CS3920/CS5920 Lab Worksheet 8: Neural nets and SVM

Volodymyr Vovk

November 18, 2022

This lab worksheet should be completed during the lab session of 21 November and your independent study time.

The topics that are briefly covered in this worksheet are:

- Neural networks.
- Maximum margin classifiers (= linear SVM).
- Kernel SVM.
- Multiclass classification with SVM (and other binary classification algorithms).

For further details of some functions used in this worksheet, see [1, Chapter 2] and [2].

If you are getting results that are not exactly the same as given below, usually there is no need to worry: you may be using a different version of `scikit-learn`.

1 Neural networks

We will build a neural net for a 2D classification dataset called `two_moons`. The dataset consists of two half-moon shapes, with each class consisting of equal numbers of samples. Simple neural networks, of the kind we are studying, are sometimes referred to as multi-layer perceptrons. This is why this model is called `MLPClassifier` in `scikit-learn`.

```
In[1]:
  from sklearn.neural_network import MLPClassifier
  from sklearn.datasets import make_moons
  from sklearn.model_selection import train_test_split
  import mglearn
  %matplotlib inline
  import matplotlib.pyplot as plt
  X,y = make_moons(n_samples=100, noise=0.25, random_state=0)
  X_train,X_test,y_train,y_test = train_test_split(X, y, random_state=0)
  mlp = MLPClassifier(solver='lbfgs', activation='tanh', random_state=0,
```

```
    hidden_layer_sizes=[10]).fit(X_train,y_train)
  mglearn.plots.plot_2d_separator(mlp, X_train, fill=True, alpha=.3)
  mglearn.discrete_scatter(X_train[:,0], X_train[:,1], y_train)
  plt.xlabel("Feature 0")
  plt.ylabel("Feature 1")
```

Now you can see the dataset and the decision boundary produced by our neural net, the `MLPClassifier`. The `mglearn` library is used for plotting only and can be ignored (together with the options used, whose only purpose is to make the picture look prettier).

We provided `MLPClassifier` with quite a few parameters, not relying on their default values.

- The available solvers (algorithms for fitting the parameters) are `lbfgs`, `sgd`, and `adam`, the default being `adam`. While `adam` works well on relatively large datasets (with thousands of training samples or more) in terms of both training time and predictive performance, for small datasets `lbfgs` can converge faster and perform better.
- By default, the MLP uses 100 hidden nodes, which is quite a lot for this small dataset. We reduce this number (which reduces the complexity of the model) and still get a good result. You can replace `[10]` by, e.g., `[10,10]`, which would give you two hidden layers of size 10 each.
- The default activation function is piecewise-linear, and we replace it by the smooth `tanh`.
- The classifier is randomized, and we set `random_state` for reproducibility.

2 Linear SVM

Linear support vector machines (linear SVMs) are implemented in the module `svm.LinearSVC` (SVC standing for "support vector classifier"). Let us apply `LinearSVC` to the `forge` dataset, which we used in Chapter 4, and visualize the decision boundary found by it.

```
In[2]:
  from sklearn.svm import LinearSVC
  X, y = mglearn.datasets.make_forge()
  svm = LinearSVC().fit(X, y)
  mglearn.plots.plot_2d_separator(svm, X, fill=False, eps=0.5, alpha=.7)
  mglearn.discrete_scatter(X[:,0], X[:,1], y)
  plt.xlabel("Feature 0")
  plt.ylabel("Feature 1")
```

The decision boundary found by `LinearSVC` is a straight line, separating the area classified as class 1 on the top from the area classified as class -1 on the bottom. In other words, any new sample that lies above the black line will be classified as class 1 by our classifier, while any sample that lies below the black line will be classified as class -1.

The model misclassifies two of the samples. By default, the model applies L_2 regularization, in the same way that `Ridge` does for regression.

