their methods are concerned, they are similar. Calls to both `forest_regressor` and `forest_classifier` are made directly to

the corresponding method in `forest_regressor` or `forest_classifier`.

Random Forest Snippet

```
from AdvancedAnalytics.Forest import forest_regressor
from AdvancedAnalytics.Forest import forest_classifier
from sklearn.ensemble import RandomForestClassifier
                          . . .
                          . . .
rf = RandomForestClassifier(n_estimators=100, n_jobs=6,
                      criterion="gini", max_depth= None,
                      min_samples_split=10)
rf = rf.fit(X,y)
tree_classifier.display_importance(rf, X.columns)
tree_classifier.display_metrics(rf, X, y)
```

The following methods are available in both **forest_regressor** and **forest_classifier** classes.

METHODS

⇑

display_metrics(rf, X, y)

This function displays the fit metrics for the sci-learn random forest `rf` after it is trained using the random forest method `rf=rf.fit(X, y)`.

rf is the random forest object created by the method `rf=rf.fit(X,y)` from either the `sklearn.ensemble.RandomForestClassifier` or `sklearn.tree.RandomForestRegressor` module.

X is the random forest feature matrix used in `rf=rf.fit(X,y)`. Sci-Learn accepts either pandas arrays or numpy matrices for X.

y is the target used in `rf=rf.fit(X,y)`. Sci-Learn accepts either pandas or numpy series for y.

`display_importance(rf, col, top="all", plot=False)`
This function displays the importance metrics for a sci-learn decision tree or random forest trained with either interval or categorical targets. This method is called using two required and two optional parameters. It does not return an object.

`rf` is the random forest object created by the method `rf=rf.fit(X,y)` from either the `sklearn.ensemble.RandomForestClassifier` or `sklearn.tree.RandomForestRegressor` module.

`col` is a list of the column names in the random forest features matrix `X`. If `X` is a pandas dataframe, `col=X.columns`. The names in `col` are used to label the importance values.

`top` is optional. By default it is set to `"all"` which indicates that the importance of all features should be reported. Alternatively, you can request to display only the top `m` features by setting `top=m`.

`plot` is optional. By default it is set to `False` which suppresses plotting the decision tree. Setting `plot=True` displays a `png` bar-chart of the importance values.

display_split_metrics(rf,Xt,yt,Xv,yv, target_names=None)

The sci-learn model selection method `model_selection.train_test_split` randomly divides data described by `X` and `y` into two subsets, referred to as training and validation subsets. Typically the training subset is 60% to 70% of the data, and the test subset is the remaining data. Users have control over the proportion of data allocated to training and test, and also the randomization.

`rf` is the state of the random forest model after the method `rf=rf.fit(Xt,yt)` is called. Notice that `Xv` and `yv` are not used in training `rf`.

`Xt` and **`yt`** are the features and target values for the training data obtained from `Xt,Xv,yt,yv= train_test_split(X, y, train_size=ts)`, where `ts` is the proportion of the data randomly selected for training.

`Xv` and **`yv`** are the features and target values for the validation data obtained from `Xt,Xv,yt,yv= train_test_split(X, y, train_size=ts)`, where `ts` is the proportion of the data randomly selected for training.

`target_names` is only used with classification applications that involve nominal targets. In these applications, `target_names` is a list of labels for the classes of the target. This is used to label the summary metrics returned by the `classification_report` method in the `sklearn.metrics` package.

Class NeuralNetwork: Artificial Neural Networks ⇑

Classes `nn_regressor` and `nn_classifier` display results from sci-learn neural networks with interval and categorical targets, respectively. Currently the methods in these classes support the sci-learn package `sklearn.neural_network`. This package has two modules: `MLPRegressor` and `MLPClassifier` for training and applying neural networks with interval and categorical targets, respectively.

Since the fit metrics are for interval and categorical targets are different, the two classes in AdvancedAnalytics display different metrics. However, as far as the

organization of these classes are concerned, they are similar. Calls to both `nn_regressor` and `nn_classifier` are made directly to the corresponding method in their classes.

Neural Network Snippet

```
from AdvancedAnalytics.NeuralNetwork import
   nn_regressor
from AdvancedAnalytics.NeuralNetwork import
   nn_classifier
from sklearn.neural_network import MLPClassifier
                                . . .
                                . . .
nn = MLPClassifier(hidden_layer_sizes=(3,2))
nn = nn.fit(X,y)
nn_classifier.display_metrics(nn, X, y)
```

The following methods are available in **nn_regressor** and **nn_classifer** classes.

