1. What is the difference between a rule-based system and a machine learning system?

A rule-based system is implemented as a set of rules, which is usually executed in some sequential or conditional flow, which means the exact steps of execution is known.

A machine learning system includes some kind of model which can be updated iteratively and used to predict a value; this can be singular or multi-dimensional. Mathematically, this equates to learning a function f(x) = y where x and y can be singular or multi-dimensional.

For real-world problems, there are too many unknowns for a rule-based system to be useful for solving these problems, whereas a machine learning system can be used heuristically.

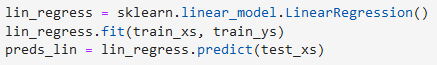
1. What is the difference between unsupervised and supervised learning?

The major difference is the input data, where data for supervised learning includes labels for the ground truth, whereas data for unsupervised learning doesn’t include these labels. The reason for unsupervised learning is that data is expensive to label, since it requires human interaction, and so unsupervised learning can be used on extremely large, unlabelled datasets. But it’s limited in output, since it can’t predict values or classifications, but can cluster or autoencode data. The mid-ground is hybrid learning, where some data is labelled but most isn’t, meaning predictions can be made while learning on mostly unlabelled data.

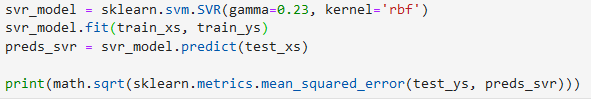
1. What do we mean when we say that a machine learning system is overfitting?

An initial machine learning model is said to be simple, that is it doesn’t adequately represent the input data. Through training and updating of the model, it is said to become more complex, that is a closer representation of the data. The issue is that a model can become overly complex, and while it may fit the training data well, it does not when new data is introduced. This is called overfitting, where the training error is very low, but the validation error is very high. Regularisation and early stopping are two methods to combat this.

1. Part 1 question 2.1 - include all steps
2. Part 1 question 2.2 – include Python snippets and RMSE performance



RMSE = 0.7743778617318132



RMSE = 0.7263170196336258

Train an SVM binary classifier using the Hateval dataset (available in Learning Central). The task consists of predicting whether a tweet represents hate speech or not. You can preprocess and choose the features freely. Evaluate the performance of your classifier in terms of accuracy using 10-fold cross-validation. Write a table with the results of the classifier (accuracy, precision, recall and F-measure) in each of the folds and write a small summary (up to 500 words) of how you preprocessed the data, chose the feature/s, and trained and evaluated your model (35%)

For data pre-processing, I chose to represent each sentence as a vector of word frequency. That is, I first created a dictionary of all the words in the training set and ordered by frequency in descending order. Only the top 1000 most frequently used words were used, so each data item is represented as a 1000-feature vector, where the n-th element in the vector is the frequency of the n-th word in the dictionary found in the data item.

Before this though, each data item was first lemmatised, which unlike stemmatisation depends on correctly identifying the intended part of speech. This helped simplify the dictionary, since ‘going’, ‘went’ and ‘gone’ would all be changed to ‘go’.

For feature selection, I introduced the chi-squared test method, which removes features that appear to be irrelevant to the label. In this case, I selected the 500 most relevant features from the 1000 features the word frequency dictionary gave us. Since the chi-squared requires that vector elements be non-negative, I also made a copy of the vectorised dataset and standardised the data, that is to change the mean of each feature to be 0 and the standard deviation to be 1. This meant I could compare feature selection against data standardisation.

The first training and evaluation stage didn’t involve cross-validation, instead I trained three separate Support Vector Machines (sklearn.svm.SVC), the first was trained on just the vectorised data, the second was trained on the standardised data, and the third was trained on the feature-selected data using chi-squared. The reason for this was to see quickly if there was any difference in the accuracy, precision etc. when tested on the test set. What could be seen was that the first model was slightly below (as little as 1%) in most scores, whereas the standardised data and feature-selected data scored almost exactly the same. However, going forward I decided that the feature-selection would be more appropriate than standardisation, since less features means faster training of the model and faster prediction.