In `scikit-learn` (as in Chapter 8), the trade-off parameter that determines the strength of the regularization is called `C`, and higher values of `C` correspond to less regularization. In other words, when you use a high value for the parameter `C`, `LinearSVC` tries to fit the training set as best as possible, while with low values of the parameter `C`, the model puts more emphasis on finding a coefficient vector (w) that is close to zero.

There is another interesting aspect of how the parameter `C` acts. Using low values of `C` will cause the algorithms to try to adjust to the majority of samples, while using a higher value of `C` stresses the importance that each individual sample be classified correctly. The library `mglearn` produces this nice picture:

```
In[3]:
  mglearn.plots.plot_linear_svc_regularization()
```

On the left-hand side, we have a very small `C` corresponding to a lot of regularization. Most of the samples in class -1 (shown as 0 in the legend) are at the bottom, and most of the samples in class 1 are at the top. The strongly regularized model chooses a relatively horizontal line, misclassifying two samples. In the centre plot, `C` is slightly higher, and the model focuses more on the two misclassified samples, tilting the decision boundary. Finally, on the right-hand side, the very high value of `C` in the model tilts the decision boundary a lot, now correctly classifying all samples in class -1. One of the samples in class 1 is still misclassified, as it is not possible to correctly classify all samples in this dataset using a straight line. The model illustrated on the right-hand side tries hard to correctly classify all samples, but might not capture the overall layout of the classes well. In other words, this model is likely overfitting.

3 Kernel SVM

The kernel SVM is also implemented in the `scikit-learn` module `svm`. Let's see how this classifier with radial kernel can be applied to the Breast Cancer dataset. By default, `C=1` and `gamma=1/n_features`:

```
In[4]:
  from sklearn.svm import SVC
  from sklearn.datasets import load_breast_cancer
  cancer = load_breast_cancer()
  X_train, X_test, y_train, y_test = train_test_split(cancer.data,
    cancer.target, random_state=0)
  svc = SVC()
  svc.fit(X_train, y_train)
  print("Accuracy on training set:", svc.score(X_train, y_train))
  print("Accuracy on test set:", svc.score(X_test, y_test))

  Accuracy on training set: 0.903756
  Accuracy on test set: 0.937063
```

The model does not perform very well, and for other random states it may overfit quite substantially. While SVMs often perform quite well, they are very sensitive to the settings of the parameters and to the scaling of the data. In particular, they require all the features to vary on a similar scale. But in this case the features are on vastly different scales! (Have a look at them.) This would be somewhat of a problem for other models (like linear models), but it often has devastating effects for the kernel SVM.

Let us now normalize the data.

```
In[5]:
  from sklearn.preprocessing import MinMaxScaler
  scaler = MinMaxScaler()
  scaler.fit(X_train)
  X_train_scaled = scaler.transform(X_train)
  X_test_scaled = scaler.transform(X_test)
  svc.fit(X_train_scaled, y_train)
  print("Accuracy on training set:", svc.score(X_train_scaled, y_train))
  print("Accuracy on test set:", svc.score(X_test_scaled, y_test))

  Accuracy on training set: 0.983568
  Accuracy on test set: 0.972028
```

Normalizing the data made a significant difference. From here, we can try either increasing `C` or decreasing `gamma` to fit a more complex model. For example:

```
In[6]:
  svc = SVC(C=1000)
  svc.fit(X_train_scaled, y_train)
  print("Accuracy on training set:", svc.score(X_train_scaled, y_train))
  print("Accuracy on test set:", svc.score(X_test_scaled, y_test))

  Accuracy on training set: 1.0
  Accuracy on test set: 0.958042
```

However, increasing `C` leads to overfitting; now we have perfect performance on the training set but poorer performance on the test set.

