

Open in app ↗



Search

Write



★ Member-only story

# Semi-Supervised Learning: Label Propagation for Classification

Beginner friendly tutorial for understanding what label propagation for semi-supervised learning is and learn how to use it for both tabular and text data



Seungjun (Josh) Kim · Follow

Published in Geek Culture · 7 min read · Apr 20





[Free for use photo from Unsplash](#)

## Introduction

What is semi-supervised learning? As the name suggests, semi-supervised learning lies somewhere between supervised and unsupervised learning. Supervised learning in machine learning refers to the classical models we encounter where we have training data each data point labeled (in a classification problem setting) and a model is trained using that very data. Semi-supervised learning is appropriate when the available training data does not have enough data points and thus making use of the predicted labels of the unlabeled data makes sense. In this article, we look at the following:

- How the Label Propagation Algorithm (LPA) works
- Implementation of LPA for tabular data

- Implementation of LPA for text data

## What is LPA and how does it work?

Label Propagation Algorithm (LPA) is an iterative algorithm where we assign labels to unlabeled points by propagating (hence the name label “propagation”) labels through the dataset. It was first proposed in 2002. Take a look at the source paper [here](#). It is a type of transductive learning. The word transductive here means that the algorithm aims to classify the unlabeled input data by exploiting the information derived from labeled data. Another example of a transductive learning algorithm would be the transductive Support vector machine (TSVM).

In a nutshell, the LPA propagates existing labels of data points to other unlabeled data points by making use of two concepts — graphs and random walk. The word graph is sometimes interchangeably used with the term visualization or plot but that is not the graph we are talking about here. Graph here refers to the structure that is comprised of nodes, edges and vertices. The following is the broad steps of how LPA works.

- We create a fully connected graph by linking edges between different nodes. Here, the nodes would be the respective data points.
- The weights for each edge are determined in a way where edges for closer data points have larger weights and edges for points farther away have smaller weights.
- A random walk (this is where the concept of random walk comes in!) from a unlabeled point to a labeled one is performed and the probabilities are calculated for each unlabeled and labeled data point combination. Random walk is performed throughout many iterations. All

paths will be explored and only then, convergence is reached and the iteration will stop.

Something to consider is whether you actually need a “fully” connected graph. Theoretically, it would be ideal to have a fully connected graph so that every random walk could be explored and the best propagation of a label can be selected among them. However, creating a fully connected graph and exploring every possible combination can be computationally expensive. Say you have  $N$  number of nodes. A fully generated graph would have  $N$  choose 2 which amounts to  $N(N-1)/2$ . Therefore, we often find a middle ground by connecting each node with its  $k$  nearest neighbors. In a sense, the way the LPA works is similar to that of the  $K$  Nearest Neighbors algorithm for supervised learning. However, note that multiple unlabeled samples are dealt with synchronously in LPA while KNN deals with only one sample at a time. More importantly, remember that LPA is a semi-supervised learning algorithm and KNN is for supervised learning and so both algorithms, although similar in nature, address different machine learning problems.

For more in-depth understanding of the math behind the scenes, please refer to the following articles that explain the math using examples of random walks in a small graph.

- [Label Propagation Demystified](#)
- [Semi-Supervised Learning — How to Assign Labels with Label Propagation Algorithm](#)
- [How to get away with few Labels: Label Propagation](#)
- [Semi-Supervised Learning using Label Propagation](#)

# Implementation

## Tabular Data

First, import the necessary libraries and packages you need for modeling.

```
import numpy as np
import pandas as pd
from sklearn import datasets
from sklearn.semi_supervised import LabelPropagation

from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
```

We use the breast cancer data from sklearn package's default list of datasets. You load the dataset and randomly assign some data points to be the unlabeled data points (since this dataset does not have any data points with inherently missing labels).

```
# Load in Breast Cancer (Tabular) Data from sklearn's datasets module
cancer_df = datasets.load_breast_cancer()

# To randomly assign some data points to be unlabeled
random = np.random.RandomState(42)

# Randomly assign some data points to be unlabeled
# Denote unlabeled data points with label == -1
rnd_unlabeled_points = random.rand(len(cancer_df.target)) < 0.3
labels = np.copy(cancer_df.target)
labels_orig = np.copy(cancer_df.target) # Keep a copy of the original labels
labels[rnd_unlabeled_points] = -1
```