Next, cross-validation was done with 10-fold and a list of parameter values for chi-squared feature size, from 1000 i.e. all features to 10. The accuracy, precision, recall and f1-score were recorded for each fold, and below is that table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Chi-squared size | Accuracy | Recall | Precision | F1 Score |
| 1000 | 0.813333 | 0.731844 | 0.784431 | 0.757225 |
| 750 | 0.798889 | 0.678977 | 0.778502 | 0.725341 |
| 500 | 0.79 | 0.671795 | 0.811146 | 0.734923 |
| 250 | 0.773333 | 0.689744 | 0.764205 | 0.725067 |
| 150 | 0.758889 | 0.630556 | 0.729904 | 0.676602 |
| 100 | 0.772222 | 0.671756 | 0.776471 | 0.720327 |
| 75 | 0.763333 | 0.656863 | 0.785924 | 0.715621 |
| 50 | 0.756667 | 0.613514 | 0.749175 | 0.674591 |
| 25 | 0.748889 | 0.552083 | 0.796992 | 0.652308 |
| 10 | 0.718889 | 0.449735 | 0.790698 | 0.573356 |

We can see, apart from precision, that the accuracy score, recall and f1 score decline when selecting fewer features, so here there is a trade-off between accuracy and speed of training and predicting, since less features means less evaluation.

10-fold was used again for gaining a better average for these scores by keeping parameter values static. Below are the tables for feature size = 1000 and 500

Feature size = 1000

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| K-fold | Accuracy | Recall | Precision | F1 Score |
| 1 | 0.778889 | 0.680441 | 0.74848 | 0.712843 |
| 2 | 0.785556 | 0.669231 | 0.80308 | 0.73007 |
| 3 | 0.764444 | 0.635359 | 0.74194 | 0.684524 |
| 4 | 0.781111 | 0.678378 | 0.76292 | 0.718169 |
| 5 | 0.782222 | 0.640506 | 0.8241 | 0.720798 |
| 6 | 0.782222 | 0.669065 | 0.82789 | 0.740053 |
| 7 | 0.79 | 0.689474 | 0.78679 | 0.734923 |
| 8 | 0.783333 | 0.686486 | 0.76276 | 0.722617 |
| 9 | 0.78 | 0.661017 | 0.75 | 0.702703 |
| 10 | 0.784444 | 0.672775 | 0.78834 | 0.725989 |
|  |  |  |  |  |
| Average | 0.781222 | 0.668273 | 0.77963 | 0.719269 |

Feature size = 500

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| K-fold | Accuracy | Recall | Precision | F1 Score |
| 1 | 0.758889 | 0.634465 | 0.759375 | 0.691323 |
| 2 | 0.802222 | 0.705722 | 0.787234 | 0.744253 |
| 3 | 0.786667 | 0.661417 | 0.8 | 0.724138 |
| 4 | 0.775556 | 0.67013 | 0.774775 | 0.718663 |
| 5 | 0.781111 | 0.685237 | 0.745455 | 0.714078 |
| 6 | 0.787778 | 0.656992 | 0.803226 | 0.722787 |
| 7 | 0.801111 | 0.708108 | 0.786787 | 0.745377 |
| 8 | 0.784444 | 0.672872 | 0.780864 | 0.722857 |
| 9 | 0.757778 | 0.678947 | 0.728814 | 0.702997 |
| 10 | 0.784444 | 0.679901 | 0.80826 | 0.738544 |
|  |  |  |  |  |
| Average | 0.782 | 0.675379 | 0.777479 | 0.722502 |

Below is the result of the test set using a model with a feature size of 500 after using chi-squared.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support |
| 0 | 0.79 | 0.86 | 0.83 | 1566 |
| 1 | 0.78 | 0.68 | 0.73 | 1135 |
|  |  |  |  |  |
| accuracy |  |  | 0.79 | 2701 |
| macro avg | 0.79 | 0.77 | 0.78 | 2701 |
| weighted avg | 0.79 | 0.79 | 0.79 | 2701 |

Part 2

In Part 2, students are provided with a sentiment analysis dataset (IMDb). The dataset

contains positive and negative movie reviews. Training, development and test splits are

provided. Based on this dataset, students will be asked to preprocess the data, select

features and train a machine learning model of their choice to solve this problem. Students

should include at least three different features to train their model, one of them should be

based on some sort of word frequency. Students can decide the type of frequency (absolute

or relative, normalized or not) and text preprocessing for this mandatory word frequency

feature. The remaining two (or more) features can be chosen freely. Then, students are

asked to perform feature selection to reduce the dimensionality of all features.

