Model Zoo

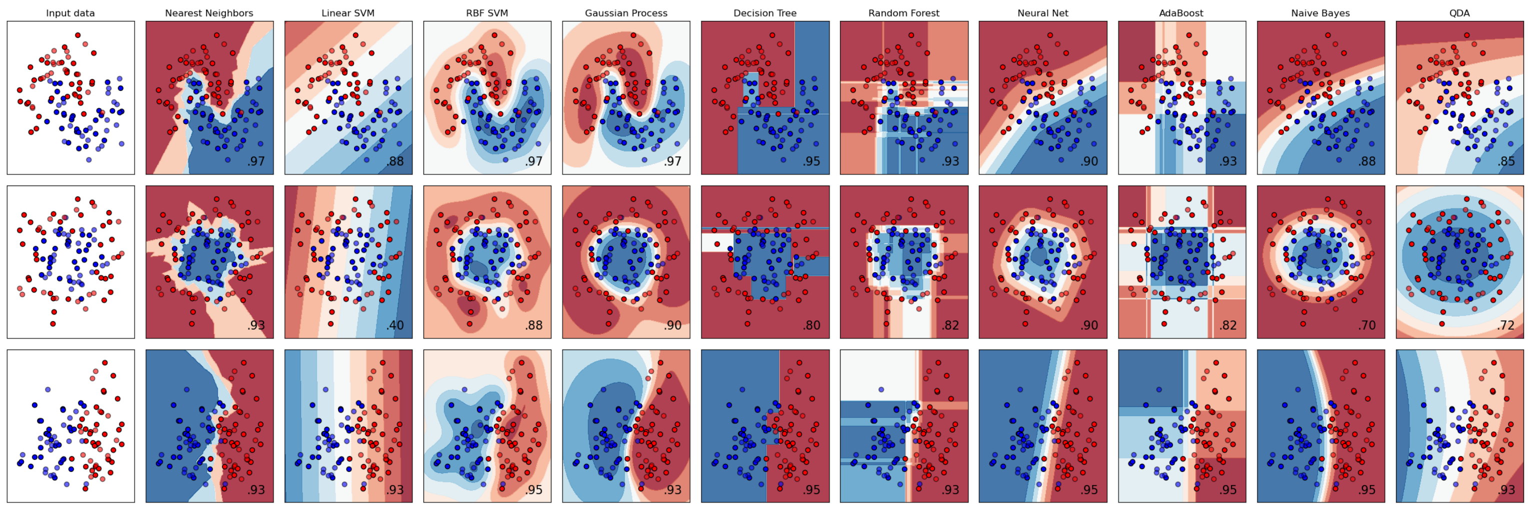
1 Introduction

Scikit-learn implements many different classification models. The VC dimension gives us a generic tool for understanding the generalization performance of all of these models. These notes will give a high level overview of the most important variants of these models and their VC dimension.

None of this information is in the textbook. Citations are provided for all the information, but you're not expected to look at the original sources. There are two goals for including this material: (1) so that you can use this material to determine which models to use in practice; (2) so that you get a sense of what the state of the art in machine learning theory is.

There's one last section for us to cover in the textbook (4.3.2 model selection), but it makes sense for us to have a large set of example models to think about before covering this section.

2 Models



Source: `https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html`

2.1 Linear/Quadratic Models

Recall that a model refers to a hypothesis class and a training algorithm. All linear models use the hypothesis class of halfspaces, and all quadratic models use halfspaces with a 2nd degree polynomial kernel. Scikit learn implements the following models.

1. Linear classifiers:

   ```
   sklearn.discriminant_analysis.LinearDiscriminantAnalysis
   sklearn.linear_model.LogisticRegression
   sklearn.linear_model.Perceptron
   sklearn.linear_model.SGDClassifier
   sklearn.linear_model.PassiveAggressiveClassifier
   sklearn.svm.LinearSVC
   ```

2. Quadratic classifiers:

   ```
   sklearn.discriminant_analysis.QuadraticDiscriminantAnalysis
   sklearn.naive_bayes.GaussianNB
   ```

The only difference between these models is the training algorithm.

Note 1. You should at this point understand most of the hyperparameters to `sklearn.linear_model.LogisticRegression` and `sklearn.linear_model.SGDClassifier`. In particular, you should understand the effects of

1. `penalty`
2. `tol`
3. `C`
4. `fit_intercept`
5. `intercept_scaling`
6. `max_iter`
7. `l1_ratio`
8. `loss` (SGD only)

Problem 1. A colleague is using quadratic discriminant analysis to solve a machine learning problem. Provide simple suggestions on how they could improve classification accuracy in the following situations.