Next, we fit the LabelPropagation function from sklearn package's semi-supervised learning module.

```
X = cancer_df.data
unlabeled = labels[labels==-1]

# Specify model
model = LabelPropagation(kernel='knn',n_neighbors=5, gamma=30, max_iter=2000)

# Fit Model
model.fit(X, labels)

# Make predictions on originally unlabeled data
predicted_labels = model.predict(X[rnd_unlabeled_points])
true_labels = labels_orig[rnd_unlabeled_points]

# Display classification report and confusion matrix
cm = confusion_matrix(true_labels, predicted_labels, labels=model.classes_)
print(classification_report(true_labels, predicted_labels))
print("Confusion matrix")
print(cm)
```

You will see the following output.

	precision	recall	f1-score	support
0	0.97	0.88	0.92	74
1	0.91	0.98	0.94	95
accuracy			0.93	169
macro avg	0.94	0.93	0.93	169
weighted avg	0.94	0.93	0.93	169

Confusion matrix

```
[[65  9]
 [ 2 93]]
```

Source: From the Author

Let us look at what parameters the LabelPropagation function in sklearn has. This allows us to understand which options could be explored to tweak the model. Refer to the documentation [here](#)!

- kernel: {'knn', 'rbf'} → As I briefly mentioned earlier, creating a fully connected graph can be costly. The radial basis function kernel (abbreviated as rbf) is the default kernel function here. But if you want to compromise the “fully” connected nature of the graph but instead connect each node to only its k nearest neighbors, you can change this kernel parameter to be 'knn' instead.
- gamma: float, default=20
- gamma: float, default=20 → You can perform hyper-parameter tuning using the some parameter tuning function such as the GridSearchCV function in sklearn. A short example would be presented after this section.
- n\_neighbors: int, default=7 → It is a parameter when you use the knn kernel. This obviously needs to be strictly positive since we are talking about the “number” of neighbors.
- max\_iter: int, default=1000 → You can change the maximum number of iterations allowed. You may want to increase this number when you are dealing with a large dataset or the number of neighbors specified is big. This is because it may take more iterations than you think for the algorithm to converge and the convergence will fail if the number of iterations required for convergence exceeds the maximum number of iterations allowed.

The following is an example where you use the GridSearchCV function in the sklearn package to tune the hyper-parameters of the LabelPropagation

function.

```
# Define the parameter grid for the hyperparameters to tune
param_grid = {'kernel': ['rbf', 'knn'],
              'gamma': [0.1, 0.5, 1.0, 5.0, 10, 20, 30, 50, 100],
              'n_neighbors': [4, 5, 6, 7, 8]}

# Specify LPA model
model = LabelPropagation(max_iter=4000)

# Create GridSearchCV object
grid_search = GridSearchCV(estimator=model,
                           param_grid=param_grid, cv=5, scoring='accuracy')

# Fit the GridSearchCV object to training data
grid_search.fit(X, labels)

# Use the best model to make predictions on the unlabeled data points
y_pred = grid_search.best_estimator_.predict(X[rnd_unlabeled_points])

# Calculate the accuracy of the predictions
accuracy = accuracy_score(true_labels, y_pred)
print("Accuracy: {:.2f}%".format(accuracy*100))
```

## Text Data

Nothing changes except for the fact that you would need to vectorize the text data to become some numerical input before you fit the LPA.

We use the [Spam Text Message Classification data](#) from Kaggle. The data has two columns, Message and Category, each of which are storing the text data and the corresponding labels (e.g. spam, ham).

Assuming all the packages we need have already been imported from the previous section, we first read in the data.



```
spam_df = \
pd.read_csv("/kaggle/input/spam-text-message-classification/SPAM text message 20
```

We would want to do some pre-processing of the text data to improve the accuracy of the model (e.g. remove punctuations, remove stop words). There is a neat package (pun not intended) called `neattext` that already contains some of these text cleaning functionalities.

```
# Install the neattext package
!pip install neattext
```

```
from neattext.functions import clean_text
# More from the following website
# https://github.com/Jcharis/neattext

# Apply the clean_text function in neattext to all the messages
spam_df['Message'] = spam_df['Message'].apply(lambda x: clean_text(x))
```

```
# Encode the Category column
spam_df['Category'] = spam_df['Category'].replace({'spam':0, 'ham':1})
```