METHODS

⇑

display_metrics(nn, X, y)

This function displays the fit metrics for the sci-learn neural network `nn` after it is trained using the random forest method `rf=rf.fit(X, y)`.

nn is the neural network object created by the method `nn=nn.fit(X,y)` from either the `sklearn.neural_network.MLPClassifier` or `sklearn.neural_network.MLPRegressor` module.

X is the neural network feature matrix used in `nn=nn.fit(X,y)`. Sci-Learn accepts either pandas arrays or numpy matrices for `X`.

y is the target used in `nn=nn.fit(X,y)`. Sci-Learn accepts either pandas or numpy series for `y`.

display_split_metrics(nn,Xt,yt,Xv,yv, target_names=None)

The sci-learn model selection method `model_selection.train_test_split` randomly divides data described by `X` and `y` into two subsets, referred to as training and validation subsets. Typically the training subset is 60% to 70% of the data, and the test subset is the remaining data. Users have control over the proportion of data allocated to training and test, and also the randomization.

nn is the state of the neural network model after the method `nn=nn.fit(Xt,yt)` is called. Notice that `Xv` and `yv` are not used in training `nn`.

Xt and **yt** are the features and target values for the training data obtained from `Xt,Xv,yt,yv= train_test_split(X, y, train_size=ts)`, where `ts` is the proportion of the data randomly selected for training.

Xv and **yv** are the features and target values for the validation data obtained from `Xt,Xv,yt,yv= train_test_split(X, y, train_size=ts)`, where `ts` is the proportion of the data randomly selected for training.

target_names is only used with classification applications that involve nominal targets. In these applications, `target_names` is a list of labels for the classes of the target. This is used to label the summary metrics returned by the `classification_report` method in the `sklearn.metrics` package.

Class Text: text_analysis

⇑

Class `text_analysis` displays and summarizes the analysis of text from the sci-learn and NLTK text analytics packages. Unlike the previous applications, text analytics does not involve supervised learning. It has no specific target. It does not directly involve forecasting or classification.

Text are referred to as unstructured data. Text analytics brings structure to these data by identifying topics discussed and their emotional content, referred to as *sentiment*.

This class contains methods that require `text_analysis` be instantiated as illustrated in the following snippet.

NLP Snippet

```
from AdvancedAnalytics.Text import text_analysis
from sklearn.feature_extraction.text import
   CountVectorizer
from sklearn.decomposition import
   LatentDirichletAllocation
                              . . .
                              . . .
# Instantiate text_analysis without additional stop
   words
ta = text_analysis(synonyms= None, stop_words= None,
                                    pos= True, stem= True)
# Instantiate the counting machine
cv = CountVectorizer(max_df=0.95, min_df=0.05,
            max_features= None, analyzer=ta.analyzer)
# Create the term/document matrix from a column of text
td    = cv.fit_transform(np.array(df['text']))
# Get the terms found in the documents
terms = cv.get_feature_names()
# Instantiate the LDA Decomposition Algorithm
uv    = LatentDirichletAllocation(n_components=9)
# Decompose td into the n by k matrix of topic
   likelihoods
# n=number of documents, k=n_components (topics)
U     = uv.fit_transform(td)
text_analysis.display_topics(uv, terms, n_terms=10,
                                   word_cloud= True))
```

Instantiating `text_analysis` establishes the basic structure for the text analysis. This includes:

- **Synonyms**: a dictionary of different words with the same meaning.
- **Stop Words:** a list of words that are not needed to identify topics.
- **POS:** Parts of Speech. `pos=True` parses the documents or reviews taking into account the part of speech for each word. Words that are used both as a noun and verb, are counted separately.
- **Stem:** `stem=True` converts and tally's all words into their root form.

