Hidden Markov Model: Multinomial Edition

Adrian Campbell

Department of Computer Science and Engineering

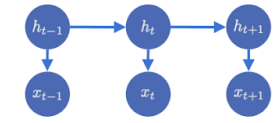
University at Buffalo, Buffalo, NY 14260

aacampbe@buffalo.edu

What is a Hidden Markov Model and its Importance to Machine Learning?

A Hidden Markov Model is a probabilistic model that is utilized to describe or obtain the probabilistic feature of any arbitrary process. In this sort of model, an assumption that is made is there are hidden states (representation of the values of various variables at a particular point) and these states are dependent on the Markov process/chain. What a Hidden Markov Model is trying to do is to predict values of one or more values from an observed event based on the current state of those variables and not from any prior activity. The values of a state are represented as a probability distribution (the probability of different possible values of a variable).

Example of a Hidden Markov Model and the Processes Involved



$p(h_0)$	
$h_0 = 0$	$h_0 = 1$
0.4	0.6

	$p(h_t \mid h_{t-1})$	
	$h_t = 0$	$h_t = 1$
$h_{t-1} = 0$	0.5	0.5
$h_{t-1} = 1$	0.5	0.5

	$p(x_t \mid h_t)$	
	$x_t = 1$	$x_t = 2$
$h_t = 0$	0.4	0.6
$h_t = 1$	0.6	0.4

From the model above there is a hidden variable called x and x has probabilities when it equals 1 and also when it equals 2. In order to get those probabilities what must be know is the values and probabilities of the other variables that are linked to getting the probability of the x value or the state. For example, to find the probability of x = 1 it will be represented by the following calculations

$$p(x_1 = 1) = p(h_0 = 0)p(h_1 = 1|h_0 = 0)p(x_1 = 1|h_1 = 1) + p(h_0 = 1)p(h_1 = 1|h_0 = 1)p(x_1 = 1|h_1 = 1)$$
$$+ p(h_0 = 0)p(h_1 = 0|h_0 = 0)p(x_1 = 1|h_1 = 0) + p(h_0 = 1)(p(h_1 = 0|h_0 = 1)p(x_1 = 1|h_1 = 0)$$

The additions represent the summations of different states that can lead up to the probability of x equaling 1 based on the model above and for each state the probabilities or multiplied due to the fact the variables are dependent on one another to get to either x=1 or x=2.

Multinomial Hidden Markov Model in Python

Modeling Stages of a Conversation

In the demonstration of Multinomial Hidden Markov Model done in Python what will be observed is stages of a conversation in which the topic will be either cat or dog. An assumption that will be made in the conversations is that people are more likely to talk about cats first and if people are talking about cats then people are more likely to talk about a mouse. Also, another assumption that will be made during the conversations is if the topic of conversation is dogs then in the conversation people are more likely to talk about fetching. The probabilities are represented below

```
vocabulary = ["tail", "fetch", "mouse", "food"]

# if the topic is "cat", we are more likely to talk about "mouse"
# if the topic is "dog", we are more likely to talk about "fetch"
emission_probs = np.array([[0.25, 0.1, 0.4, 0.25],
                           [0.2, 0.5, 0.1, 0.2]])
# Also assume it's more likely to stay in a state than transition to the
other
trans_mat = np.array([[0.8, 0.2], [0.2, 0.8]])
```

Now in the next step of the demonstration of Multinomial Hidden Markov Model is the probabilities of the topics mentioned in the conversation when it is repeated a specific amount of times.

```
observations = [["tail", "mouse", "mouse", "food", "mouse"],
        ["food", "mouse", "mouse", "food", "mouse"],
        ["tail", "mouse", "mouse", "tail", "mouse"],
        ["food", "mouse", "food", "food", "tail"],
        ["tail", "fetch", "mouse", "food", "tail"],
        ["tail", "fetch", "fetch", "food", "fetch"],
        ["fetch", "fetch", "fetch", "food", "tail"],
        ["food", "mouse", "food", "food", "tail"],
        ["tail", "mouse", "mouse", "tail", "mouse"],
        ["fetch", "fetch", "fetch", "fetch", "fetch"]]
```

