eYSIP-2018

Coding Tutorial: Machine Learning

Authors:
Swapnil Masurekar
Abhishek Sharma

Duration of Internship: 21/05/2018 – 06/07/2018

2018, e-Yantra Publication

0.1 Overview

0.2 Data-Preprocessing

Machine learning algorithms learn from data. It is critical that you feed them the right data for the problem you want to solve. Even if you have good data, you need to make sure that it is in a useful scale, format and even that meaningful features are included.
In this post you will learn how to prepare data for a machine learning algorithm. This is a big topic and you will cover the essentials.
The more disciplined you are in your handling of data, the more consistent and better results you are like likely to achieve. The process for getting data ready for a machine learning algorithm can be summarized in three steps:

- **Step 1: Select Data:** This step is concerned with selecting the subset of all available data that you will be working with. There is always a strong desire for including all data that is available, that the maxim more is better will hold. This may or may not be true.
- **Step 2: Pre-process Data:** After you have selected the data, you need to consider how you are going to use the data. This preprocessing step is about getting the selected data into a form that you can work.
- **Step 3: Transform Data:** The final step is to transform the process data. The specific algorithm you are working with and the knowledge of the problem domain will influence this step and you will very likely have to revisit different transformations of your preprocessed data as you work on your problem.

0.2.1 Importing the Dataset:

All the datasets for experimentations can be found here - Datasets
Before Preprocessing dataset needs to be imported, in order to import a .csv file pandas library is used. The code is as follows:
Python code:

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Data.csv')
```

```
##print(dataset['Country'][1])
X = dataset.iloc[:, :-1].values
#Depends on independent variable distribution in dataframe
y = dataset.iloc[:, 3].values
#Depends on dependent variable distribution in dataframe
```

0.2.2 Encoding Categorical Data:

Categorical data cannot be fitting to a machine learning model. It is required to encode such data in form of labels.
For example: If a dataset contains a country column as an independent variable, let the contents be, "India","France","Germany", etc. These country names must be encoded in label form such as "India":1,"France":2,"Germany":3, etc. Now labels "1, 2, 3 ...." represents these countries and can be fitting into a machine learning model. But, some models doesn't support label encoded values, as each country would get different weight age depending on the value of the label, hence these labels need to be converted to sparse matrix. Eg: If there are only 3 countries then after label encoding the sparse matrix is represented as: 1-001, 2-010, 3-100. The code snippet of this is as shown below:
Python code:

```
# Encoding categorical data
# Encoding the Independent Variable
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_X = LabelEncoder()
X[:, 0] = labelencoder_X.fit_transform(X[:, 0])
onehotencoder = OneHotEncoder(categorical_features = [0])
X = onehotencoder.fit_transform(X).toarray()
# Encoding the Dependent Variable
labelencoder_y = LabelEncoder()
y = labelencoder_y.fit_transform(y)
```

0.2.3 Missing Data:

If any data is missing in the dataset, it can be determined by using various imputing strategies, one of them is mean. The code for computing mean is as shown, more imputing strategies can be experimented by changing 'strategy' parameter by "mean","median","most_frequent".
Python code:

```
# Taking care of missing data
from sklearn.preprocessing import Imputer
imputer = Imputer(missing_values = 'NaN', strategy = 'mean',
    axis = 0)
```

```
imputer = imputer.fit(X[:, 1:3])
X[:, 1:3] = imputer.transform(X[:, 1:3])
```

0.3 Regression:

Regression is basically a statistical approach to find the relationship between variables. In machine learning, this is used to predict the outcome of an event based on the relationship between variables obtained from the data-set.
An example where regression is applicable:
Given housing price data of a city X which contains information like year house was built, lot size, of bedrooms etc about house also known as independent variables and there is a price associated with each house also known as dependent variable (because its value depends on the independent variable).Now the task is to predict price of a new house given its data. Problems like these are solved usually using Regression.
Some basic details about it:
Regression analysis helps one understand how the typical value of the dependent variable changes when any one of the independent variables is varied. It is widely used for prediction and forecasting (weather forecasting).

0.3.1 Simple Linear Regression:

Linear regression is one of the regression technique where the model can be used to predict the valued outcome depending on the values of independent variable. The model is basically a straight line fitted as per the data points available. Simple Linear Regression is used in case of **single dependent and independent variable**.The model equation is represneted as,

$$y = b_0 + b_1 x_1$$

To learn mathematics of linear regression refer to this link.
Python code:

```
# Simple Linear Regression

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Salary_Data.csv')
X = dataset.iloc[:, :-1].values
```

```
y = dataset.iloc[:, 1].values

# Splitting the dataset into the Training set and Test set
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size = 1/3, random_state = 0)

# Feature Scaling
"""from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
sc_y = StandardScaler()
y_train = sc_y.fit_transform(y_train)"""

# Fitting Simple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)#press ctrl+i to learn about the
    method
# In above statements we have as such got
# the equation of the line of Linear Regression
# which can be used to predict the values of salary
# in the test set, which we do further.


# Predicting the Test set results
y_pred = regressor.predict(X_test)
#we here predict the values of the dependent variable by
# passing independent variable as the parameter in the method
    predict of the
# LinearRegression() object regressor trained earlier.


# Visualising the Training set results
plt.scatter(X_train, y_train, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')
plt.title('Salary vs Experience (Training set)')
plt.xlabel('Years of Experience')
plt.ylabel
('Salary')
plt.show()

# Visualising the Test set results
plt.scatter(X_test, y_test, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')
#if we write X_test and y_pred
# here, then we will get the same linear regression line, hence
    no need.
```

```
plt.title('Salary vs Experience (Test set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```

0.3.2 Multiple Linear Regression:

Unlike simple linear regression, in multiple linear regression there are one dependent and multiple independent variables. For eg, the model equation can be represented as,

$$y = b_0 + b_1x_1 + b_2x_2... + b_nx_n$$

The same above code can be used for multiple linear regression as *sklearn.linear_model.LinearRegre* module supports multiple independent variables. The dataset which you can use for this problem is *"50_Startups.csv"*

0.3.3 Polynomial Regression:

The polynomial regression is used when there is no linear relationship between dependent and independent variable. The example of model equation can be represented as,

$$y = b_0 + b_1x_1 + b_2x_1^2... + b_nx_1^n$$

In the same code of simple linear regression instead of *"# Fitting Simple Linear Regression to the Training set"* substitute this code snippet depending on the degree of your choice,

Python code:

```
# Fitting Polynomial Regression to the dataset
from sklearn.preprocessing import PolynomialFeatures
poly_reg = PolynomialFeatures(degree = 4)
X_poly = poly_reg.fit_transform(X)
poly_reg.fit(X_poly, y)
lin_reg_2 = LinearRegression()
lin_reg_2.fit(X_poly, y)
```

0.3.4 Support vector regression:

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which

categorizes new examples. In two dimentional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.
For understanding support vector machine refer: SVM For SVR use this code snippet for fitting the Dataset. Use dataset *"Position_Salaries.csv"*
Python code:

```
# Fitting SVR to the dataset
from sklearn.svm import SVR
regressor = SVR(kernel = 'rbf')
'''
kernel [string, optional (default= rbf )]
Specifies the kernel type to be used in the algorithm.
It must be one of linear , poly , rbf ,
    sigmoid ,
 precomputed or a callable.
If none is given, rbf will be used.
If a callable is given it is used to precompute the kernel
   matrix.
'''
regressor.fit(X, y)

# Predicting a new result
y_pred = regressor.predict(6.5)
y_pred = sc_y.inverse_transform(y_pred)
```

0.3.5 Decision Tree Regression:

For understanding Decision tree algorithms refer: Decision tree algorithms.Use dataset *"Position_Salaries.csv"*.
Python code:

```
# Fitting Decision Tree Regression to the dataset
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor(random_state = 0)
regressor.fit(X, y)

# Predicting a new result
y_pred = regressor.predict(6.5) # example
```

0.3.6 Random Forest Regression:

Random Forest is capable of performing both regression and classification tasks. It also treats missing values, outlier values and other essential steps required for data exploration. Use dataset *"Position_Salaries.csv"*. Refer this code snippet.
Python code:

```
# Fitting Random Forest Regression to the dataset
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(n_estimators = 10,
    random_state = 0)
regressor.fit(X, y)

# Predicting a new result
y_pred = regressor.predict(6.5)
```

0.4 Classification:

Classification predictive modeling is the task of approximating a mapping function (f) from input variables (X) to discrete output variables (y).The output variables are often called labels or categories. The mapping function predicts the class or category for a given observation.For example, an email of text can be classified as belonging to one of two classes: spam and not spam in case of Natural Language processing.