Description of all steps taken in the process (preprocessing, choice of features,

feature selection and training and testing of the model). **(25% - The quality of the**

**preprocessing, features and algorithm will not be considered here)**

2) Justification of all steps. Some justifications may be numerical, in that case a

development set is included to perform additional experiments. **(25% - A reasonable**

**reasoned justification is enough to get half of the marks here. The usage of the**

**development set is required to get full marks)**

3) Overall performance (precision, recall, f-measure and accuracy) of the trained model

in the test set. **(10% - Indicating the results, even if very low, is enough to get half**

**of the marks here. A minimum of 65% accuracy is required to get full marks)**

4) Critical reflection of how the deliverable could be improved in the future and on

possible biases that the deployed machine learning may have. **(15% - The depth and**

**correctness of insights related to your deliverable will be assessed)**

Differently to implementing previous questions, here I decided to implement this problem as a class; this way I can have methods for intermediate steps, and I can use the ‘pickle’ library to serialise and save the object, which includes all the data and models, to load and de-serialise for later use or to re-run a subset of experiments. The reason for this is the pre-processing, training and predicting can take a long time, on the order of minutes. We will see later in the report that the training and prediction times can be reduced drastically, but the pre-processing time remains the same.

# Pre-processing and choice of features

Initially, the data is cleaned up using regex for punctuation, HTML tags etc. which leaves the text for each data item clean. Next the data is lemmatised, similar to the previous question. From this, a dictionary is created of all the words in the training set and ordered by frequency in descending order. Unlike before, the top 2000 words are used for the dictionary; the justification for this value can be seen later in the report. This gives a feature vector for each data item of dimension 2000.

On top of this, I also decided to convert the data to vectors of TF-IDF features, which is similar to the dictionary created, except that it is inversely weighted by the frequency over multiple documents, which in this case are the data items. This will be useful for words which are important to a small subset of reviews, and will have almost zero or zero scores for words that are high-frequency over all documents, such as ‘and’, ‘the’ etc.

TODO: talk about feature size of TF-IDF once the experiments have been run .

The third pre-processing feature involves the use of a sentiment analysis library called VADER (Valence Aware Dictionary and sEntiment Reasoner), which can be found on Github with the following link: <https://github.com/cjhutto/vaderSentiment>. Although it is tuned specifically for sentiments on social media, it will work equally well with reviews. The input is a string, in this case the concatenated lemma terms from each review, and the output is a 4-dimensional vector with scores for negativity, neutrality, positivity and compound, which is a score by ‘summing the valence scores of each word in the lexicon, adjusted according to the rules, and then normalised to be between -1 (most extreme negative) and +1 (most extreme positive)’.

To create the final representation of the review, we can concatenate all the features together, giving a 2000+x+4 = 2004+x (where x is TF-IDF feature size) vector representation. For the training set there are 15,000 labelled reviews, giving a 15,000x2004 matrix, with each row having a label of 0 or 1 depending on whether it is a positive (1) or negative(0) review.

# Feature selection

For the previous question, feature selection using the chi-squared test was used. Here instead, we will use PCA (principal component analysis), since unlike the chi-squared test it allows for negative numbers, meaning the data can be standardised first. PCA also allows for more features to influence the final value than dimensions of the final value, since it’s a projection into a lower-dimension space. For this, a dimension of size 20 was chosen as a trade-off for speed and accuracy, as can be seen in the justification section. The PCA is fitted to the training data, and then used to project the training, test and development sets.

TODO: write a function to save the important models, being svm, pca transformer, vocabulary etc. so they can be loaded and used on other datasets in other programs

TODO: include gamma as a cross-validation parameter to be tuned, since it can change the accuracy a decent amount compared to ‘scale’

# Training

The machine learning method used is Support Vector Classification, similar to the previous question; the reason being for the sentiment analysis on tweets it achieved a high accuracy, and seems to be a good standard for classification problems. The kernel used is a radial based function, which allows for non-linear learning. The gamma value is set to x, again the justification for which can be seen later in the report.

Two models are trained for comparison; the first is trained on the full feature matrix, which is 2004+x features for each data item and standardised, the second is trained on this dataset after being projected using PCA to a dimension of 20 features. This is to be able to make comparisons in speed and accuracy, so the time taken for each model to be trained is recorded.

# Testing and results

The trained models are then used to predict values for the test set as input, of which there are two copies as explained in the above section. Again, the time for predicting for both models is recorded.

As the results show below, while the metric scores (accuracy, recall, precision, f1-score) are reduced slightly through using the PCA transformed data, but the speed of training and prediction was reduced by 99.4%. This shows that the use of PCA is extremely important when applied to this problem. // note: talk about the change in time for projecting PCA against not projecting PCA

# Development set experiments and justification of parameters