Results

```
Learned emission probs:
[[2.57129200e-01 2.86190571e-02 4.28541642e-01 2.85710101e-01]
 [1.33352852e-01 7.33292496e-01 2.67548571e-05 1.33327897e-01]]
Learned transition matrix:
[[0.71429762 0.28570238]
 [0.50007593 0.49992407]]

New Model
Topics discussed:
['cat', 'dog', 'cat', 'dog', 'cat', 'dog', 'cat', 'dog', 'cat', 'dog', 'cat', 'dog', 'cat', 'dog', 'cat', 'dog', 'cat', 'do
g', 'cat', 'dog', 'cat', 'dog', 'cat', 'dog', 'cat', 'dog', 'cat', 'dog', 'cat', 'dog', 'cat', 'dog', 'cat', 'dog', 'cat',
'dog', 'cat', 'dog', 'cat', 'dog', 'cat', 'dog', 'cat', 'dog', 'cat', 'dog', 'cat', 'dog', 'cat', 'dog']
Learned emission probs:
[[0.32 0.16 0.4  0.12]
 [0.12 0.32 0.2  0.36]]
Learned transition matrix:
[[3.81533706e-14 1.00000000e+00]
 [1.00000000e+00 2.37734700e-13]]
```

As can seen above when the topic of conversations is levelled at that level of frequencies then the probabilities of saying these sub topics with cat being the main topic is listed in the first array under learned emission probabilities and the probabilities of saying these sub topics with dog being the main topic is listened in the second array.

Conclusion

In conclusion, a Hidden Markov Model is a probabilistic model that it utilized to describe the probabilistic feature of any arbitrary process. In this model, states are used and these states are

dependent on a chain of events and these chains of events are dependent on each other to give a probability. Studying Hidden Markov Models are important especially since there are real life usages in studying DNA and reconstructing damaged data collected through sequencing software, weather forecasting, and speech tagging.

References

[1]https://analyticsindiamag.com/a-guide-to-hidden-markov-model-and-its-applications-in-nlp/#:~:text=The%20Hidden%20Markov%20model%20is,a%20set%20of%20probability%20distributions.

[2] https://github.com/cchangyou/hmmlearn/tree/main/examples

[3] https://www.quora.com/How-can-we-apply-the-Markov-chain-analysis-in-real-life

Random Forest Classification

Adrian Campbell
Department of Computer Science and Engineering
University at Buffalo, Buffalo, NY 14260
aacampbe@buffalo.edu

Introduction

A random forest is a Supervised Machine Learning Algorithm that is mainly used in classification and regression. What a random forest does it constructs several decision trees based on different samples and combines them together in order to get a more precise prediction. For a classification problem, a random forest will take the majority vote from those decision trees while for a regression problem a random forest will take the average from those decision trees. Random forests fall into the type of ensemble learning model in which for an ensemble learning weaker models are used in order to strengthen the output and in this case the weaker models would be the decision trees and the decision trees will work together to create a stronger model which is the random forest. Lastly, in a random forest there's a creation portion in which each tree grows within in the algorithm and a prediction portion in which the random forest predicts an output.

Random Forest Algorithm

Creation

In a given amount of N cases in the training set, sample N cases at random but with the replacement of the original data as this will be used as the training set for growing the tree. Next, if there are M input variables then this means a number is specified such that at each node m variables are selected randomly out of the M and the optimal split on this m will be used to split the node.