PARAMETERS

synonyms

Synonyms are words with the same meaning. This parameter is by default `None`. Otherwise it is a python dictionary where the dictionary keys are the word and its value the synonym. For example, the word *Ed* is a synonym for *Edward*. In a synonym dictionary the entry for this pair would be `"Edward" : "Ed"`. In this case, all instances of these two words would totaled and associated with *Edward*. The shorter version would not appear in the term summary.

stop_words

Stop words are automatically excluded from the list of terms tallied. They are considered unimportant for identifying topics or estimating sentiment. There is a short list of stop words used by the `sklearn` method that tallies terms, the `CountVectorizer`. Often instead of using the default setting `stop_words=None`, additional words are added to the stop list after examining the most frequent words. These are often pronouns and articles that convey little information about topics.

In these cases, `stop_words` is set equal to a list of strings, each representing a word or word combination that should be excluded from the term tallies.

pos

POS is an abbreviation for Parts Of Speech. This parameter controls whether parts of speech should used to tally terms. For example the term *sentence* can be a noun or a verb. The noun would be associated with meaningful collection of words. The verb sentenced is used to describe sentencing associated with a criminal trial. If your body of text makes different use of some words based on their parts of speech, then using `pos=True` might produce a better topic list.

stem

Stemming refers to the process of translating every word to its root form. *Running*, for example becomes run. *Ran* also becomes *run* when `stem=True`. By default `stem=False`, which turns off word stemming in the analysis.

ATTRIBUTES

None

METHODS

analyzer(s)

This method accepts a sentence, contained in a single string `s`, and tokenizes it into a one or two dimensional array of tokens. Tokens are individual terms, numbers and punctuation. `analyzer` is used exclusively for the required parameter `analyzer` of the sci-learn method `CountVectorizer`.

The method `CountVectorizer` creates the term/document matrix `TD`, which is a large spase matrix of token scores for each docment. This is required input to the decomposition algorithm in natural language processing **(NLP)**.

`CountVectorizer` uses the settings from the instantiation of `text_analysis` for `synonyms`, `stop_words`, `POS` and `stem` to create `TD`. These settings are passed onto `analyzer(s)`

`analyzer` returns a list of tokens if `POS=False`. Otherwise it returns a two-dimensional matrix where the first column is tokens and the second is parts of speech for each token.

display_topics(uv, terms, n_terms=15, mask=None, word_cloud=False)

This method displays the results of a topic analysis. The top `n_terms` words associated with each topic are displayed, and if requested, word clouds are created and graphed.

uv the decomposition object created by the `uv = fit_transform(td)` method of the sci-learn decomposition algorithms. Currently, sci-learn has four algorithms.

- LDA - Latent Dirichlet Allocation
- SVD - Singular Value Decomposition
- NMF - Non-negative Matrix Factoriation
- KLD - Kullback-Liebler Non-negative Matrix Factorization

The `SVD` algorithm does not rotate the solution. It common practice to rotate this solution using a *Varimax* rotation. In addition, `SVD` is the only algorithm of the four that returns both positive and negative topic weights in the score matrix `U`.

terms a list of all terms in the term document matrix.

n_terms the number of terms displayed to describe each topic By default this is set to 15 terms..

mask is set to either `None`, the default, or the name of a mask file. This is only used when `word_cloud=True`. A mask file is a black and white image. Words displayed in the cloud are only placed inside the black area of the mask. The default uses a rectangular mask.

word_cloud by default this is False, and word clouds are not graphed for each topic. If this is set to True, a word cloud is created for each topic based upon the terms describing the topic.

score_topics(U, display=True)

This method calculates the topic assignments and their scores for each document and returns them in a pandas DataFrame, **df**. Topic assignments are returned as integers in **df['topic']**, one value for each document. The topic assignments describe the topic most strongly associated with each document. These are numbered from zero to $k-1$, where k is the number of topics extracted during topic decomposition.

Topic scores are returned in **df['score']**.

For each document, the topic score is normalized to fall between 0 and 1. Each score can be viewed as an estimated probability that the document contains a discussion of that topic. The higher the score the more likely the document contains a discussion of that topic. The topic assignment for each document corresponds to the topic with the highest score.

In addition to returning these calculations, `score_topics` displays the topic frequency table. This contains both the number of documents and the percentage associated with each topic.

U: is the only one required parameter for this method: `U`. All text decomposition algorithms in sci-learn return `U` from the function `U = fit_transform(td)`, where **td** is the term/document matrix created using the sci-learn module `CountVectorizer`.

