Short course on adaptive modelling: Lab session

prepared for 2012/13 class by: Dino Sejdinovic
previous versions by: John Shawe-Taylor, Tom Diethe

May 14, 2013

Abstract

Using a series of examples, in this exercise session you will familiarise yourselves with the Naive Bayes Classifier and Support Vector Machines. This exercise sheet is available at `http://www.gatsby.ucl.ac.uk/~dino/lab/handout.pdf`

Keywords: Naive Bayes — Support Vector Machines (SVM)

1 Naive Bayes classifier

Assume that we have training examples $S = \left\{(\mathbf{x}^{(i)}, y^{(i)})\right\}_{i=1}^{m}$, where each $\mathbf{x}^{(i)} = \left(x_1^{(i)}, \ldots, x_D^{(i)}\right)$ is a D-dimensional vector and $y^{(i)}$ is the corresponding label. For a new data point $\mathbf{x}_{tst}$, we wish to predict its label y_{tst} using the Bayes theorem:

$$\begin{aligned} y_{tst} &= \arg\max_y P(y|\mathbf{x}_{tst}) \\ &= \arg\max_y \frac{P(\mathbf{x}_{tst}|y)P(y)}{P(\mathbf{x}_{tst})} \\ &= \arg\max_y P(\mathbf{x}_{tst}|y)P(y), \end{aligned}$$

since denominator does not depend on y. However, this requires estimation of a high-dimensional probability distribution $P(\mathbf{x}|y)$, which is impossible in most interesting cases. Naive Bayes makes a strong (*naive*) independence assumption on this probability distribution, i.e., that

$$P(\mathbf{x}|y) = \prod_{j=1}^{D} P(x_j|y),$$

i.e., individual components of $\mathbf{x}$ are conditionally independent given its label y. Classifier then proceeds by estimating D one dimensional distributions $P(x_j|y)$,

which is a much easier task. For example, when variables x_j are binary, estimation of $P(x_j|y)$, can be expressed through:

$$p_{jk} \quad = \quad P(x_j = 1|y = k), \quad \pi_k = P(y = k),$$

with maximum likelihood (ML) estimates given by:

$$\begin{aligned} \hat{p}_{jk} &= \frac{\#\{(\mathbf{x}, y) \in S \,:\, x_j = 1, y = k\}}{\sum_{l=1}^{D} \#\{(\mathbf{x}, y) \in S \,:\, x_l = 1, y = k\}}, \\ \hat{\pi}_k &= \frac{\#\{(\mathbf{x}, y) \in S \,:\, y = k\}}{m}. \end{aligned} \tag{1}$$

1.1 Dataset

We will use a real-world dataset of text extracted from Yahoo! pages which has been used in previous studies ([1, 2]). The original dataset was split into 3 categories (Sport, Aviation, Paintball), but we will only be using two of these.

- Download the data from:
  `http://www.gatsby.ucl.ac.uk/~dino/lab/TextData.mat`
- Load the file `'TextData.mat'` into Matlab. The cell array `Docs` contains the original text documents (400 from each of the three categories). Have a look at the first documents from 'Sport' and 'Aviation':

  ```
  » Docs{1}
  » Docs{401}.
  ```

We will construct a classifier that predicts whether a given document belongs to 'Aviation' ($y = -1$) or 'Paintball' ($y = 1$) category. As working with text itself is unwieldy, each document is represented by a D-dimensional vector encoding the number of times that each of the D words in the dictionary (the union of all words occurring in any of the texts) occurs in that document - the so called 'bag of words' representation (this is clearly a very crude representation since it does not take into account the order of the words). This leads to a notion of similarity (kernel) between two text documents, as an inner product between these 'bag of words' representations, given by:

$$k(\texttt{text 1}, \texttt{text 2}) = \sum_{\texttt{word} \in \texttt{dictionary}} (\#\text{occ. of } \texttt{word} \text{ in } \texttt{text 1}) \times (\#\text{occ. of } \texttt{word} \text{ in } \texttt{text 2}).$$

The initial stages of extracting the dictionary and calculating the document frequency have been performed for you. The matrix `TD` contains the numbers of occurrences of each of the $D = 3522$ words in the dictionary (given in `Terms`) for each of the 1200 documents. Thus each $\mathbf{x}$ is a vector of length 3522.

1.2 Running a naive Bayes classifier

Exercise 1 (Preparing the data)**.** In this section we will load the data and separate it into training and testing sets.