Prediction

In the first step of the prediction portion, what must be done is take the test features and use the guidelines of each randomly constructed decision tree to predict the result and store the predicted output. After, compute the votes for each predicted output. Last but not last, the high voted predicted outputs will be under consideration for the final prediction from the random forest algorithm. If a decision tree is used as a regression problem then the mean squared error will be used to branch out the data from the nodes and find which branch is the better decision for the forest and the mean squared error formula is

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (fi - yi)^2$$

N is the number of data points, fi is the value returned by the model, and yi is the actual value for data point i. Since the data that will be utilized in this demonstration of a random forest will have numerical data and not be separated into categories, the decision trees will be regression problems instead of categorial problems so there will be no need to know the formulas associated with classification problems.

Random Forest Algorithm on a Diabetes Dataset

Diabetes Dataset

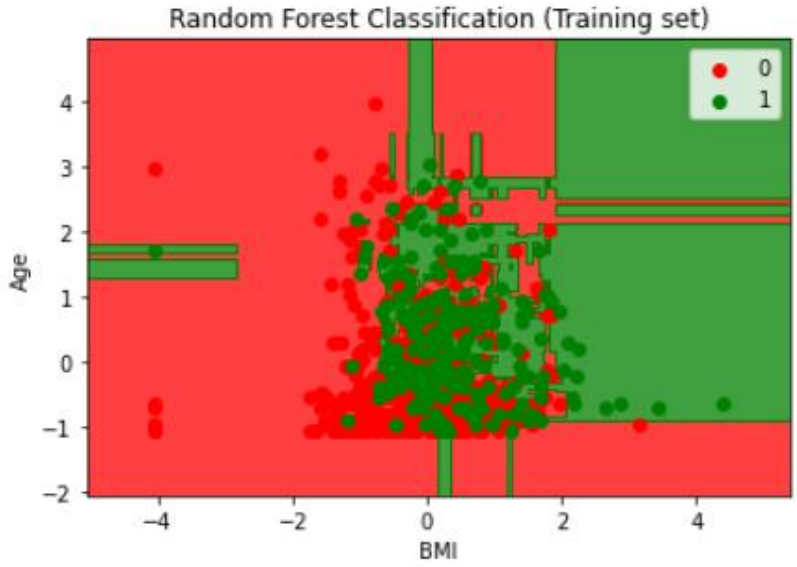
The dataset that will be used is a dataset involving diabetes and in this dataset are several factors that may play a role in a person having diabetes and this would be factors such as glucose, blood pressure, insulin, blood pressure, and age. An outcome of 0 or 1 is in this dataset and what 0 means is the person doesn't have diabetes while 1 means the person has diabetes.

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	\
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	
..	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	
764	2	122	70	27	0	36.8	
765	5	121	72	23	112	26.2	
766	1	126	60	0	0	30.1	
767	1	93	70	31	0	30.4	

	DiabetesPedigreeFunction	Age	Outcome
0	0.627	50	1
1	0.351	31	0
2	0.672	32	1
3	0.167	21	0
4	2.288	33	1
..	...	...	...
763	0.171	63	0
764	0.340	27	0
765	0.245	30	0
766	0.349	47	1
767	0.315	23	0

The data values that will be used in the support vector machine is age and BMI which will be the x-values and outcome will represent the y values and in python importation of SVC libraries and confusion matrix will be used in order to visualize the values from the dataset and also predict the values as well.

Visualization



As can be seen the two features that were used age and BMI are in two axes and there is a map with the two outcomes with no for diabetes represented in red and yes for diabetes represented in green. The decision trees worked in tangent with each other to predict whether or not the patients had diabetes and from this it can be seen the majority of patients did not have diabetes and bmi had more of a correlation with a patient having diabetes than age.

Conclusion

In conclusion, a random forest is a Supervised Machine Learning Algorithm that is mainly used in classification and regression. Within a random forest, several weaker models will be used in order to have a stronger model that will as accurately as possible predict several outcomes and in this case the weaker models were the decision trees and the stronger model was the random forest. A diabetes dataset was used in this demonstration of a random forest in which the predicted outcomes are whether or not a patient has diabetes or not and the values associated with that in the decision trees are BMI and age. Due to the random forest's nature in predicting

the best possible outcomes given several weaker models it is utilized efficiently in many real life circumstances such as fraud detection for bank accounts and credit cards, detecting and predicting the drug sensitivity of a medicine, and identification of a patient's disease by analyzing their medical records.