`U` is an $n \times k$ matrix of topic scores, where n is the number of documents and k the number of topics. Except for `SVD`, all sci-learn decomposition algorithms normalize the elements of `U`. This makes all scores range between 0 and 1, and the sum of the scores for each document equals 1.0.

display is set to `True` by default. This controls the display of the topic frequency table. Setting it to `False` will suppress the display.

The following simple example of text analytics using `AdvancedAnalytics.Text`. It illustrates a general approach for topic analysis, identifying topics within a collection of product reviews stored one row per review in a text file with the column heading *review*.

This example follows the general order for identifying topics. First `text_analysis` is instantiated to assist in parsing a collection of product reviews. After the term-frequency matrix is decomposed using Latent Dirichlet Allocation, `text_analysis` is used to display the six topics identified in these reviews.

Text Analysis Example

```
import pandas as pd
from AdvancedAnalytics.Text import text_analysis
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation

df = pd.read_excel("text.xslx")
# Parse the text and create the TD matrix
ta = text_analysis()
cv = CountVectorizer(max_df=0.9, min_df=2, analyzer=ta.analyzer)
td    = cv.fit_transform(df['review'])
terms = cv.get_feature_names()

# Decompose the TD matrix
uv = LatentDirichletAllocation(n_components=6, max_iter=10,
                               learning_method='online')
U   = uv.fit_transform(td)

# Display the topic terms and associated word clouds
text_analysis.display_topics(uv, terms, word_cloud=True)

# Display and plot the proportion of reviews for each topic
df_topic = text_analysis.score_topics(U)
```

Class Text: text_plot

⇑

Currently this class mainly makes it easier to build word clouds from small and large bodies of text or reviews. The following three methods are:

1. **term_dic(tf, terms)**
2. **word_cloud_string(s)**
3. **word_cloud_dic(td)**

The first method, `term_dic`, is used for converting term/frequency matrix into a term dictionary. This in turn can be used as input to `word_cloud_dic(td)`, and also support for sentiment analysis.

Like the second method, last method, `word_cloud_dic(td)` is used to create word clouds from a term dictionary.

This class has no instantiation; its methods are used individually for creating and displaying word clouds. The general API format for `text_plot` is:

```
from AdvancedAnalytics.Text import text_plot

text_plot.word_cloud_dic(tdic, mask= None,
          bg_color="maroon", max_words=30,
          size=(400,200), random=12345)

text_plot.word_cloud_string(s, mask= None,
          bg_color="maroon", max_words=30,
          size=(400,200), random=12345,
          stopwords= None)
```

Word clouds are created using method `WordCloud` found in the wordcloud package. This package can be installed using: `conda install wordcloud`.

PARAMETERS

None

ATTRIBUTES

None

METHODS

term_dic(tf, terms, scores=None)

This method accepts a sparse, two-dimensional term/frequency matrix containing the term frequencies, and converts this to a python dictionary. The terms are strings in the first column of `tf`. The second column are frequencies for each term. The terms are the dictionary keys and the dictionary values are either the term frequencies or a weighted version of these frequencies.

`tf` is a sparse, two-dimensional term/frequency matrix. Terms in this matrix are not required to be unique. The frequencies from multiple occurrences of a term are summed for the dictionary value.

`terms` is a list of all terms to include in the term dictionary. This list does not need to be all inclusive. All terms listed in `tf` might not appear in `terms`. This is useful in sentiment analysis in which case this parameter will only contain sentiment words.

`scores` controls whether the frequencies in the dictionary are weighted. If the default `scores=None` is used, the dictionary will contain unweighted frequency for each term. Alternatively, this parameter can be set to a two-dimensional matrix of terms and weights. In this case, the values in the term dictionary will be equal to the product of the term frequency and the score found in this matrix for that term.

