Artificial Intelligence Assignment 2: Application of Neural Networks

Roscoe Kerby (krbros002)

Introduction, Background, Objective, Problem Description, Ethics

For this assignment I struggled to decide on a specific Machine Learning (ML) problem, since there is such a wide variety of interesting topics to choose from. I managed to narrow it down to a classification problem as this seemed like an engaging topic that I could easily migrate to many different classification scenarios.

I created several classifiers that included: hate-speech detection, image classification, spam detection and a wine classifier that determined whether a wine was a “good” wine. However, I then remembered that in class I had asked a question relating to the ability of Artificial Intelligences (AIs) to pick up on sarcasm when doing Semantic Analysis and other related tasks. I decided to use this assignment as an opportunity to develop such a ML system that could attempt to classify sentences as either “sarcastic” or “genuine”.

I hoped to create a system that could give a greater than 50% (random chance) for correctness for this binary classification.

No ethical issues have been found.

Potential user community

Researchers/people who are interested in data and statistics might be interested in this kind of system. However, it is quite niche and I do not imagine it being very useful. With that said, the concepts and Machine Learning Problem (MLP) can be utilised in other classification problems.

Potential factors to considered

As pointed out in the lecture, it is sometimes quite difficult to determine sarcasm since it is so unique and subtle. To showcase this subtly it is possible for something as small as quotation marks “” to change the outcome of a specific sentence and a single word can also change its intent. Unfortunately context is very important for determining sarcasm and since this system only uses the headlines of news articles [A] context was not taken into account. Another factor for determining sarcasm is body language and tone and these aspects were obviously not taken into account.

Looking at the dataset, I was surprised by some headlines that I thought appeared sarcastic but in actuality were not and vice versa.

Problem description, Baseline Explanation

This system is a machine learning program that aims to utilise neural networks to determine whether news headlines are sarcastic represented with a 1 or genuine represented by a 0 (absence of sarcasm).

I picked a baseline model of simply determining whether a sentence is sarcastic or not.

Model Design Choices

I originally created an image classifier in PyTorch following their Blitz tutorial [1] as I was hoping that this would help me gain a baseline to understanding PyTorch and help me create more ML programs. However, this tutorial was not enough to help me fully comprehend or understand the deeper aspects of PyTorch, so I decided to look for more classifier examples on GitHub and other such repo sites [2]. I saw that a common library that was being used was Sci Kit [3]. I then found a very useful YouTube Video [4] which helped me get a base structure for my code.

I originally used the 4 classifiers used in [4] on my model as I wanted to do a comparison analysis on accuracy between the different model types. I originally had a Decision Tree Classifier, Random Forest Classifier, Support-Vector Machine as well as the neural network based classifier - Multilayer perceptron (MLP) Classifier. However, after rereading the specifications in the assignment brief I realised that it was very specifically about neural networks so I scrapped the other three classifiers since they took a long time to all run and it was pointless having them since they were not neural network related.

I then wanted to add more neural network based classifiers, however, I found that I struggled to get a decent prediction using these other classifiers which included a Restricted Boltzmann Machine, however I could not get a reading better than 30% accuracy early on so I scrapped that classifier as well since the assignment only required a single neural network classifier.

So the assignment ended with only having a MLP Classifier which is totally fine since I was able to achieve an average of above 75% total accuracy on the test data using this classifier.

Setup

The setup was done using both Jupyter Notebook [5] and Google Colab [6]. For the project handin I shall use the Jupyter version since access to my google drive will not be allowed. The setup was adapted from [7]. I shall also handin a plain Python version.