You do the same thing as you did for tabular data but instead of fitting the LPA right away, you vectorize the input data using some form of vectorizer such as a `CountVectorizer`.

```

np.random.seed(42)

# Specify X and y matrices
X = spam_df[['Message']].values
y_true = spam_df['Category']

## Vectorize the data before fitting LPA
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
# tfidf = TfidfVectorizer()
countvec = CountVectorizer()
X = countvec.fit_transform(X.ravel()).toarray()

# Randomly assign some data points to be the unlabeled data points
n = len(y_true)
mask = np.random.choice(range(n), 9*n//10, replace=False)
y_unlabeled = y_true.copy()
y_unlabeled[mask] = -1 #-1 denotes a missing label

lpa = LabelPropagation(gamma=0.3) # rbf is the default kernel function
lpa.fit(X, y_unlabeled)

# Print out classification report
print(classification_report(y_true[mask], lp.transduction_[mask]))

```

You will get the following output.

	precision	recall	f1-score	support
0	1.00	0.01	0.02	671
1	0.87	1.00	0.93	4343
accuracy			0.87	5014
macro avg	0.93	0.51	0.48	5014
weighted avg	0.89	0.87	0.81	5014

Source: From the Author

This is a wrap! Now you understand what a LPA is and how to implement it.  
Other types of semi-supervised learning algorithms for classification

including Label Spreading and Self-Learning will be introduced in future articles. Stay tuned!

If you found this post helpful, consider supporting me by signing up on medium via the following link : )

[joshnjuny.medium.com](https://joshnjuny.medium.com)

You will have access to so many useful and interesting articles and posts from not only me but also other authors!

## About the Author

*Data Scientist. 1st Year PhD student in Informatics at UC Irvine.*

*Former research area specialist at the Criminal Justice Administrative Records System (CJARS) economics lab at the University of Michigan, working on statistical report generation, automated data quality review, building data pipelines and data standardization & harmonization. Former Data Science Intern at Spotify. Inc. (NYC).*

*He loves sports (tennis and surfing nowadays!), working-out, cooking good Asian food, watching k-dramas and making / performing music and most importantly worshipping Jesus Christ, our Lord. Checkout his [website](#)!*

Semi Supervised Learning

Label Propagation

Sklearn

Machine Learning



## Written by Seungjun (Josh) Kim


355 Followers · Writer for Geek Culture

Follow

Data Scientist; PhD Student in Informatics; Artist (Singing, Percussion); Consider Supporting Me :) <https://joshnjunymedium.com/membership>

### More from Seungjun (Josh) Kim and Geek Culture



 Seungjun (Josh) Kim in Towards Data Science

### Let us Extract some Topics from Text Data—Part IV: BERTopic

Learn more about the family member of BERT for topic modelling

🌟 · 10 min read · Dec 20, 2022



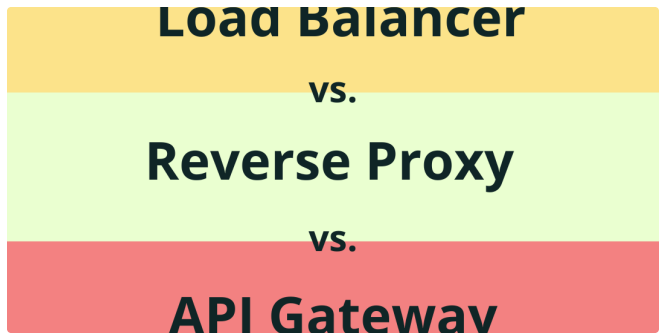
74




3



...



 Arslan Ahmad in Geek Culture

### Load Balancer vs. Reverse Proxy vs. API Gateway

Understanding the Key Components for Efficient, Secure, and Scalable Web...

12 min read · May 17



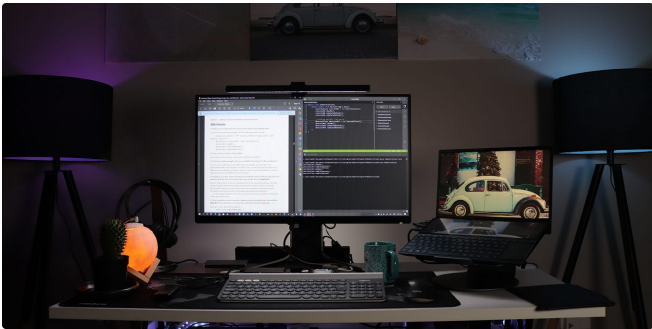
2K



12



...