⇑

word_cloud_dic(tdic, mask=None, random=12345, bg_color="maroon", max_words=30, size=(400,200))

This method prepares and display a word cloud based upon the terms and weights found in the term/frequency dictionary, `tdic`.

tdic is the term/frequency matrix expressed as a dictionary. The method `term_dic(tf)` is available for converting a term/frequency matrix into a dictionary. The keys in `tdic` are the terms and the values are the total count for that term. It is assumed that any unwanted terms, or stop words, are removed from `tdic` before passing it to this method.

bg_color is the word cloud background color. By default is it `maroon`. Term characters are displayed using light shades of gray.

mask is a black and white image mask. It is rectangular by default. This can be replaced by setting `mask` for an image file with a black mask on a white background. This will be used to pattern the word cloud terms. The term of clouds will follow the mask contours. Using a larger mask is an option, but in that case you might want to use the parameter `size` to accommodate the mask.

max_words controls the maximum number of terms used in constructing the word cloud. If the number of terms in `tdic` is more than this number, then the top `max_words` are drawn from `tdic` to appear in the word cloud. By default `max_words=30`. Using a larger value is an option, but in that case you might want to use the parameter `size` to enlarge the word cloud.

random the package wordcloud randomizes word placement. By default this parameter is set to a fixed value, 12345. Changing this to another value changes word placement.

size by default the size of word clouds is 400 by 200 pixels, `size=(400,200)`. The first number is the horizontal pixels and the second the height. If you use a larger mask, or increase the number of words in the cloud, the cloud can be enlarged using this parameter.

⇑

word_cloud_string(s, mask=None, random=12345, bg_color="maroon", max_words=30, size=(400,200), stopwords=None)

This method prepares and displays a word cloud based upon the terms found in the string `s`.

s is a string containing a review or document. A simple string of only a few terms results in a cloud with only those terms.

bg_color is the word cloud background color. By default is it `maroon`. Term characters are displayed using light shades of gray.

mask is a black and white image mask. It is rectangular by default. This can be replaced by setting `mask` for an image file with a black mask on a white background. This will be used to pattern the word cloud terms. The term of clouds will follow the mask contours. Using a larger mask is an option, but in that case you might want to use the parameter `size` to accommodate the mask.

max_words controls the maximum number of terms used in constructing the word cloud. If the number of terms in `tdic` is more than this number, then the top `max_words` are drawn from `tdic` to appear in the word cloud. By default `max_words=30`. Using a larger value is an option, but in that case you might want to use the parameter `size` to enlarge the word cloud.

random the package wordcloud randomizes word placement. By default this parameter is set to a fixed value, 12345. Changing this to another value changes word placement.

size by default the size of word clouds is 400 by 200 pixels, `size=(400,200)`. The first number is the horizontal pixels and the second the height. If you use a larger mask, or increase the number of words in the cloud, the cloud can be enlarged using this parameter.

⇑

METHOD - continued

stopwords by default `stopwords=None`, and stopwords are not removed from s. In this case, the cloud will include stopwords. As a result, it is recommended this parameter be set to a list of terms that should be excluded from the word cloud. On approach is to produce the cloud with the default setting, and then identify stopwords in the cloud. Using that information set this parameter to a list of those words.
Alternatively, the wordcloud package includes a list of stopwords. They can be imported using: `from wordcloud import STOPWORDS`

With that set `stopwords=STOPWORDS`

Another option is to use the NLTK, Natural Language Tool Kit, package:
`from nltk.corpus import stopwords`

Class Text: sentiment_analysis ⇑

This class can be used for text sentiment analysis. Sentiment analysis attempts to classify documents according to their emotional content. That is, psychologists point out that certain words express how person feels about what they are saying. These emotions are often conveyed using words like *love* or *hate*. Sometimes less emotional words are used to color the communication, words like *good* and *bad*.

This has entered into the practice of text analysis. Document words that carry emotional content identified, weights are assigned according to how strong the emotion, and then tallied for a document. A score of zero would indicate the document is emotionally neutral. A positive score would suggest a document with positive emotional content, and negative scores would be just the opposite.

⇑

This class contains methods that require `sentiment_analysis` be instantiated using the following code.

```
from AdvancedAnalytics.Text import sentiment_analysis

sa = sentiment_analysis(sentiment_dic= None, n_terms=4)
```

PARAMETERS

sentiment_dic

The sentiment dictionary is key to a good sentiment analysis. This is a python dictionary where the keys are sentiment terms and the values represent the weight assigned to each term. One of the first dictionaries is the afinn list. If no sentiment dictionary is provided, then by default, a variation of the afinn list is used. Some analysts would rather use other sentiment lists, and some would create their own.

n_terms

The sentiment analysis in this class will identify individual documents or reviews considered to have the highest and lowest sentiment scores, i.e. the most positive or negative. This parameter is the minimum number of sentiment words required to be considered as a candidate for most positive or negative. By default `n_terms=4`, which means that only documents or review with at least 4 sentiment terms will be listed among the most positive and most negative documents or reviews.