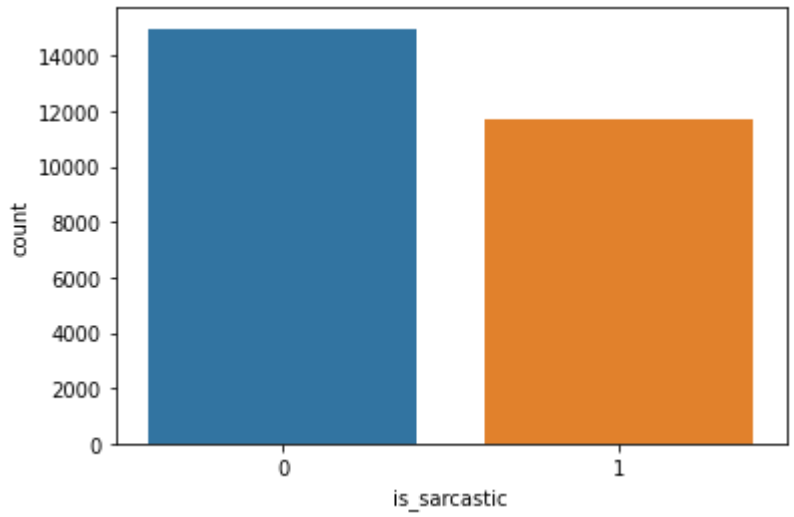
To start with, the preamble, which is the code that deals with importing the external libraries. The preamble also includes the mounting of the google drive and the reading in of the datasets “Sarcasm_Headlines_Dataset.json” and “Sarcasm_Headlines_Dataset_v2.json “. To ascertain that the json files were read in correctly, I check the top of the file using the head() function and check that the number of entries for all the columns is equal using the info() function.

Next, the data normalisation stage, which deals with ensuring that the data is usable. First I check that there are no null values using the isnull().sum() function. Luckily there are no null values in either data set. However, there is a completely useless column called “article link” so I get rid of the column using the drop function. I can confirm that the column is gone using the head function again.

After the normalisation I set the label encoder which basically just sets a 1 value to be equated to “is_sarcastic” and a 0 value to be equated with it not being sarcastic.

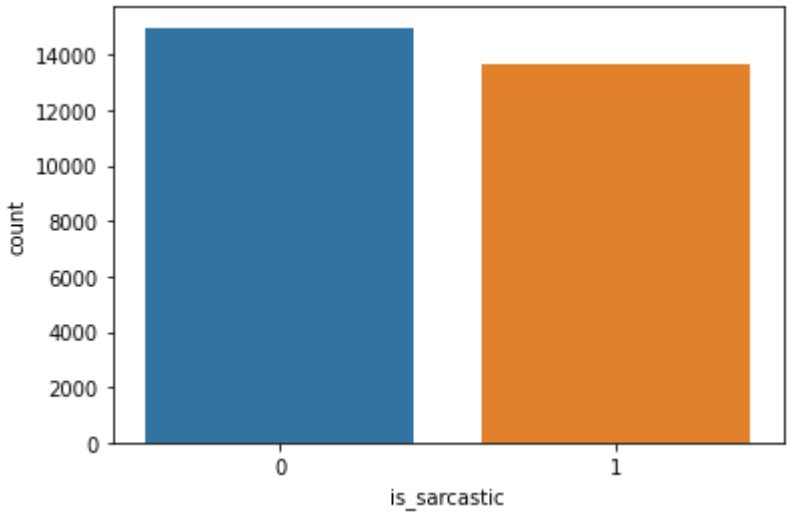
Breakdown of training data (Sarcasm_Headlines_Dataset.json [dataset 1]):

0 14985 (0 = not sarcastic)
1 11724 (1 = is sarcastic)
Name: is_sarcastic



Breakdown of training data (Sarcasm_Headlines_Dataset_v2.json [dataset 2]):

0 14985 (0 = not sarcastic)
1 13634 (1 = is sarcastic)
Name: is_sarcastic



Next I need to normalise the words - like basically stemming them as if I were taking herbs and cutting off the leaves and stems to be left with the good stuff. Basically this is a dataset of English words and I only want the core part of the words from the headlines - I do not want any prefixes or suffixes. So this stemmer cleans the words.