4 Multiclass classification

Let's apply the one-vs-rest method to a simple three-class classification dataset. The dataset is 2D and each class is given by data sampled from a Gaussian distribution. We use an `mglearn` function for visualizing it:

```
In[7]:
  from sklearn.datasets import make_blobs
  X, y = make_blobs(random_state=42)
  mglearn.discrete_scatter(X[:,0], X[:,1], y)
  plt.xlabel("Feature 0")
  plt.ylabel("Feature 1")
  plt.legend(["Class 0", "Class 1", "Class 2"])
```

We would like to use `LinearSVC` as our base method for binary classification. However, this function in `scikit-learn` already incorporates the one-vs-rest method! To see this, let us train a `LinearSVC` classifier on our dataset:

```
In[8]:
  svm = LinearSVC().fit(X, y)
  print("Coefficient shape:", svm.coef_.shape)
  print("Intercept shape:", svm.intercept_.shape)

  Coefficient shape: (3, 2)
  Intercept shape: (3,)
```

We see that the shape of `coef_` is $(3, 2)$; each row of the matrix `coef_` contains the coefficient vector w for one of the three classes and each column holds the coefficient value for a specific feature (there are two in this dataset). The vector `intercept_` stores the intercepts b for each class. (I am using the notation w and b as in Chapter 8; see, e.g., the bottom of slide 47.)

Let's visualize the lines given by the three binary classifiers:

```
In[9]:
  mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
  line = np.linspace(-15, 15)
  for coef, intercept, color in zip(svm.coef_,
      svm.intercept_, ['b', 'r', 'g']):
    plt.plot(line, -(line * coef[0] + intercept) / coef[1], c=color)
  plt.ylim(-10, 15)
  plt.xlim(-10, 8)
  plt.xlabel("Feature 0")
  plt.ylabel("Feature 1")
  plt.legend(['Class 0', 'Class 1', 'Class 2', 'Line class 0',
    'Line class 1', 'Line class 2'], loc=(1.01, 0.3))
```

(If you are getting a syntactic error, correct it.)

The command `mglearn.discrete_scatter` in `In[9]` produces a scatterplot, as in `In[7]`; please try to understand what the other commands are doing (see the exercises below).

You can see that all the samples belonging to class 0 in the training data are above the line corresponding to class 0, which means they are on the "class 0" side of this binary classifier. The samples in class 0 are above the line corresponding to class 2, which means they are classified as "rest" by the binary classifier for class 2. The samples belonging to class 0 are to the left of the line corresponding to class 1, which means the binary classifier for class 1 also classifies them as "rest". Therefore, any sample in this area will be classified as class 0 by the final classifier (the value of the scoring function, reflecting confidence, for classifier 0 is greater than zero, while it is smaller than zero for the other two classes).

But what about the triangle in the middle of the plot? All three binary classifiers classify samples there as "rest". Which class would a sample there be assigned to? The answer is the one with the highest value for the scoring function: the class of the closest line.

The following code produces the predictions for all regions of the 2D space:

```
In[10]:
  mglearn.plots.plot_2d_classification(svm, X, fill=True, alpha=.7)
  mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
  line = np.linspace(-15, 15)
  for coef, intercept, color in zip(svm.coef_,
      svm.intercept_, ['b', 'r', 'g']):
    plt.plot(line, -(line * coef[0] + intercept) / coef[1], c=color)
  plt.legend(['Class 0', 'Class 1', 'Class 2', 'Line class 0',
    'Line class 1', 'Line class 2'], loc=(1.01, 0.3))
  plt.xlabel("Feature 0")
  plt.ylabel("Feature 1")
```

5 Exercises

Write all your answers in your Jupyter notebook.

1. Print out the array `svm.coef_`, as in `In[8]`. Notice that the signs of the two numbers in rows 0 and 1 are opposite, while the signs of the two numbers in row 2 coincide. Explain briefly why.

2. Explain the role of the function `zip` in `In[9]` and `In[10]`.

3. Explain the meaning of the formula

   ```
   -(line * coef[0] + intercept) / coef[1]
   ```

   in `In[9]` and `In[10]`.

References

[1] Andreas C. Müller and Sarah Guido. *An introduction to machine learning with Python*. O'Reilly, Beijing, 2017.

[2] `scikit-learn` tutorials. `http://scikit-learn.org/stable/tutorial/`, 2007–2022.