Processing Data to Improve Machine Learning Models Accuracy

Occasionally we build a machine learning model, train it with our training data, and when we get it to predict future values, it yields **poor** results.

This article aims to provide data processing techniques that can be followed to produce a good quality training set.
It is important to note that each domain is different and hence, it is important to attempt different data optimization techniques to increase accuracy of your model.

Choosing the optimal algorithm for your problem is dependent on its features such as speed, forecast accuracy, training time, amount of data required to train, how easy is it to implement and how difficult is to explain it to others. This is because a large task of data scientist is to discuss and explain patterns along with ML algorithms. Hence, it is crucial to choose the algorithm that solves the problem you are trying to resolve. Having said that, a big task is to ensure that the data that you are feeding is suitable for your target problem and that is what I want to concentrate on in this article.

"It is crucial to pre-process data and improve its quality because the quality of our predictive machine learning model is directly correlated with the quality of the data that we feed into our model."

Although we could tweak model parameters or choose a different algorithm all together but a large part of improving accuracy of your predictive model relies on further enhancing quality of your input data. This article is written to demonstrate the steps we can follow to improve quality of our data before it is fed into a predictive machine-learning model.

Processing Data Is A Crucial Part Of Data Science.
Building A Good Quality Training Data Set Is The Most Important Phase Of Predictive Analysis. Occasionally, It Is Also the Most Time Consuming Part

Common Data Quality Issues

Usually following use cases are experienced when preparing data for your machine-learning model:

1. There might be missing or erroneous values in the data set
2. There might be categorical (Textual, Boolean) values in the data set and not all algorithms work well with textual values.
3. Some features might have larger values than others might and are required to be transformed for equal importance.
4. Sometimes data contains a large number of dimensions and the number of dimensions are required to be reduced.

Article Structure

This article highlights a number of techniques to follow to process data.

1. How we fill missing or erroneous values?
2. How we need to handle categorical or textual values?
3. How we need to scale the values?
4. How we need to select features?
5. How we need to reduce dimensions?

Techniques To Improve Data Quality

Use case 1: Filling Missing Values

Let us assume we want to forecast a variable e.g. Company Sales and it is dependent on following two **variables:** Share Price and Total Employees of the company.

Both Share Price and Total Employees contain numerical values.

Let us also assume that the data for Share Price and Total Employees over a range of dates is stored in different csv files.

Scenario: Once we join the two data sets using Python Data Frame Merge () method, we might see empty values, or placeholder strings such as NaN indicating that the number is blank.

Issue: Most of the models are unable to fit and predict the values when we feed them with missing values.

Solution: Pandas data frame provides a number of features to replace missing values.

Step 1: Place data into a pandas data frame

```
data_frame = pd.read_csv(my_data)
```

Step 2: One option is to remove columns/rows with empty values; however, I do not recommend this approach:

```
#remove all missing data
data_frame.dropna()
#remove missing data from specific columns
data_frame.dropna(subset=['TOtal Wealth'])
```

Gathering clean data is a time consuming task and removing columns (features) or rows can end up losing important information from a data set. Better option: Replace missing values by setting a default value to replace NaN, back or front filling data sets, interpolate or extrapolate the values etc.

We can also use a model and train it with the training data set so that it can return appropriate values to fill missing values. One of the appropriate strategy is to interpolate values using sci kit learn Imputer.

As an example, we can do this:

```
#set default value
data_frame = pd.read_csv(my_data)
#back fill
data_frame.replacena(method='bfill')
#front fill
data_frame.replacena(method='bfill')
#interpolate by placing mean values using sci-kit learn
from sklearn.preprocessing import Imputer
imr = Imputer(missing_values='NaN', strategy='mean', axis=0)
imr = imr.fit(df)
imputed_data = imr.transform(data_frame.values)
```

Once we have replaced missing values, now we need to see if we have any categorical values in our data set.

Use case 2: Handling Categorical Values

Let's assume we want to forecast a variable e.g. Number of Tweets and it is dependent on following two varaibles

Variables: Most Active Current News Type and Number of Active Users.

In this instance, Most Active Current News Type is a categorical feature. It can contain textual data such "Fashion", "Economical" etc. Additionally, Number of Active Users contains numerical fields.

Scenario: Before we feed the data set into our machine-learning model, we need to transform categorical values into numerical values because many models do not work with textual values.

Solution: There are a number of strategies to handle categorical features:

1. Create a dictionary to map categorical values to numerical values

A dictionary is a data storage structure. It contains a list of key-value paired elements. It enables a key to be mapped to a value.

```
map = {'Fashion': 1, 'Economical':2}
#this will map categorical to numerical values
target_feature = 'Most Active Current News Type'
data_frame[target_feature] = data_frame[target_feature].map(map)
```

This strategy works well for ordinal values too. Ordinal values are those textual values that can be ordered such as Clothes Size (Small, Medium, Large etc).

2. Another strategy is to use encoders to assign a unique numerical value to each textual value. This strategy works better for variable with a large number of distinct values (>30) such as for managing Organisational Job Hierarchy.

We could use manual or sci-kit encoders.