References

[1] https://github.com/cchangyou/100-Days-Of-ML-Code/blob/master/Code/Day%2034%20Random_Forest.md

[2] https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/

[3] https://medium.com/capital-one-tech/random-forest-algorithm-for-machine-learning-c4b2c8cc9feb

[4] https://www.kaggle.com/general/208443

Application of AdaBoost Classifier

Adrian Campbell

Department of Computer Science and Engineering

University at Buffalo, Buffalo, NY 14260

aacampbe@buffalo.edu

What is an AdaBoost Classifier?

An AdaBoost (Adaptive Boosting) Classifier is a sought-after boosting, a method used in machine learning to reduce errors in predicting data, technique that focuses on merging several weak classifiers to create one strong classifier. Weak classifiers would be ones that execute badly at grouping classes to objects but are better than classifiers that just do random guesses; a classifier that can only classify data at a 60% accuracy would be considered a weak classifier. One example of a weak classifier that will be used in an AdaBoost Classifier in order to create a strong classifier is a decision tree and what a decision tree does is create predictions on how a previous collection of questions were answered and it splits the answers to these questions by nodes and branches. What an AdaBoost Classifier will do to the decision trees is modify versions of the weak classifiers and increase the weights of the misidentified observations so that a weak decision tree could focus on the error of a previous weaker decision tree and improve from that. A step by step process must be done from the AdaBoost Classifier to the decision trees and this will be known as the AdaBoost algorithm.

AdaBoost Algorithm

Weight Initialization

First, what must be done in this algorithm is initialize the weights of all the data points and this is given by $w_i = 1/N, i = 1, 2, \ldots, N$ where N represents the number of data points and all the weights of the data points are weighted equally.

Error Calculation and Weight Updating

Next, what must be done in this step of the algorithm is find the Total Error and the Total Error is the total number of misclassifications of the dataset divided by the dataset size. From that is finding Alpha or how much of an impact a classifier has in the final classification and Alpha is given by $\alpha_m = log((\frac{1-Total\,Error}{Total\,Error}))$. Finally, the weights can be updated and it's given by the following formula $w_i = w_{i-1} * e^{\pm\alpha}$ where the current weight is equal the previous weight times Euler's number to the alpha that was calculated in the previous step.

Output

The AdaBoost algorithm will now output the updated weights from the weak learners and then predictions will be made through a weighted majority vote for all the learners. This will repeat for a certain amount of times that the algorithm specifies, so for example, if the algorithm wants to repeat 200 times it will do all the steps previously 200 times with majority voting each time. Eventually, the algorithm would want to have the correct outputs with as less of an error as possible.

AdaBoost Classifier in Python

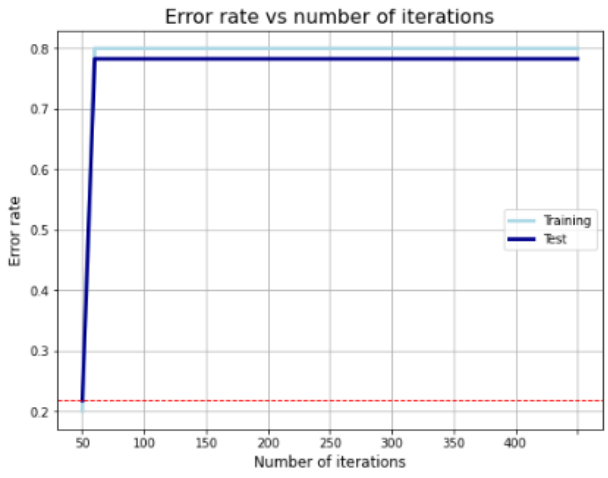
Bank Customer Churn Dataset

The dataset that will be used in the demonstration of AdaBoost Classifier in Python will be a Bank Customer Churn Dataset and what is does is predict whether or not a customer churns (inactivity in transactions in a financial institution) based on several factors such as credit score, age, credit card, and salary. Down below is some of the data associated with predicting whether or not a customer churn or not

	customer_id	credit_score	country	gender	age	tenure	balance
0	15634602	619	France	Female	42	2	0.00
1	15647311	608	Spain	Female	41	1	83807.86
2	15619304	502	France	Female	42	8	159660.80
3	15701354	699	France	Female	39	1	0.00
4	15737888	850	Spain	Female	43	2	125510.82