Next is the training of the data. I understand the difference between training, validation and test data, however, I believe that some of this is done on the backend by the libraries and I followed the tutorial by the official SciKit YouTube account so I am pretty confident that they did it correctly. I also followed [7]. So I selected the specific columns and split the data into 20% testing set and 80% training set and I think this was fine. I also tuned a 67% testing vs 33% training however, it did not change the accuracy of the results a significant amount.

I shall discuss the hyperparameters and tuning in the following section.

Instructions to reproduce results

From terminal, cd to folder containing py/ipynb and datasets:
To run in python: ~python3 single_hyperparameter_tuning_of_neural_network_sarcasm_detection_scikit.py

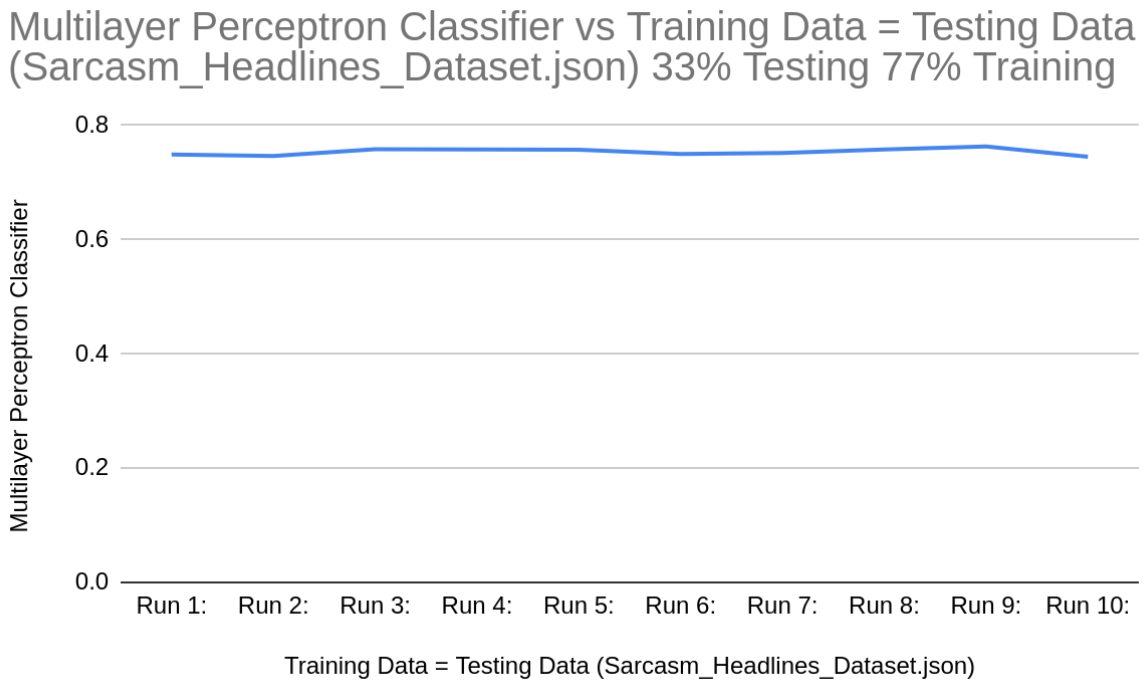
To run in jupyter notebook: ~jupyter notebook

The ones with low precision in the 45-55 range means that the training and testing sets are different files. The rest are done on the same testing and training sets, just using mutually exclusive subsets and that ranged from 69-79 accuracy. Mostly around 75-78%.

Results on both the validation set (for tuning) and the test set (for final evaluation)

Analysis of model performance

Please refer to PDF: Sarcasm Detection Data - Sarcasm Data.pdf



For the MLPC hyperparameters - the default is 200 iterations,however set to 500 due to low hidden layer numbers (only 3 layers). However, I was warned that 500 did not converge so it increased to 10000.

