Assignment 2

October 28, 2015

Adrian Coooney (12394581, 4BCT)

1 Preface

In my last assignment I used Go Lang. In hindsight, this was a poor decision because Go is still a very young language with a developing ecosystem. The package I used for the last assignment, GoLearn[1], was great for educational purposes and the task at hand but inadequete when attempting any further complex data science. Considering this, I decided to jump into the mature Scikit-Learn[2] and Python in my long journey from Go to R and back to Go.

2 Question

Construct learning curves for the two classification algorithms that you used in Assignment 1. A description of your methodology to construct learning curves, including any assumptions you made and procedures you followed. If using a package that does the work for you, explain what it does in sufficient detail to demonstrate your clear understanding of it [4 marks].

Graphs of the learning curves for each of the two algorithms [1.5 x2 = 3 marks]. A discussion of your observations from examining the two learning curves, individually and/or taken together. You must make 6 separate observations and will score up to 0.5 marks for each observation that is appropriately discussed and backed up by evidence [0.5 x6 = 3 marks].

3 Import

First let's input our data and libraries. Use `pandas` to read the CSV data into a `DataFrame` for easy manipulation. Next, we separate our class and attribute columns for use in the classification step.

In [2]:
```
%matplotlib inline
%config InlineBackend.figure_format = 'svg'

import pandas
import numpy as np
from pylab import *
from sklearn.learning_curve import learning_curve

# Import the CSV data
data = pandas.read_csv('./data/illness-mapped.csv')

class_name = "test_result"
attributes = list(set(data.columns.values) - set([class_name]))
```

4 Learning Curves

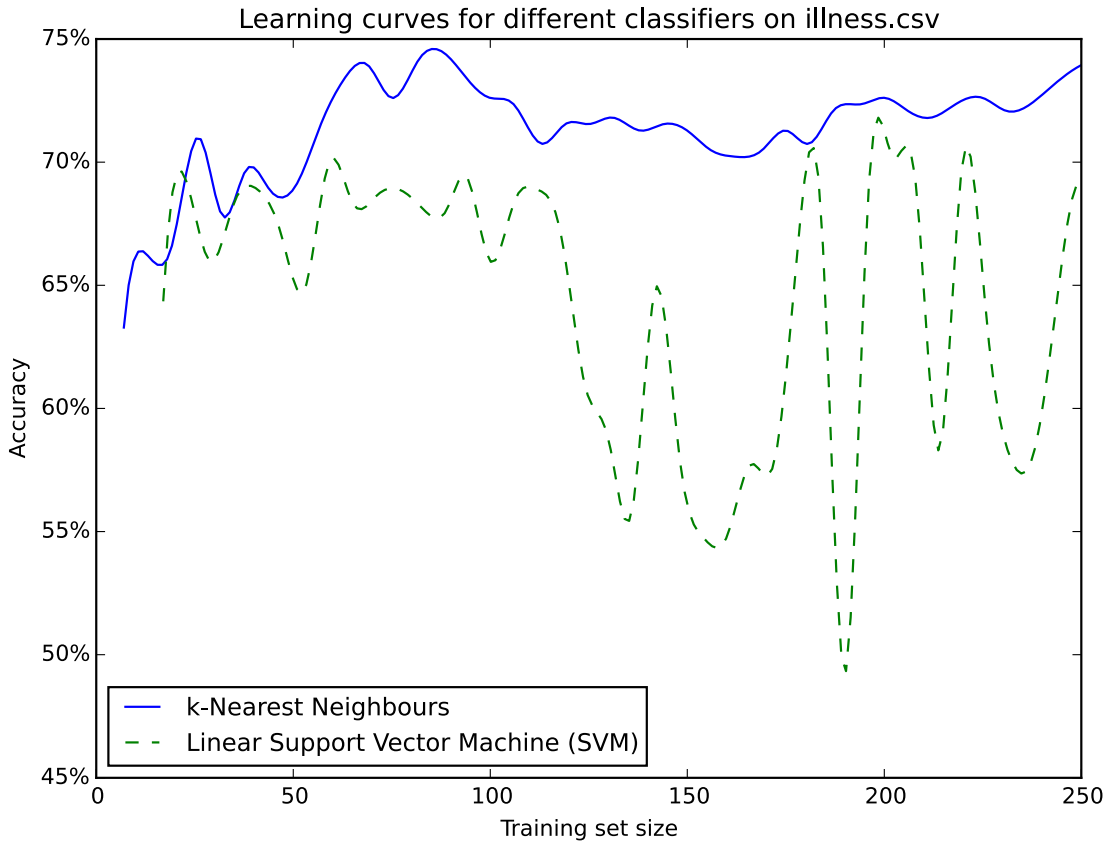
To get the learning curve of any classification, we need to plot the accuracy of the classifier versus the increase in size of the training set. This involves iteratively increasing the size of the training set (and thus decreasing the size of the testing set) as long as $|S_{train}| > 1$ and $|S_{test}| > 1$.