- Let us split the data into training and testing sets. We will be using the last two categories (Aviation, Paintball) of the data. Create a random permutation of indices as follows. Note that we used two independent permutations of length 400 (number of documents in each class) and added 400 and 800 respectively to access class Aviation (indexed between 401 and 800) and class Paintball (indexed between 801 and 1200).

  ```
  » perms = [randperm(400)+400, randperm(400)+800];
  ```

  Allocate 50 documents from each category for training and the remainder for testing:

  ```
  » idxstrn = [perms(1:50), perms(401:450)];
  » idxstst = [perms(51:400), perms(451:800)];
  ```

  Select the appropriate part of the data for training and testing as follows:

  ```
  » Xtrn = TD(:,idxstrn)';
  » ytrn = Target(idxstrn)';
  » vals = unique(ytrn);
  » ytrn(ytrn==vals(1)) = -1;
  » ytrn(ytrn==vals(2)) = 1;
  » Xtst = TD(:,idxstst)';
  » ytst = Target(idxstst)';
  » vals = unique(ytst);
  » ytst(ytst==vals(1)) = -1;
  » ytst(ytst==vals(2)) = 1;
  ```

  Note that we have transposed the matrices (X and y).

Exercise 2 (Training the Naive Bayes classifier)**.** In this section we will train the naive Bayes classifier on the data. Before proceeding, consider the following question: why is ML estimate in (1) a bad idea in the case where for some j, $x_j = 0$ in all of the training examples? For example, assume that none of the training text documents contains the j-th word in the dictionary? What happens when this word appears in the test document? How would you modify the estimate to rectify this problem?

a. The naive bayes classifier has been provided for you. You might like to examine the algorithm and try to understand how it works and relate it

to the question above. Download the file `naive_bayes.m` from:

`http://www.gatsby.ucl.ac.uk/~dino/lab/naive_bayes.m`

b. Train the classifier on the training data, and calculate the predicted classes for the testing data as follows:

```
» [ypred_NB] = naive_bayes(Xtrn, ytrn, Xtst);
```

Examine the plots of the predictive posterior for each of the classes.

c. Calculate the percentage of correct predictions of the classifier on the test set and print it to the screen as follows:

```
» err_NB = mean(ypred_NB~=ytst);
» display(sprintf('Accuracy %.2f', 100*(1-err_NB)));
```

Check how many documents in each of the classes were misclassified and compare this with the predictive posterior plots. Experiment with different sizes of splits of training and test data, and mark how performance depends on the size of the training data.

1.3 Summary

Naive Bayes is one of the simplest density estimation methods from which we can form one of the standard classification methods in machine learning. Its fame is partly due to the following properties:

- Very easy to program and intuitive
- Fast to train and to use as a classifier
- Very easy to deal with missing attributes
- Very popular in fields such as computational linguistics/NLP

As we have seen, Naive Bayes can be useful in classification of text documents. The reason that Naive Bayes may be able to classify documents reasonably well in this way is that the conditional independence assumption is not so silly: if we know people are talking about politics, this perhaps is almost sufficient information to specify what kinds of other words they will be using - we don't need to know anything else (of course, if you want ultimately a more powerful text classifier, you need to relax this assumption).

2 Support Vector Machines (SVM)

We will describe two possible ways of running Vapnik's Support Vector Machines [3] for classification in Matlab. One is based on Matlab's built in functions

and the other is the SVMlight implementation. There are many other SVM implementations available online.

2.1 SVM with Matlab's toolbox

We will be using an implementation of SVMs in Matlab's Bioinformatics toolbox.

Ensure that the Bioinformatics toolbox is included in your Matlab version using:

```
» ver
```

If this toolbox is not available on your machine, you should ignore the rest of this Section and continue to Section 2.2

Two main functions that we will use are `svmlearn` and `svmclassify`. You can see the help by typing:

```
» help svmtrain
```

You should see some information about the function printed on the screen.

Exercise 3 (Training the SVM)**.**

a. Train the SVM using a value of 0.5 for the C-parameter (trade-off between the training error and the size of the margin) and a linear (Bag of Words) kernel as follows:[1]:

```
» svmStruct = svmtrain(Xtrn, ytrn, 'boxconstraint',0.5);
```

The output structure `svmStruct` contains the learnt model. The element 'SupportVectors' holds the support vectors which are the points required for classification. Note that the number of support vectors is significantly smaller than the number of datapoints, and hence the SVM learns a model that is "sparse".

b. Predict the labels on the test set using function `svmclassify`, calculate the percentage of correct predictions and print it to the screen as follows:

```
» ypred_SVM = svmclassify(svmStruct, Xtst);
» err_SVM = mean(ypred_SVM~=ytst);
» display(sprintf('Accuracy %.2f', 100*(1-err_SVM)));
```

[1] in case you get an error stating: `Undefined function 'optimset' for input arguments of type 'char'`, run the following two comands:

```
» restoredefaultpath
» rehash toolboxcache
```

2.2 SVMlight

You should read this Section only if you do not have a Bioinformatics toolbox available. Otherwise, continue to Section 2.3.