1. They are overfitting.

2. The training error E_{in} is large.

2.2 Multi Layer Perceptron (MLP)

Recall that the class of halfspaces is

$$\mathcal{H} = \left\{ \mathbf{x} \mapsto \text{sign}(\mathbf{w}^T \mathbf{x}) : \mathbf{w} : \mathbb{R}^{d'} \right\} \tag{1}$$

and it is common to adjust the VC-dimension of the problem by applying a feature map $\Phi : \mathbb{R}^d \to \mathbb{R}^{d'}$ to get the hypothesis class

$$\mathcal{H} = \left\{ \mathbf{x} \mapsto \text{sign}(\mathbf{w}^T \Phi(\mathbf{x})) : \mathbf{w} : \mathbb{R}^{d'} \right\}. \tag{2}$$

In logistic regression, we fix the Φ function in advance and learn only the $\mathbf{w}$ vector.

The idea of the MLP is to also learn the Φ function from the data. In a 1-layer MLP, we set

$$\Phi(\mathbf{x}) = \sigma(W_1 \mathbf{x}) \tag{3}$$

where $W_1 : \mathbb{R}^{d' \times d}$, and $\sigma : \mathbb{R} \to \mathbb{R}$ is called the *activation function* and is applied element-wise to $W_1\mathbf{x}$. The W_1 matrix is learned from the data (typically via stochastic gradient descent) at the same time as the $\mathbf{w}$ vector. In the i-layer perceptron, the Φ function is recursively defined to be

$$\Phi^{(i)}(\mathbf{x}) = \begin{cases} \sigma(W_1 \mathbf{x}) & \text{if } i = 1 \\ \sigma(W_i \Phi^{(i-1)}(\mathbf{x})) & \text{otherwise} \end{cases} \tag{4}$$

where each matrix $W_i : \mathbb{R}^{d_i \times d_{i-1}}$ and $d_0 = d$ and $d_i = d'$. The value i is called the *depth* of the neural network or the *number of layers* of the network, and the value of d_i is called the width of the ith layer.

Note 2. The MLP is a special case of a *neural network*. TensorFlow/PyTorch are libraries from Google/-Facebook for implementing neural networks that give much more control than scikit-learn. TensorFlow has a good visualization of MLPs at: `https://playground.tensorflow.org`

Note 3. You are responsible for understanding how the following parameters affect scikit learn's `sklearn.neural_network.MLPClassifier`.

1. `hidden_layer_size`
2. `activation`
3. `solver`
4. `alpha`

Theorem 1 (universal approximation)**.** Let g be the ERM hypothesis for the 1-layer MLP hypothesis class. Then,

$$\lim_{d_1 \to \infty} E_{\text{in}}(g) = 0. \tag{5}$$

Theorem 2. Define

$$E = \prod_{i=1}^{k} d_i d_{i-1} = d' d \prod_{i=1}^{k-1} d_i^2. \tag{6}$$

$$V = \prod_{i=1}^{k} d_i \tag{7}$$

The VC-dimension of the k-layer MLP with the identity activation function is

$$\Theta(d). \tag{8}$$

The VC-dimension of the k-layer MLP with the $\sigma = \text{sign}$ activation function is

$$d_{\text{VC}} = O(E \log E). \tag{9}$$

With the $\sigma = \tanh$ activation function, the tightest known bounds on the VC dimension are

$$d_{\text{VC}} = \Omega(E^2) \quad \text{and} \quad d_{\text{VC}} = O(V^2 E^2). \tag{10}$$

Proof. See Section 20.4 of *Understanding Machine Learning: From Theory to Algorithms.* □

Theorem 3. Let E be defined as in Equation (6) above. Then the VC dimension of neural networks with the ReLU (hinge loss) activation function is upper bounded by

$$d_{\text{VC}} = O(Ek \log(E)) \tag{11}$$

and lower bounded by

$$d_{\text{VC}} = \Omega(Ek \log(E/k)). \tag{12}$$

Proof. See "Nearly-tight VC-dimension and pseudodimension bounds for piecewise linear neural networks", COLT 2017. □

Fact 1. The optimization problem for neural networks is non-convex. Optimization is therefore not guaranteed to lead to a global minimum.

Problem 2. You are training a `sklearn.neural_network.MLPClassifier` model with the hyperparameter `activation` set equal to `relu` on 1000 dimensional data. You have the choice of setting the `hidden_layer_sizes` hyperparameter to either (a) `[10,10,10]` or (b) `[100]`.

1. Which choice is more likely to overfit the data?

2. Which choice will likely require a larger value of `alpha`?

Problem 3. You have successfully trained a `sklearn.neural_network.MLPClassifier` model with `hidden_layer_sizes` equal to `[100,100]` in order to predict whether a user will click on an advertisement. The model is neither overfitting nor underfitting the data, and so you have successfully tuned all the hyperparameters of this model into their optimal configuration.

Your colleagues in the marketing department have collected more data, however, and so are requesting a new model be trained that is more accurate. The new dataset is twice the size of the old dataset.

1. Your colleagues suggest that since you have twice the amount of data, you could make the neural network twice as deep and use `hidden_layer_sizes` equal to `[100,100,100,100]`. Does VC theory predict this will lead to good generalization?

2. Different colleagues suggest that since you have twice the amount of data, you could make the neural network twice as wide and use `hidden_layer_sizes` equal to `[200,200]`. Does VC theory predict this will lead to good generalization?