Model Zoo

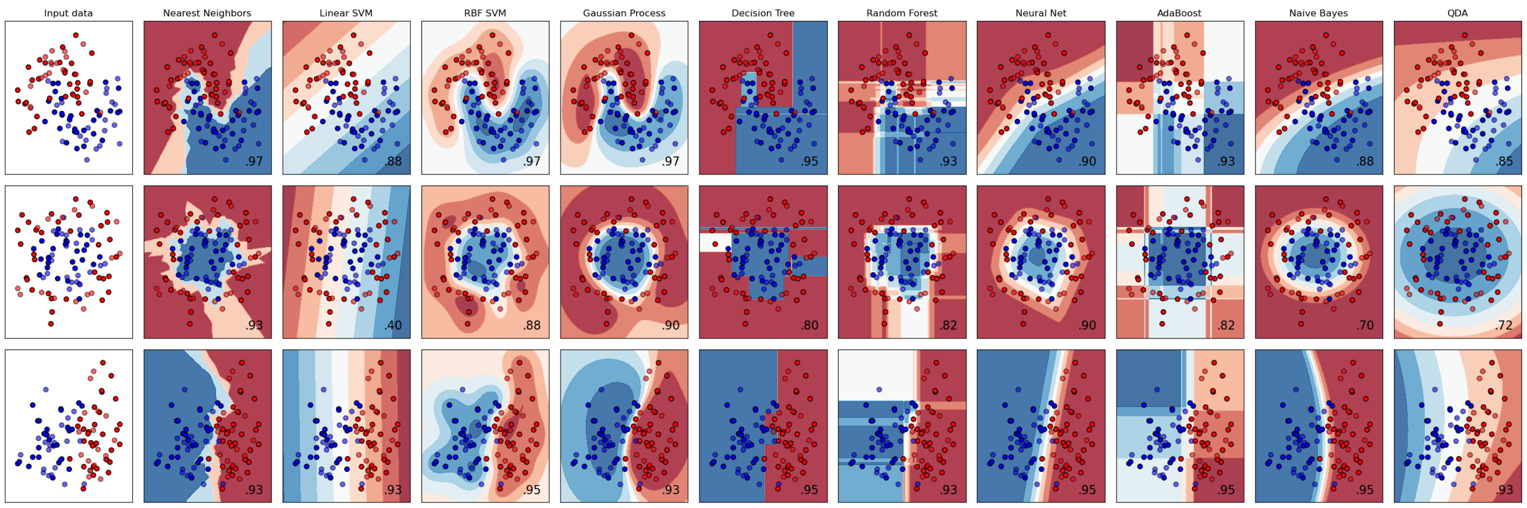
1 Introduction

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. Therefore, the VC dimension of all linear models is $\Theta(d)$ and all quadratic models is $\Theta(d^2)$. This immediately implies that the generalization error is

$$E_{\text{out}} = E_{\text{in}} + O\left(\sqrt{\frac{d \log N}{N}}\right) \tag{1}$$

for linear models and

$$E_{\text{out}} = E_{\text{in}} + O\left(\sqrt{\frac{d^2 \log N}{N}}\right) \tag{2}$$

for quadratic ones.

The only difference between these models and logistic regression is the training algorithm. Models like `sklearn.discriminant_analysis.LinearDiscriminantAnalysis` and `class sklearn.naive_bayes.GaussianNB` make parametric assumptions on the underlying data and minimize E_{in} under those assumptions. Logistic regression and SVMs are able to minimize E_{in} without any assumptions, however, so they "usually" have better training error.

In practice, practitioners almost always use logistic regression or SVMs to optimize linear/quadratic hypothesis spaces because the training error is smaller and the generalization error is the same.

Note 1. A common interview question is to explain the assumptions behind some of these "bad" models like naive bayes or linear discriminant analysis. I won't ask you to do that for this class.