 Farhan Tanvir in Geek Culture

## 7 Useful Java Libraries You Should Use in Your Next Project

Power up your Java development

🌟 · 5 min read · Aug 30


 223

 4







 Seungjun (Josh) Kim in Geek Culture

## Introduction to the medspaCy, the medical Named Entity...

Take a look at the medspaCy Python package, an open source package effective for...

🌟 · 5 min read · Mar 3

 25



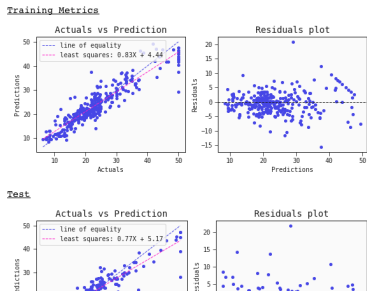




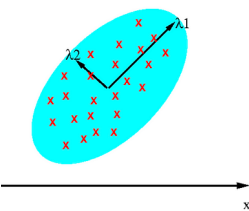
See all from Seungjun (Josh) Kim

See all from Geek Culture

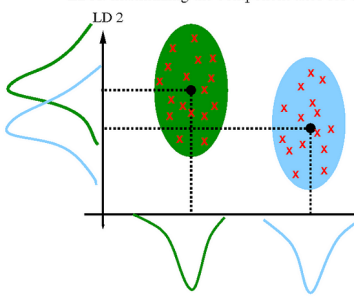
## Recommended from Medium



at axes that maximize the variance



LDA: maximizing the component axes for c





Casper Skern Wilstrup

## Symbolic Regression: a Simple and Friendly Introduction

Symbolic Regression is like a treasure hunt for the perfect mathematical equation to...

3 min read · May 5



82



1



...



Seshu Kumar Vungarala

## PCA vs LDA—No more confusion!

Introduction

3 min read · Apr 30

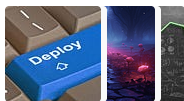


10



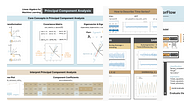
...

### Lists



#### Predictive Modeling w/ Python

20 stories · 482 saves



#### Practical Guides to Machine Learning

10 stories · 554 saves



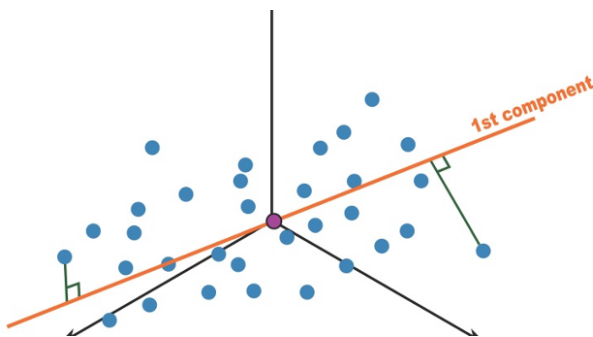
#### Natural Language Processing

698 stories · 309 saves



#### The New Chatbots: ChatGPT, Bard, and Beyond

13 stories · 141 saves



Huda Swati

## Understanding Principal Component Analysis (PCA)

What is PCA?

8 min read · Sep 25



D Sunitha

## Naïve Bayes Algorithm

What is Naïve Bayes ?

7 min read · Jul 3

	Not Long	Sweet	Not Sweet	Yellow	N Yel
0	100	350	150	450	5
	300	150	150	300	
0	100	150	50	50	1
0	500	650	350	800	2

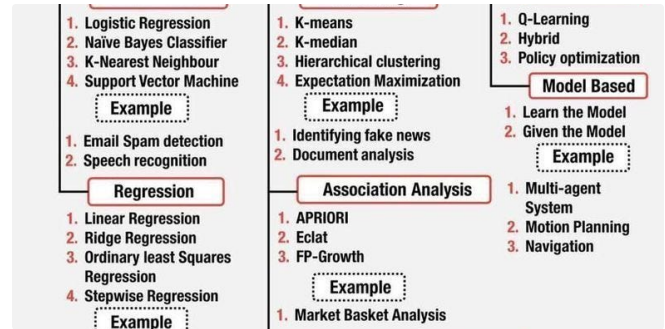


 Saloni Jhalani

## Food Delivery Time Prediction Model

Streamlining Food Delivery: Data Cleaning, Feature Engineering, and Model Building for...

8 min read · Jun 21



 Mohsen Nabil

## Machine Learning Algorithms ..

Hello ... I want to share some insights about the different types of Machine Learning...

2 min read · Jul 16

See more recommendations