INF264 - Exercise 5

Pierre Gillot Natacha Galmiche*

Week 39 - 2021

1 Linear SVM

In this first exercise, we get familiar with the linear SVM classifier on simple datasets. Answer the following questions:

1. Load and visualize the dataset contained in the file 'svm_data_1.csv'. Is it linearly separable ?
2. Train and evaluate the performance of a 'sklearn.svm.SVC' model on this first dataset, where the argument 'kernel' is set to 'linear'. You can set the argument 'C' to a very high value like 1E10 (explain why).
3. What is the value of the maximized margin you obtained ? **Hint:** Recall that the margin can be expressed as a function of the weights in the primal form of SVM.
4. In a 2D plot, display the training data, the support vectors, the decision boundaries, the decision hyperplane and the separating hyperplanes of your model. **Hint:** Since the data features are two-dimensional, hyperplanes are simply lines. You can display these lines by plotting the levels $\{-1, 0, 1\}$ of the 'decision_function' method of your model applied to a grid, using the 'matplotlib.pyplot.contour' function.
5. Load and visualize the dataset contained in the file 'svm_data_2.csv'. Is it linearly separable ?
6. Which argument of the 'sklearn.svm.SVC' model should you tune in order to learn this type of data ? What does this parameter control ?
7. Perform a model selection on the aforementioned parameter of the 'sklearn.svm.SVC' model, on the second dataset (remember to evaluate the performance of the best model). Plot the maximized margin as a function of the parameter and comment.
8. Same question as Question 4., this time on the best model that you selected via model selection on the second dataset.

2 Kernel trick

In order to fit an SVM on a dataset $X \subset \mathcal{X}$ that is non-linear, one can map the data to a higher dimensional space by applying on it a certain function $\phi : \mathcal{X} \mapsto \phi(\mathcal{X})$, called a feature map. SVM applied to the transformed non-linear data then involves evaluating the inner product of vectors from $\mathcal{X}$ mapped to the higher dimensional feature space $\phi(\mathcal{X})$, that is:

$$\langle \phi(x), \phi(y) \rangle, \text{ where } x, y \in \mathcal{X}. \tag{1}$$

Depending on the function ϕ, it may be a very inefficient strategy to explicitly map the vectors to the higher dimensional feature space $\phi(\mathcal{X})$. Indeed, the inner product $\langle \phi(x), \phi(y) \rangle$ may reduce to a certain

*Not a TA for this course this year.

function $K(x, y)$ that operates directly in the original (lower dimensional) space $\mathcal{X}$. Such a function K is called a kernel. Note that in practice, the feature map may be implicit and only the kernel is known. To verify that a function K is indeed a kernel, it suffices to identify the feature map ϕ such that $\forall (x, y) \in \mathcal{X} \times \mathcal{X}, K(x, y) = \langle \phi(x), \phi(y) \rangle$.

As an example, let us verify that the function $K_1 : (x, y) \in \mathbb{R}^n \times \mathbb{R}^n \mapsto x^t y + c$ where $c \geq 0$ is a kernel:

$$K_1(x, y) = x^t y + c = \sum_{i=0}^{n-1} x_i y_i + \sqrt{c}\sqrt{c} = \langle \phi_1(x), \phi_1(y) \rangle \tag{2}$$

where $\phi_1 : \ z \in \mathbb{R}^n \mapsto (z_0, \ldots, z_{n-1}, \sqrt{c}) \in \mathbb{R}^{n+1}$.

Let us now consider the function $K_2 : (x, y) \in \mathbb{R}^n \times \mathbb{R}^n \mapsto (x^t y + c)^2$ where $c \geq 0$. Answer the following questions:

1. Show by finding its implicit feature map $\phi_2 : \ \mathbb{R}^n \mapsto \phi_2(\mathbb{R}^n)$ that K_2 is a kernel. **Hint:** Remember that
$$\forall z \in \mathbb{R}^n, \ \left(\sum_{i=0}^{n-1} z_i \right)^2 = \sum_{i=0}^{n-1} z_i^2 + 2 \sum_{i=1}^{n-1} \sum_{j=0}^{i-1} z_i z_j. \tag{3}$$
2. What is the dimension of the higher dimensional feature space $\phi_2(\mathbb{R}^n)$?
3. In terms of complexity, what is the benefit of computing $K_2(x, y)$ as opposed to computing $\langle \phi_2(x), \phi_2(y) \rangle$ explicitly ?
4. Implement the two versions and verify in practice that computing $K_2(x, y)$ is more efficient.

3 Bagging

A bootstrap data sample is a sample of a dataset with replacement. Bootstrap aggregating is a method in machine learning that can be used in some of the high-variance machine learning algorithms such as decision tree in order to prevent overfitting.

The dataset we will use in this exercise is the Sonar dataset. This dataset describes sonar chirp returns bouncing off different surfaces. The 60 input variables are the strength of the returns at different angles. It is a binary classification problem with 208 observations that requires a model to differentiate rocks (labeled R) from metal (labeled M) cylinders.

There is a template available, you can choose whether you want to use it or not.

1. Load the sonar dataset and store the features in a matrix X and labels in a vector y.
2. Subsample function:
   (a) Write a 'subsample' function that returns random subsamples with replacement X_sample and y_sample from X and y. X_sample and y_sample must have the same length N as X and y but may contain duplicates. You can use the function 'random.choices()' to get a subsampling with replacement.
   (b) Call this function with X and y as parameters and count the ratio 'unique occurences of datapoints in X_sample'/'length of X_sample'. Compare this ratio to the one mentioned in the lecture slide on 'Bootstrap sampling'.
3. Bagging training function:
   (a) This function should take as input a training dataset X_train and y_train; a number of classifiers n_cfls; a classifier $Classifier$ (could be any sklearn class implementing a classifier).
   (b) This function should return a list of trained classifiers composing the bagging classifier.

4. Bagging predict function:
   (a) This function should take as input a list of trained classifiers $clfs$ composing a bagging classifier and a matrix of features X.
   (b) This function should return a vector of predictions made by the bagging classifier with the rule of majority vote.

5. Evaluation of bagging classifiers based on DecisionTrees:
   (a) Train and evaluate using cross validation a bagging classifiers based on n_clfs Decision Trees classifiers for different number of classifiers n_clfs.
   (b) Compare the performance on the test set of a bagging classifier consisting of a single Decision Tree (that is to say the bagging classifier is actually a decision tree) with a bagging classifier consisting of 30 Decision Trees.
   (c) What can we say about the variance and bias of a simple decision tree compared to a bagging classifier with 30 Decision Trees?
   (d) What are the disadvantages of a bagging classifier then?