SVMlight is an implementation of SVMs which can be used for classification, regression, and preference ranking. The optimization algorithms used in SVMlight are described in [4, 5]. The algorithm has scalable memory requirements and can handle problems with many thousands of support vectors efficiently. Consult `http://svmlight.joachims.org/` for more details.

You will need to download and install the Matlab interface to SVMlight. The precompiled libraries are available at:

`http://www.gatsby.ucl.ac.uk/~dino/lab/SVM_lightMex.zip`

Ensure that the path to the installation directory is added to the Matlab path by right-clicking the installation folder and selecting option *Add to Path→Selected Folders and Subfolders* or by typing:

```
» addpath('/install-path/SVM_lightMex/bin/');
» addpath('/install-path/SVM_lightMex/matlab/');
```

where `install-path` is the path to your installation (e.g. `~/svm_light`)

Test your installation by typing: `» help svmlearn`
You should see some information about the function printed on the screen.

Exercise 4 (Training the SVM)**.** In this section we will use SVMLight to classify our data.

a. Train the SVM using a value of 0.5 for the C-parameter (trade-off between training error and margin) and a linear (Bag of Words) kernel as follows:

```
» model = svmlearn(Xtrn, ytrn, '-v 0 -t 0 -c 0.5');
```

The text string `'-v 0 -t 0 -c 0.5'` specifies the various model properties. For example, you can choose a Gaussian kernel with `'-t 2'` and set its bandwidth to $\gamma = 0.1$ with `'-g 0.1'`. The output structure contains the learnt model. The element `supvec` holds the support vectors which are the points required for classification. Note that the number of support vectors is significantly smaller than the number of datapoints, and hence the SVM learns a model that is "sparse".

b. Calculate the percentage of correct predictions of the classifier on the test set and print it to the screen as follows:

```
» [err_SVM, ypred_SVM] = svmclassify(Xtst, ytst, model);
» display(sprintf('Accuracy %.2f', 100*(1-err_SVM)));
```

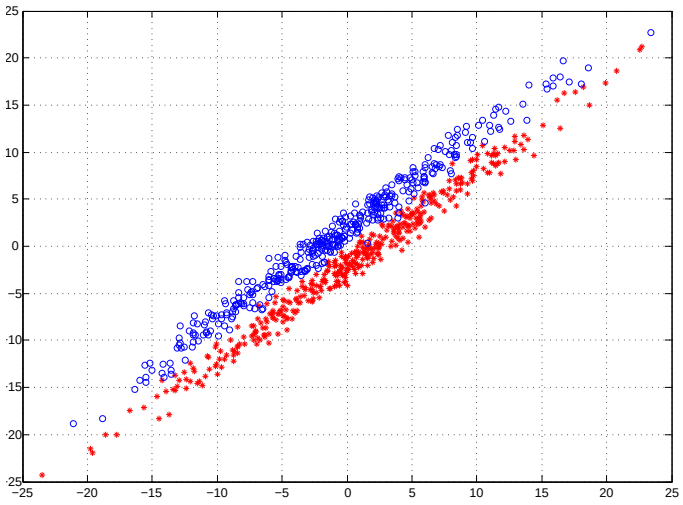

Figure 1: Elongated Gaussians

2.3 When Naive Bayes fails

Download the script from:

`http://www.gatsby.ucl.ac.uk/~dino/lab/gaussexample.m`

The data consists of two elongated Gaussian distributions depicted in Fig. 1: positive examples have mean $(1, -1)$ and negative examples have mean $(-1, 1)$. Looking at the data set, why do you think naive Bayes approach fails to classify correctly? Hint: are dimensions conditionally independent given the label? To run the naive Bayes classifier on this example, you will need the matlab function that uses a gaussian fit along each dimension, available at:

`http://www.gatsby.ucl.ac.uk/~dino/lab/naive_bayes_gaussianfit.m`

Is the dataset still linearly separable? Apply an SVM with a linear kernel and compare the results.

2.4 Non-linearly separable data

We will now consider a simple synthetic data which illustrates the strength of non-linear kernels. Download the script from:

`http://www.gatsby.ucl.ac.uk/~dino/lab/mixtureexample.m`

This script generates bivariate data which looks similar to that in Figure 2, where positive examples are a mixture of bivariate gaussians with means (-1,-1) and (1,1) while the negative examples are a mixture of bivariate gaussians with means (-1,1) and (1,-1). All gaussian components have the same covariance matrix.