Two values that will be used in order to predict customer churn will be age and salary.

Visualization



The AdaBoost algorithm was used on the bank customer churn dataset and the results are seen above. From the first 50 iterations there was an error rate of 25% in correctly predicting

whether or not the customer churned or not. However, in more iterations the error become higher so it seemed like the algorithm was more efficient in the only two factors of age and credit score the first 50 times so most likely this is when the weak learners peaked in making the strong learner.

Conclusion

In conclusion, an AdaBoost is a technique used in supervised learning that aims to reduce error in predicting data and is an algorithm that relies on repetition amongst weak learners in order to build a strong learner. Decision trees are examples of weak learners that can be used in the AdaBoost algorithm in order to build a strong learner due to its binary nature of predicting and how its weakness can be controlled by depth and number of nodes. AdaBoost is essential in real life applications such as customer churn predictions and classifying the types of topics customers are talking or calling about.

References

[1] https://blog.paperspace.com/adaboost-optimizer/

[2] https://github.com/cchangyou/adaboost-implementation

[3] https://hackernoon.com/under-the-hood-of-adaboost-8eb499d78eab

Unsupervised Learning Technique: Autoencoder

Adrian Campbell

Department of Computer Science and Engineering

University at Buffalo, Buffalo, NY 14260

aacampbe@buffalo.edu

What are Autoencoders and their Purpose?

Autoencoders are an unsupervised learning technique in which neural networks are used to make out representations in data to have feature detection or classification. The main purpose of an autoencoder is to reduce noise (meaningless data) and focus on essential parts of data in things such as a photo. It does so by using an encoder and decoder and what an encoder will do is it will reduce the complexity of the data by compressing the data inside a neural network and then the decoder will be reconstructed except the complexity will match the complexity that the neural network did instead of the complexity of the data before the neural network involvement. With the utilization of the neural network, the data that will be encoded and decoded using specific functions that pertain to a neural network.

Equations Utilized

Encoding Equation

An activation function will be passed for the encoding part where a variable z represents a latent dimension at is represented by $z=\sigma(Wx+b)$. W represents weights, x represents the inputs and b represents the bias.

Decoding Equation

The decoding equation is represented in a similar fashion but the weights and bias are different and potentially the activation functions are different as well. This decoding equation is

$$x' = \sigma'(W'z + b')$$

Loss Function

The loss function is responsible for backpropagation (a way for the neural network to test errors) in a neural network and the loss function is

$$L(x, x') = ||x - x'||^2 = ||x - \sigma'(W'(\sigma(Wx + b)) + b'||^2$$

Conclusion

In conclusion, autoencoders are an unsupervised learning technique in which neural networks are utilized to denoise data and bring out the important parts of the data. Autoencoders use neural network techniques in order to encode and decode data so that the complexity of the data can be reduced efficiently and a more optimal set of data can be outputted and this can be seen in something like a photo that can contain many unnecessary blurs and appear grainy. Many real-life applications can be done with autoencoders and this includes areas such as facial recognition, feature detection, anomaly detection, and finding out the meaning of words.

References

[1] https://towardsdatascience.com/auto-encoder-what-is-it-and-what-is-it-used-for-part-1- 3e5c6f017726

[2] https://towardsdatascience.com/generating-images-with-autoencoders-77fd3a8dd368

[3] https://rapidminer.com/blog/autoencoders/

[4] https://www.jeremyjordan.me/autoencoders/