2.1 Manual Encoders

```
import numpy as np
target_feature = 'Most Active Current News Type'
#get unique values
unique = np.unique(data_frame[target_feature])
map = {textual_value:index for index,textual_value in enumerate(map)}
#apply map
#this will map categorical to numerical values
data_frame[target_feature] = data_frame[target_feature].map(map)
```

2.2 Sci Kit Learn Encoders

```
from sklearn.preprocessing import LabelEncoder
target_feature = 'Most Active Current News Type'
#use encoder and transform
encoder = LabelEncoder()
encoded_values =
encoder.fit_transform(data_frame[target_feature].values)
data_frame[target_feature] = pd.Series(encoded_values,
index=data_frame.index)
#to inverse, use inverse method
decoded = encoder.inverse_transform(data_frame[target_feature].values)
data_frame[target_feature] = pd.Series(decoded, index=data_frame.index)
```

One more step which is often missed out

I have often seen this scenario: After the textual values are encoded to numerical values, we will see some values which will be greater than the other values. Higher values imply they have higher importance. This can lead to our models treating features differently. As an instance, Fashion news type might get a value of 1 and Economical news type might get a value of 10. This makes the machine learning model assume that Economical news type has more importance than Fashion news type.

Solution: We can solve this by using One-Hot Encoding

One Hot Encoding

To prevent some categorical values getting higher importance than the others, we could use the one hot encoding technique before we feed encoded data into our machine learning model.

One hot encoding technique essentially creates a replica (dummy) feature for each distinct value in our target categorical feature. Once the dummy values are created, a Boolean (0 or 1) is populated to indicate whether the value is true or false for the feature. Therefore, we end up get a wide sparse matrix, which has 0/1 values populated.

As an instance, if your feature has values “A”, “B” and “C” then three new features (columns) will be created: Feature A, Feature B and Feature C. If first row’s feature value was a then for feature A, you will see 1 and for feature B and C, it will be 0 and so on.

Solution:

We can use Pandas get_dummies() method that only converts categorical values to integers.

```
data_frame = pd.get_dummies(data_frame)
```

Additionally, we could use sklearn.preprocessing.OneHotEncoder
Tip: Always One Hot Encode After Encoding Textual Values To Prevent Ordering

Use case 3: Scaling Features

Now all missing values are populated and categorical values have been transformed to numerical values. Usually, when we have multiple features in our data set, we need to ensure that the values of the data set are scaled properly.

Range of values in a feature should reflect their importance.
Higher values imply higher importance

Scenario: Let’s assume we want to measure closing stock price. We want to use a simple best fit-line regression model that uses GBP to EUR exchange rate and number of employees of a company to predict a stock’s price.

Therefore, we gather data set that contains GBP to EUR exchange rate and number of employees of the company over time.

Exchange rates will range from 0 to 1 whereas number of employees will be far bigger values, and could be in 1000s.
Subsequently, the model will consider number of employees to have higher precedence than exchange rates.

There are two ways to scale the features:

Normalisation: Ensuring all values range between 0 and 1. It can be done by implementing following routine:

```
Normalised Value = (Value - Feature Min)/(Feature Max - Feature Min)
```

sklearn.preprocessing.MinMaxScaler can be used to perform normalisation.

Standardisation: Ensuring values in a feature follows normal distribution whereby mean of the values is 0 and standard deviation is 1.

```
Standarderised Value = (Value - Feature Mean)/Feature Standard Deviation
```

sklearn.preprocessing.StandardScaler can be used to perform standarisation.

Standardisation technique is a superior than normalisation technique in most scenarios because it maintains outliers and transforms data into normal distribution. Normal distribution allows models to predict values easily and the weights can also be easily determined which again helps predictive models.
Key: Train Scalers On Training Set Only. Do Not Use All Of The Data.
When we are training our models even when we are training imputers or scalars, always use training set to the train models. Leave test or validation set for testing only.

Use case 4: Remove Existing Features

Let's assume you train your machine learning model on a training set and you are using a measure, such as Adjusted R Squared to assess quality of your machine learning model. Your model's Adjusted R Squared is 90%+ implying that your model can predict 90% of the values accurately.

Scenario: When you feed your test data in to your model, you experience a very low adjusted R squared score implying that the model is not accurate and is over-fitting the training data.

This is a classic case of over-fitting
Some features are just not as important as we first conclude from the training set. It can end up over-fitting our machine-learning model.

Solutions:

There are several methods to prevent over-fitting such as by adding more data and/or eliminating features. I have outlined some solutions in my article:

1. We can remove features that have strong correlation with each other. You can use correlation matrix to determine correlation between all independent variables.
2. We could also use scatter mix plot to determine how all variables are linked to each other.
3. We can use **Random Forest Classifier** which can give us importance of each feature:

```
my_importance_model = RandomForestClassifier(n_estimators=10000,
random_state=0, n_jobs=-1)
my_importance_model.fit(independent_variables, dependent_variables)
print(my_importance_model.feature_importances_)
```

Use case 5: Creating New Features From Existing Features

Occasionally we want to create a new feature out of one or more of the features. Sometimes we can also create a new feature out of the dependent variable, the variable which we we want to predict.

As an example, in time series predictive analysis, we can extract trend and seasonality out from the data and then feed Trend and Seasonality as separate features to forecast our target variable.

Time series analysis is a complex topic and is covered in detail here, here and here.

Use case 6: Reducing Dimensions

Scenario: Occasionally we want to reduce the number of dimensions. An example is when we want to scrape websites and convert textual data into vectors by using word to vector encoding algorithms. We can end up getting a sparse matrix.

Issue: Sparse matrix can slow down the algorithms.

Solution: Decompose the matrix but ensure valuable information is not lost.We can use Principal component analysis (PCA), Linear Discriminant Analysis (LDA) or Kernel principal component analysis to reduce the dimensions.