ATTRIBUTES

sentiment_word_dic

The sentiment dictionary used in the analysis. This is a *deep* copy of the dictionary, not a pointer to the memory location of the dictionary.

METHODS

analyzer(s)

This method accepts a sentence, contained in a single string `s`, and returns a one or two dimensional array of tokens. Tokens are individual terms, numbers and punctuation. `analyzer` is used exclusively as the required parameter `analyzer` of the sci-learn method `CountVectorizer`.

The method `CountVectorizer` creates the term/document matrix `TD`, which is a large sparse matrix of token scores for each document. This is required input to the decomposition algorithm in natural language processing **(NLP)**.

`CountVectorizer` uses the settings from the instantiation of `sentiment_analysis` for `synonyms`, `stop_words`, `POS` and `stem` to create `TD`. These settings are passed onto `analyzer(s)`

`analyzer` returns a list of tokens if `POS=False`. Otherwise it returns a two-dimensional matrix where the first column is tokens and the second is parts of speech for each token.

scores(tf, terms)

This method calculates the sentiment for each document. It returns a dataframe `df_scores` with the sentiment for each document or review in `df_scores['sentiment']`, and the number of sentiment words for that document or review in `df_scores['n_words']`

`tf`: the sparse term-frequency matrix created by the `CountVectorizer` method in the sci-learn. After `CountVectorizer` is instantiated as `cv`, `tf` is created by the function `tf=cv.fit_transform(Y)` where `Y` is a series of documents or reviews.

`terms` is a list of terms extracted by `CountVectorizer`. It can be captures using the code: `terms=cv.get_feature_names()`.

Sentiment Analysis Example

```
import pandas as pd
from AdvancedAnalytics.Text import sentiment_analysis
from sklearn.feature_extraction.text import CountVectorizer

df = pd.read_excel("text.xslx")
# Parse the text and create the TD matrix
sa = sentiment_analysis()
cv = CountVectorizer(max_df=1.0, min_df=1, max_feature=None,
                         ngram_range=(1,2), analyzer=sa.analyzer,
                         vocabulary=sa.sentiment_word_dic)
tf    = cv.fit_transform(df['review'])
terms = cv.get_feature_names()

# Calculate document sentiment and return a dataframe
# with each document's sentiment and number of sentiment words
df_sentiment = sa.scores(tf, terms)
```

Class Internet: Web Scraping

The module `Internet` is a collection of methods for capturing data from the internet. Currently this module has two classes:

- **`Class scrape`**: methods for scraping data from the internet.
- **`Class metrics`**: methods for calculating quality metrics.

⇑

The following snippet describes the four methods currently available from class `scrape`.

```
from AdvancedAnalytics.Internet import scrape

scrape.newspaper_stories(search_terms, display= True)
scrape.newsapi_get_urls(search_words, key, urls= None)
scrape.request_pages(df_urls)
scrape.clean_html(web_pages)
```

PARAMETERS None

ATTRIBUTES
None

METHODS

newspaper_stories(search_terms, display=True)

This method from the newspaper3K package searches a list of news agencies for articles that contain one or more of the terms in `search_terms`. A dataframe is returned containing three columns:

- **agency** - the news agency name
- **url** - the url for the downloaded story
- **story** - a string containing a copy of the web page found at that `url`.

Information about newspaper3K is available at:
https://newspaper.readthedocs.io/en/latest/

`search_terms` is list of strings representing the search terms.

`display` controls display of the search process. The default `True` causes the search progress to display. Setting this to `False` turns off the progress display.

METHODS

newsapi_get_urls(search_terms, key, urls=None)

This method is provided by **https://newsapi.org**. It searches a list of over 50 news agencies for articles that contain one or more of the terms in `search_terms`. A dataframe is returned containing the url's for all articles discovered during this search. The articles can be viewed in an internet browser, or they can be downloaded as a group using `scrape.request_pages`.

Use of this service requires a `key`. This is obtained after registering for an account at *https://newsapi.org/account*. Currently limited access is free and generous.

`search_terms` is list of strings representing the search terms.

`key` an access key obtained from newsapi.org. The key is a long alpha-numeric string.