- A classification problem requires that examples be classified into one of two or more classes.
- A classification can have real-valued or discrete input variables.
- A problem with two classes is often called a two-class or binary classification problem.
- A problem with more than two classes is often called a multi-class classification problem.
- A problem where an example is assigned multiple classes is called a multi-label classification problem.

It is common for classification models to predict a continuous value as the probability of a given example belonging to each output class. The probabilities can be interpreted as the likelihood or confidence of a given example belonging to each class. A predicted probability can be converted into a class value by selecting the class label that has the highest probability. Some of the classification algorithms are:

0.4.1 Logistic Regression:

For theory and mathematics refer *"ML_Tutorial.pdf"*. Use *"Social_Network_Ads.csv"* dataset. The code for logistic regression is as follows:
Python code:

```
# Logistic Regression

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size = 0.25, random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Fitting Logistic Regression to the Training set
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1,
stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1,
                     stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
      X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green'
    )))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
```

```
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label =
    j)
plt.title('Logistic Regression (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()

# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1,
stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1,
                     stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
      X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green'
    )))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label =
    j)
plt.title('Logistic Regression (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

0.4.2 K-Nearest Neighbor:

For simple explanation of K-Nearest Neighbor algorithm refer: KNN. Feature scaling is needed in K-NN. Use *"Social_Network_Ads.csv"* dataset.
metric [string or callable, default minkowski] the distance metric to use for the tree. The default metric is minkowski, and with p=2 is equivalent to the standard Euclidean metric. See the documentation of the Distance Metric class for a list of available metrics.

Python code:

```
# Fitting K-NN to the Training set
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5,
        metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)
```

0.4.3 Support vector machine:

For linearly separable data the svm can be used, but if the data is not linearly separrable then we have to create a higher dimensional space to make that data linearly separable and which allows us to use SVM on such data. This is done by various kernel tricks To understand the basics of SVM refer: SVM Tutorial. To understand the Kernel method refer: Kernel Functions
For better understanding watch these videos:
Linear SVMs, primal form
Dual & soft-margin forms
SVM Kernels
Use *"Social_Network_Ads.csv"* dataset.
kernel parameter description:
kernel [string, optional (default=rbf)] Specifies the kernel type to be used in the algorithm. It must be one of linear, poly, rbf, sigmoid, precomputed or a callable. If none is given, rbf will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape (n_samples, n_samples).
Python code:

```
# Fitting SVM to the Training set
from sklearn.svm import SVC
classifier = SVC(kernel = 'rbf', random_state = 0)
classifier.fit(X_train, y_train)
```

0.5 Validation and Model Selection:

Amongst all above models there would be doubt, as which model is best suitable for my regression problem. Here we will learn model selection technique and cross-validation technique for model selection and evaluating model performance.

0.5.1 Confusion Matrix:

First we have to split data into training set and test set(*"X_train, X_test, y_train, y_test"*). Then fit the required model to your training data. Then predict the dependent variable values for all independent test variable values(*"y_pred"*). Then create a confusion matrix for all the values of y_test and "y_pred. This wll give you the count of number of correct and incorrect predictions for each category. Refer this code to create confusion matrix:
Python code:

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

0.5.2 k-Fold Cross Validation:

In confusion matrix technique we performed accuracy check on just one combination of training set and test set. In this technique we compute accuracies for "k" different combinations of training set and test set. This gives the more reliable and descriptive information about the model.This technique also give the standard deviation amongst all the accuracies. Refer this code snippet, there is no need to split data before, *cross_val_score* module splits the data by itself:

Python code:

```
# Applying k-Fold Cross Validation
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = classifier,
        X = X,
        y = y,
        cv = 10) # cv represents value of k
accuracies.mean()
accuracies.std()
```

0.5.3 Grid-Search:(parameter selection technique)

For selecting the most appropriate parameters for a particular model. We perform Grid-search over various model parameters. The example of Grid-Search over SVM is shown below:

Python code:

```
# Applying Grid Search to find the best model and the best
    parameters
from sklearn.model_selection import GridSearchCV
parameters = [{'C': [1, 10, 100, 1000], 'kernel': ['linear']},
              {'C': [1, 10, 100, 1000], 'kernel': ['rbf'],
               'gamma': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8,
    0.9]}]
grid_search = GridSearchCV(estimator = classifier,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 10,
                           n_jobs = -1)
grid_search = grid_search.fit(X_train, y_train)
best_accuracy = grid_search.best_score_
best_parameters = grid_search.best_params_
```

0.6 Clustering:

Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters). It is the method of unsupervised learning as clustering is done on non-labeled data. We will learn one method for clustering i.e.

0.6.1 K-Means Algorithm:

To understand kmeans clustering algorithm refer: Kmeans. In clustering we use method called elbow method using WCSS (within-cluster sums of squares) to find the inertia associated with number of cluster. We do this to select the optimum number of clusters for the given dataset.For more info. refer: Elbow method WCSS

Python code:

```
# K–Means Clustering

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Mall_Customers.csv')
X = dataset.iloc[:, [3, 4]].values
# y = dataset.iloc[:, 3].values

# Splitting the dataset into the Training set and Test set
"""from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size = 0.2, random_state = 0)"""

# Feature Scaling
"""from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
sc_y = StandardScaler()
y_train = sc_y.fit_transform(y_train)"""

# Using the elbow method to find the optimal number of clusters
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
```

```
    kmeans = KMeans(n_clusters = i, init = 'k-means++',
    random_state = 42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()

# Fitting K-Means to the dataset
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state
    = 42)
y_kmeans = kmeans.fit_predict(X)

# Visualising the clusters
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c
    = 'red', label = 'Cluster 1')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c
    = 'blue', label = 'Cluster 2')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c
    = 'green', label = 'Cluster 3')
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c
    = 'cyan', label = 'Cluster 4')
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c
    = 'magenta', label = 'Cluster 5')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.
    cluster_centers_[:, 1], s = 300, c = 'yellow', label = '
    Centroids')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```

0.7 Natural language processing basics:

In NLP the first step is create the bag of words model. This involves two stage, first stage involves cleaning the data and in second stage we create the Bag of words model. The steps involved are:

1. Substituting non-characters with white spaces
2. Manual Tokenization: Split into words by white space
3. Removing Stop-words
4. Text Normalization - Stemming
5. Creating vocabulary of 50 most occurring words in Dataset

6. Convert a collection of text documents to a matrix of token counts
7. Giving weights to words which belong to category

After creating bag of words model, we fit the Gaussian Naive Bayes classifier taking Bag of words model as an independent variable and labeled sentiments as the classification category. Use *Restaurant_Reviews.tsv* Dataset for this experiment.

Python code:

Keep code and dataset in same directory. For sample code click this link: Sample_Codes

0.8 Deep learning basics:

For Installation Instructions of Tensorflow and Keras refer this page:

0.8.1 Artificial Neural Networks:

For theory of ANN refer *"ML_Tutorial.pdf"*
Use *Churn_Modelling.csv* dataset for ANN.
For sample code click this link: Sample_Codes

0.8.2 Convolutional Neural Network:

For theory of CNN refer *"ML_Tutorial.pdf"*
For dataset and sample codes click here : CNN

Useful ML Resources:

- Text Documents
  1. Machine Learning and Algorithms Here you will get the brief idea about ML and Its basic learning algorithm
  2. Basic Terminology related to ML Here you will learn about the basic terminologies and jargons used in ML and Neural Network
  3. Artificial Neural network A Quick introduction to ANN
  4. Convolution Neural network Introduction to Architecture of CNN
- For video tutorial over machine learning refer to following links
  1. Machine learning course by Andrew ng - Click here (enroll for free)
  2. Machine learning A-Z Hands on python in data science - click here(refer tutorials for Python only)
  3. Recurrent Neural Network by Andrew ng - Click here(refer video till LSTM only)
  4. Deep learning by Andrew ng - Click here
  5. Convolution Neural Network by Andrew ng - Click here