I ended up using 10 iterations as this seemed to get a good accuracy despite the low iteration number! This also increased the speed of my tests so I could run them more quickly, the data in the PDF was run with 10000 iterations each.

Below are examples of accuracies running completely different training set and test set (ie dataset 1 [training] vs dataset 2 [testing])

0.5254141139096892
0.5242195990758454
0.5230250842381131

I cannot use this as it is basically 50%...

I discuss this low accuracy in the conclusion.

After changing the datasets to only be testing and training on a single set and after getting help with hyperparameters from [8], I was able to increase my score to

Below are the best hyperparameters I could achieve

Best hyperparameters for sgd classifier:

```
mlp = MLPClassifier(
   hidden_layer_sizes=(50,),
   max_iter=10,
   alpha=1e-4,
   solver="sgd",
   verbose=10,
   random_state=1,
   learning_rate_init=0.1,
)
Training set score: 0.959096
Test set score: 0.778173
```

And then:

```
mlp2 = MLPClassifier(
       solver= "sgd",
       learning_rate= "constant",
       momentum= 0,
       learning_rate_init= 0.2,
       max_iter=10,
       verbose=10,
)
Training set score: 0.859737
Test set score: 0.784350
```

Best hyperparameters for adam classifier:

```
mlp8 = MLPClassifier(
  solver= "adam",
       #learning_rate= "constant",
       momentum= 0,
       learning_rate_init= 0.01,
       max_iter=10,
       verbose=10,
)
Training set score: 1.000000
Test set score: 0.787158
```

Almost 80% accuracy!

Interestingly enough, the Adam classifier did not do better given unlimited iterations and actually performed worse than the best hyperparameter metric which had 10 iterations.

I tried to efficiently and effectively tune the hyperparameters using a Gridsearch:

```
parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}

from sklearn.model_selection import GridSearchCV

clf = GridSearchCV(mlp, parameter_space, n_jobs=-1, cv=3)
clf.fit(X_train, y_train)
```

Unfortunately Gridsearch did not work so I ended up just doing it manually, so I am sure that the hyperparameters could be improved but I found 78% accuracy to be satisfactory. I believe I was just impatient as I have found that Gridsearch can run for hours and I stopped it each time as I assumed it was just stuck.

I could have used RandomizedSearch but I ran out of energy and time to do this.

Conclusion

The model does not work efficiently on completely different training and testing sets. The amount of data is not big enough to successfully extrapolate in this way. The model is successful with a 75% accuracy on a 67 percent training to 33 percent testing amount as well as 80 percent training and 20 percent testing alike.

I was able to achieve 78% accuracy using both the ‘adam’ and ‘sgd’ classifiers on a Multilayer perceptron (MLP) Neural Network.

Sarcasm was detectable and the single sentence example at the end of my code is very accurate when single headlines from the training dataset are used to test for sarcasm.

Resources:

Training Data:

[A] https://www.kaggle.com/rmisra/news-headlines-dataset-for-sarcasm-detection

References:

[1] https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html

[2] https://github.com/

[3] https://scikit-learn.org/

Title for citation: **API design for machine learning software: experiences from the scikit-learn project**. Authors: Lars Buitinck and Gilles Louppe and Mathieu Blondel and Fabian Pedregosa and Andreas Mueller and Olivier Grisel and Vlad Niculae and Peter Prettenhofer and Alexandre Gramfort and Jaques Grobler and Robert Layton and Jake Vanderplas and Arnaud Joly and Brian Holt and Gaël Varoquaux.

[4] https://www.youtube.com/watch?v=0Lt9w-BxKFQ&ab_channel=Simplilearn

[5] https://jupyter.org/

[6] https://colab.research.google.com/

[7] https://thecleverprogrammer.com/2021/08/24/sarcasm-detection-with-machine-learning/

[8] https://scikit-learn.org/stable/auto_examples/neural_networks/plot_mnist_filters.html#sphx-glr-auto-examples-neural-networks-plot-mnist-filters-py