We create the learning curve by first generating our x values which descibe the percentage size of the overall data we will use as S_{train}. This is an array of values $\{\sigma \leqslant x \leqslant 1 - \sigma\}$ where $\sigma = 0.01$ and is also the interval between values. Next, we loop over each value in x to generate our $f(x)$ values. For each value in x, we split our data into two sets S_{train} and S_{test} where $|S_{train}| = x_n \times |D|$ and $|S_{test}| = |D| - |S_{train}|$. We then train our classifier using S_{train} and test with S_{test}. $f(x)$ is the accuracy of the classifier at predicting the data. Once we have our x and $f(x)$, we simple can plot and give us our learning curve.

Scikit-Learn has some fantastic utilities available for analyzing different algorithms. Included in this toolkit is a `learning_curve` function which does all the heavy lifting for us. `learning_curve` will take in our classifier, get results decsribed above and return them. We then plot these results using another fantastic library, Matplotlib[3]. To improve the aesthetics, we employ a simple curve smoothing algorithm (`spline`) to remove the jagged edges. For resuability, we define a simple `plot_curve` function which takes in our classifier, generates the curve and adds it to the plot.

The algorithms I used in my last assignment were the k-Nearest Neighbour and Linear Support Vector Machine (SVM).

```
In [9]: from sklearn.neighbors import KNeighborsClassifier
        from sklearn.svm import LinearSVC
        from sklearn.naive_bayes import GaussianNB
        from matplotlib.ticker import FuncFormatter
        from scipy.interpolate import spline

        # X values (% split of data)
        def plot_curve(classifier_name, classifier, train_sizes, line="-"):
            # Generate the learning curve
            x, train, test = learning_curve(classifier,
                       data[attributes], data[class_name],
                       train_sizes=train_sizes)

            # Plot the score and smooth the graph
            xd = np.linspace(np.amin(x), np.amax(x), 200)
            plot(xd, spline(x, np.mean(test, axis=1), xd), line, label=classifier_name)

        figure(figsize=(8, 6))

        plot_curve("k-Nearest Neighbours", KNeighborsClassifier(n_neighbors=3),
                   np.linspace(0.03, 1.0, 40))
        plot_curve("Linear Support Vector Machine (SVM)", LinearSVC(random_state=1),
                   np.linspace(0.07, 1.0, 40), line="--")

        title("Learning curves for different classifiers on illness.csv")
        xlabel("Training set size")
        ylabel("Accuracy")

        # Convert ticks to percentages
        gca().get_yaxis().set_major_formatter(FuncFormatter(lambda x, pos: "%d%%" % (x * 100)))
        legend(loc="best", prop={'size':11})

        show()
```



5 Observations

1. The accuracy of the Linear SVM algorithm fluctuates very heavily and has a very inconsistent pattern of accuracy as the training size increases. This shows us that the Linear SVM classifier is unable to accurately classify our data and thus is a bad choice of algorithm for this particular dataset.
2. The Linear SVM couldn't classify data without error until at least 20 samples of data were used in training.
3. k-Nearest Neighbour accuracy grows as the training set does and evens out, hovering around 72-74% accurate.
4. k-Nearest Neighbour performs the best out of the two on the given dataset.
5. k-Nearest Neighbour is most accurate when the training/test split is at 33%/66%. This is due to there being enough training samples to get an accurate classification and enough test data to verify the classification.
6. Neither algorithm is particularly well suited to the dataset. The accuracy score is unnacceptable.

References

[1] Github.com (2015). *sjwhitworth/golearn*. Available at: https://github.com/sjwhitworth/golearn [Accessed 20 Oct. 2015].

[2] Scikit-learn.com (2015) *scikit-learn: machine learning in Python ..* Available at: http://scikit-learn.org/ [Accessed 20 Oct. 2015].

[3] Matplotlib.org (2015) *matplotlib: python plotting — Matplotlib 1.4.3 documentation* Available at: http://matplotlib.org/ [Accessed 21 Oct. 2015]