`urls` is a list of over 50 news agency url's. If this is not provided, all English language news agencies listed with newsapi are searched. A list of these is at https://newsapi.org. Limiting the search to a smaller number of agencies, will shorten the search.

request_pages(df_urls)

This method is provided to download web pages from a list of url's. It returns another dataframe containing two features:

- **url** - the address of the articled downloaded
- **text** - a string containing a copy of the web page found at that `url`.

The web pages are returned as text with html markup removed using `scrape.clean_html`.

`df_urls` a dataframe containing one column, a columns of strings contain url's for the download pages.

⇑

clean_html(web_pages)
This method is strips web pages of html markup. Web pages are text files with html markup embedded used by web browsers to properly display the page. Removing this markup makes web pages more readable with a text editor. It is also a first step in analyzing the text in these pages. This method is used by `scrape.request_pages` to remove markup from the requested web pages.

web_pages a string representing one or more web pages.

Class Internet: MLA Quality of Fit Metrics

`Class metrics` currently contains a single method **binary_loss**. Other methods and algorithms for evaluating the goodness or quality of an algorithm or model, are expected to be maintained in `metrics`.

For now, the class is not instantiated. The method `binary_loss` is called directly as illustrated in the following snippet.

Binary Loss Snippet

```
from AdvancedAnalytics.Internet import metrics

loss, conf_mat = metrics.binary_loss(y, y_predict,
                        fn_cost, fp_cost, display= True)
```

Method `metrics.binary_loss` returns two lists.

- **loss = [fn, fp]** where `fn` and `fp` are the false negative and false positive losses, respectively; each totaled over the entire data.
- **conf_mat = [[n_tn, n_fp], [n_fn, n_tp]]** is the 2 by 2 confusion matrix of counts. `n_tn` and `n_tp` and the number of true negatives and positives, respectively. `n_fn` and `n_fp` and the number of false negatives and positives, respectively.

⇑

If δ_{neg} is the indicator function for false negatives then $\delta_{neg}[i] = 1$ for all observations $[i]$ where $y[i] = 1$ and $\hat{y}[i] = 0$; otherwise $\delta_{neg}[i] = 0$. fn, the loss for false negatives is calculated by:

$fn = \sum_{i=0,n-1} \delta_{neg}[i] \times fn_{cost}[i]$, where fn_{cost} is described as `fn_cost` in the dataframe.

Likewise, if δ_{pos} is the indicator function for false positives then $\delta_{pos}[i] = 1$ for all observations $[i]$ where $y[i] = 0$ and $\hat{y}[i] = 1$; otherwise $\delta_{pos}[i] = 0$. fp, the loss for false positives is calculated by:

$fn = \sum_{i=0,n-1} \delta_{neg}[i] \times fn_{cost}[i]$, where fp_{cost} is described as `fp_cost` in the dataframe.

The confusion matrix `conf_mat` is returned as a 2 by 2 array with rows and columns representing the actual and predicted classifications, respectively.

$$\textbf{confusion matrix} = \left[\begin{array}{c|cc} & \hat{y} = 0 & \hat{y} = 1 \\ \hline y = 0 & \textbf{n\_tn} & \textbf{n\_fp} \\ y = 1 & \textbf{n\_fn} & \textbf{n\_tp} \end{array}\right]$$

n_fn is the sum $\sum_{i=0,n-1} \delta_{neg}[i]$, and **n_fp** is equal to $\sum_{i=0,n-1} \delta_{pos}[i]$. **n_tn** and **n_tp** and the number of correct predicted negative and positive classifications, respectively.

PARAMETERS

None

ATTRIBUTES

None

METHODS

binary_loss(y, y_predict, fp_cost, fn_cost, display=True)

This method accepts a series of observations on a binary target `y`, its predicted values `y_predict`, and cost information and returns the estimated losses from false negatives and positives, and the confusion matrix.

`y` is a numpy of the binary target, encoded using 0 and 1.

`y_predict` is a numpy series of predictions for the target, also encoded using 0 and 1.

`fp_cost` is a numpy series of false positive costs. This allows false positive costs to vary among observations.

`fn_cost` is a numpy series of false negative costs. This allows false negative costs to vary among observations.

`display` controls whether this method operates quietly. The default condition `display=True`, displays the loss calculations. Otherwise, if this is set to `False`, the method returns these calculations, but does not display them.