Machine Learning & Data Mining

Adrian Cooney
12394581
Assignment 1

Open Source Machine Learning Packages

After a fruitless quick, cursory search for a Machine Learning (ML) packages with a GUI, I decided to go down the programming route for building my ML program. After a particularly exhaustive attempt at implementing the kNN algorithm in R, I decided to move away from statistical languages and use open source libraries with a language that I'm comfortable using. I decided to go with Go Lang [1], a very productive language with a promising package ecosystem.

Go has a couple of packages available for Machine Learning. I found eight general purpose machine learning packages that are under active development and have small to medium size communities around them and six other Natural Language Processing and Data Analysis packages [2]. Another library, *mlgo*, was available but unfortunately the project was dead [3].

After research into each package, many of the packages were focused on a singular aspect of ML or had a convoluted API. One standout package was GoLearn [4]. GoLearn has APIs for multiple different classification and regression algorithms. Since Go's package ecosystem is relatively young, there are no *de facto* standards when it comes to choosing packages in relation to ML. To help me make the decision to choose GoLearn, I took into account it's huge popularity on Github (2,179 stars), active community and high quality usage examples.

Data Preparation

GoLearn has an API for the Comma Separated Values (CSV) data format which is similar to the format of the data in *illness.txt*.

According to the CSV standard (RFC4180) [5]: "each record is located on a separate line, delimited by a line break (CRLF)". The data in the *illness.txt* did not follow this rule. Each line in the original data file represented a column and values within those lines separated by a comma were the rows. To convert this horizontal representation of the data to a vertical representation, a CSV Transpose Tool [6] was used and produced a correct translation of the data. The tool erroneously added a trailing comma to each record which was removed.

To accommodate for GoLearn's interesting (and undocumented) API choice to choose the last column in the data as the class for that instance, the `test_result` column had to be moved to the last index.

The last modifications to the data was to add column labels to the data. According to RFC4180: "There maybe an optional header line appearing as the first line of the file with the same format as normal record lines. This header will contain names corresponding to the fields in the file [..]." The column labels were added to the data file as per the assignment spec:

```
plasma_glucose,bp,test_result,skin_thickness,num_pregnancies,insuli
n,bmi,pedigree,age
```

Once the data modifications were complete, a valid CSV file was created and used for the Machine Learning program.

k-Nearest Neighbour Algorithm

The *k-Nearest Neighbour (kNN)* algorithm is a method of classification of data. kNN is *lazy* whereby all calculations are done on data on request. This is opposed to an *eager* algorithm which does all the calculations before any requests and simply find the results based on the request.

To enable a kNN algorithm to classify data, it first must be trained. The training phase of the kNN is comparatively simple to other classification algorithms. It involves no calculations, just simply storing the data for reference during the classification phase.

The classification phase of kNN is the process of getting the Euclidean distance of an new instance's K nearest neighbours, selecting the nearest neighbour and assuming a class based on a majority "vote". A vote is the *class* of a neighbour and given a majority of a class within a group of neighbours, that class is selected to describe the new instance. If no majority vote is found, then a winner is chosen a random from the tying classes. K denotes the size of the voting population of neighbours, with 3 being a good choice of not having too much noise and odd number for a clear winner.

A drawback of the majority voting scheme is that a heavily represented class in an area that may not be "close" (in the Euclidean measure sense) to a new instance may falsely classify it. To counteract this, a weighting scheme is introduced whereby the "closest" neighbour's vote is of greater importance to the final classification of an instance.

In my trials with the *illness.txt* data provided, the kNN algorithm performed with an 84% accuracy rate with $k = 3$. Training and testing was done via holdout technique where around 65% of the data is used to train the classifier and 35% used to test it's accuracy.

Class	True Positive	False Positive	True Negative	Precision	Recall	F1 Score
Positive	42	10	130	0.8077	0.6667	0.7304
Negative	130	21	42	0.8609	0.9286	0.8935

Confusion Matrix of kNN algorithm where $k = 3$ (holdout technique)

To further test the classifier, 10-fold cross-validation was performed. The table below is the average of the 10 confusion matrices produces from the trials. It produced an overall mean accuracy of 71% with 0.002 variance.