In addition to the naive Bayes classifier and SVM with a linear kernel, also run SVM with a gaussian rbf kernel $k(x, x') = \exp(-\sigma \|x - x'\|_2^2)$ with pa-

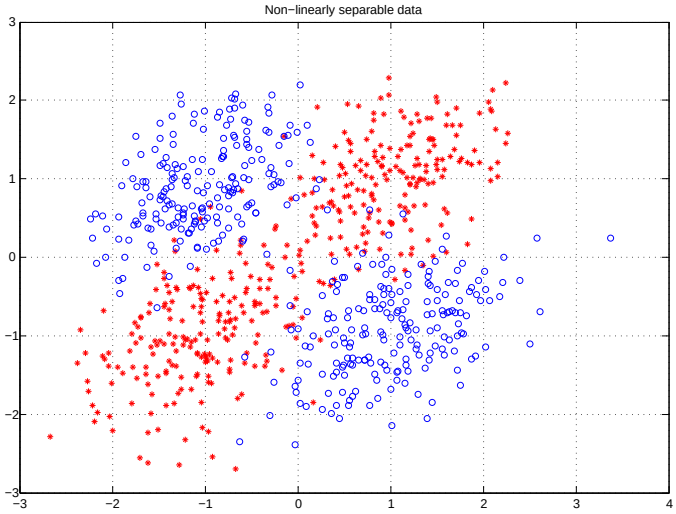

Figure 2: An example of non-linearly separable data in $\mathbb{R}^2$

rameters $\sigma = 1$ (`rbf_sigma`), and $C = 0.01$ (`boxconstraint`). Note that the SVM with a linear kernel finds a maximum-margin line that divides two sets of training examples. Since no line can separate our data, the method performs poorly. On the other hand, SVM with a gaussian kernel first maps the data from $\mathbb{R}^2$ into a higher dimensional feature space, where the resulting "features" can be linearly separated, and a maximum-margin hyperplane is fitted in this space. When mapped back to our original space, the classifier is non-linear. To get some intuition of how this works, you can have a look at `http://www.youtube.com/watch?v=3liCbRZPrZA`, where the polynomial kernel with a three-dimensional feature space is used. In general, feature space can have a large and even an infinite number of dimensions.

Exercise 5. Experiment with the file `mixtureexample.m`. In particular, make the classification problem harder by introducing additional dimensions of i.i.d. gaussian random variables, which serve as the noise. Study how performance changes with the number of dimensions. You could also try to generate a different kind of data, e.g., by "pulling" the positive and negative examples nearer to each other: set the parameters `s1` and `s2` to different values and rotate the eigenvectors by angle `theta`.

2.5 Cross-validation

In the previous exercise, we chose a single value of the parameters σ and C for the SVM. In general, classification performance can depend strongly on the choice of these (tuning) parameters. There are several approaches to choosing tuning parameters. One often used approach is *k-fold cross-validation*. It divides the

training set in k disjoint sets. Each of these k sets of samples is once lifted out as the validation set, and the remaining $k-1$ sets are used for training. As a result, we get k validation scores. The average of these scores is used as a good estimate of the test set performance.

Exercise 6 (Tuning the C parameter using cross-validation)**.** Use 5-fold cross validation to tune the C parameter.

a. Firstly create a vector of values for the C parameter between 2^{-6} and 2^{8} in steps of 1 for the exponent (i.e. $2^{-6}, 2^{-5}, \ldots, 2^{7}, 2^{8}$)

b. Next create a random permutation of the indices of the training set using `randperm()`. You can then split these indices into groups by taking the remainder of division by the number of folds using `mod()`.

c. You will then need to iterate for the number of folds, choosing the appropriate indices for training and validation. Within this loop you will need to evaluate each of the values of the C parameter (i.e. a nested for loop). Store the validation error for each fold and each value of C in a matrix.

d. Next you need to average over the folds, and choose the value of C that minimises the mean validation error. Then retrain using this value of the C parameter on the full training set, and calculate the percentage of correct predictions of the classifier on the test set as in the previous exercise. Does this method yield improved performance?

References

[1] T. Kolenda, L. K. Hansen, J. Larsen, and O. Winther (2002). Independent component analysis for understanding multimedia content. In H. Bourlard, T. Adali, S. Bengio, J. Larsen, and S. Douglas, editors, Proceedings of IEEE Workshop on Neural Networks for Signal Processing XII, pages 757-766, Piscataway, New Jersey, 2002. IEEE Press. Martigny, Valais, Switzerland, Sept. 4-6, 2002.

[2] A. Vinokourov, D. R. Hardoon and J. Shawe-Taylor (2003). Learning the Semantics of Multimedia Content with Application to Web Image Retrieval and Classification. 4th International Symposium on Independent Component Analysis (ICA 2003), Nara, Japan.

[3] V. Vapnik (1995). The Nature of Statistical Learning Theory. Springer Verlag: New York.

[4] T. Joachims (2002). Learning to Classify Text Using Support Vector Machines. Dissertation, Kluwer.

[5] T. Joachims (1999). T. Joachims, 11 in: Making large-Scale SVM Learning Practical. Advances in Kernel Methods - Support Vector

Learning, B. Schoelkopf and C. Burges and A. Smola (ed.), MIT Press