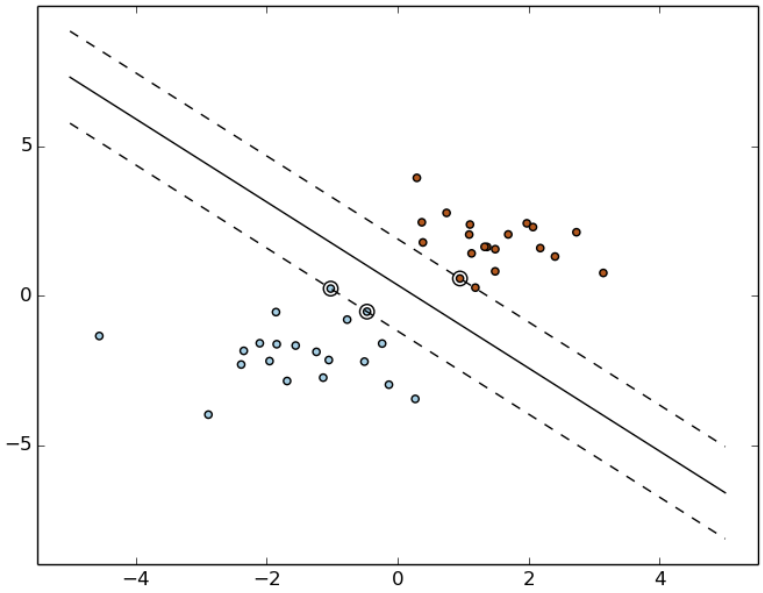
Class	True Positive	False Positive	True Negative	Precision	Recall	F1 Score
Positive	6	5	21	0.5722	0.4715	0.5015
Negative	21	6	6	0.7696	0.8207	0.7903

Averaged Confusion Matrix of kNN algorithm where $k = 3$ (10-fold cross validation)

Linear Support Vector Machine Algorithm

For my second choice of algorithm, I chose a Linear Support Vector Machine (SVM). The linear support vector machine algorithm can classify data that supports binary classification. It takes in

the instances and attempts to create hyperplane that separates the data into their respective classes. When a new instance is to be classified, it's data is computed to decide which side of the hyperplane it lies. This side is the class of the new instance. To improve the accuracy of the classifier, the hyperplane with the largest margin between the classes is desired since it reduces the false positive rate of classification when a new instance is classified.



Example of a 2 dimensional Linear SVM showing line with largest margin. [7]

In my trials with the Linear SVM, I got an overall accuracy of 69% using the holdout technique for training and testing.

Class	True Positive	False Positive	True Negative	Precision	Recall	F1 Score
Positive	10	8	110	0.5556	0.1818	0.2740
Negative	110	45	10	0.7097	0.9322	0.8059

Confusion Matrix of Linear SVM (holdout technique)

Using 10-fold cross validation, I got a overall mean accuracy of 65% with a variance of 0.016. Below is the averaged confusion matrix.

Class	True Positive	False Positive	True Negative	Precision	Recall	F1 Score
Positive	5	6	20	0.5621	0.4073	0.2700
Negative	20	7	5	0.7725	0.7491	0.7036

Average Confusion Matrix of Linear SVM (10-fold cross validation)

Conclusion

After training and testing both algorithms, it seems kNN performs the best on this particular dataset. kNN has nearly half the amount of false positive that Linear SVM classified. Of the data, it

seems that the negative class was the hardest to predict. The holdout technique for training and testing seem to yield the greatest accuracy for kNN while the converse is true for Linear SVM.

Retrospective

In the beginning, I had a lot of trouble with GoLearn. The documentation was poor, a factor which should have had greater influence on my initial decision to choose a package. I spent quite a lot of time understanding the underlying implementation which was time consuming but aided greatly in the learning process of kNN, SVM and other algorithm implementations.

References

1. Golang.org, (2015). The Go Programming Language. Available at: https://golang.org/ [Accessed 28 Sep. 2015].
2. Github.com (2015). josephmisiti/awesome-machine-learning. Available at: https://github.com/josephmisiti/awesome-machine-learning#go [Accessed 28 Sep. 2015].
3. Google.com (2015). mlgo - Machine Learning algorithms in Go - Google Code Hosting. Available at: https://code.google.com/p/mlgo/ [Accessed 28 Sep. 2015].
4. Github.com (2015). sjwhitworth/golearn. Available at: https://github.com/sjwhitworth/golearn [Accessed 28 Sep. 2015].
5. IETF.org (2015). “RFC 4180 - Common Format..”. Available at: https://tools.ietf.org/html/rfc4180 [Accessed 28 Sep. 2015].
6. ClientSideWeb.net (2015). CSV Transpose Tool. Available at: http://csvtransposetool.appspot.com/ [Accessed 28 Sep. 2015].
7. scikit-learn.org (2015). 1.4 Support Vector Machines - scikit-learn 0.16.0 documentation. Available at: http://scikit-learn.org/stable/modules/svm.html#mathematical-formulation [Accessed 12 Oct. 2015].