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 binary halfspaces is

$$\mathcal{H} = \left\{ \mathbf{x} \mapsto \text{sign}(\mathbf{w}^T\mathbf{x}) : \mathbf{w} : \mathbb{R}^{d'} \right\} \tag{3}$$

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

$$\mathcal{H} = \left\{ \mathbf{x} \mapsto \text{sign}(\mathbf{w}^T\Phi(\mathbf{x})) : \mathbf{w} : \mathbb{R}^{d'} \right\}. \tag{4}$$

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

$$\Phi(\mathbf{x}) = \sigma(W_1\mathbf{x}) \tag{5}$$

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

$$\Phi^{(i)}(\mathbf{x}) = \begin{cases} \sigma(A_1\mathbf{x}) & \text{if } i = 1 \\ \sigma(A_i\Phi^{(i-1)}(\mathbf{x})) & \text{otherwise} \end{cases} \tag{6}$$

where each matrix $A_i : \mathbb{R}^{d_i \times d_{i-1}}$ 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`

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

$$\lim_{d_1 \to \infty} E_{\text{in}}(g) = 0. \tag{7}$$

Theorem 2. Define

$$E = \prod_{i=1}^{k} d_i d_{i-1} = d' d \prod_{i=1}^{k-1} d_i^2. \tag{8}$$

$$V = \prod_{i=1}^{k} d_i \tag{9}$$

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

$$\Theta(d). \tag{10}$$

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

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

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

$$d_{\text{VC}} = \Omega(E^2) \quad \text{and} \quad d_{\text{VC}} = O(V^2 E^2). \tag{12}$$

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

Theorem 3. Let E be defined as in Equation (8) above. Then the VC dimension of neural networks with the ReLU (hinge loss) activation function is upper bounded by

$$d_{\text{VC}} = O(Ek \log(E)) \tag{13}$$

and lower bounded by

$$d_{\text{VC}} = \Omega(Ek \log(E/k)). \tag{14}$$

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 convex only when the activation function σ is the identity function. 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?

Decision Trees

Describe the decision tree as implemented in `sklearn.tree.DecisionTreeClassifier`.

Fact 2. Decision trees are commonly used in medical applications. It is easy for doctors to both interpret the resulting model, and to memorize the model and apply it manually in clinical situations.

Theorem 4 (universal approximation)**.** Let g be the ERM hypothesis for the class of binary decision trees with k nodes. Then,

$$\lim_{k\to\infty} E_{\text{in}}(g) = 0. \tag{15}$$

Theorem 5. Let k be the number of nodes in a binary decision tree. Then the VC-dimension is bounded by

$$d_{\text{VC}} = O(k\log(kd)). \tag{16}$$

Proof. See "Decision trees as partitioning machines to characterize their generalization properties," NeurIPS 2020. □

Note 3. This result directly contradicts the advice given in scikit-learn's "Tips for Practical Use":
`https://scikit-learn.org/stable/modules/tree.html#tips-on-practical-use`.

Corollary 1. Let j be the height of a binary decision tree. Then the VC dimension is bounded by

$$d_{\text{VC}} = O(2^j j\log d). \tag{17}$$

Problem 4. Describe how changes to the following hyperparameters to `sklearn.tree.DecisionTreeClassifier` affect the VC dimension (increase, decrease, stays the same).

1. `criterion`

2. `max_depth`

3. `min_samples_split`

4. `min_samples_leaf`

5. `max_features`

6. `random_state`

7. `max_leaf_nodes`

Problem 5. You have a small dataset ($N = 10^3$) with a large number of features ($d = 10^6$). Would you rather use L2 normalized logistic regression for this problem or decision trees? Why?

Problem 6. If you double the height of a decision tree from 3 to 6, approximately how much more data do you need to achieve the same generalization error?

Problem 7. What is the VC dimension of a decision tree with k nodes where the polynomial feature map of degree p has been applied to the data?

Ensemble Methods

The hypothesis class of ensemble methods is

$$L(B, T) = \left\{ \mathbf{x} \mapsto \operatorname{sign}\left(\sum_{t=1}^{T} w_t h_t(\mathbf{x}) \right) : \mathbf{w} \in \mathbb{R}^T, h_t \in B \right\} \tag{18}$$

where B is a set of "base" hypothesis classes and $T : \mathbb{Z}$ is the number of hypotheses from B to combine.

Theorem 6 (universal approximation)**.** Let g be the ERM hypothesis for $L(B, T)$. Then

$$\lim_{T \to \infty} E_{\text{in}}(g) = 0 \tag{19}$$

for any hypothesis class B that is a

Definition 1. weak learner. A *weak learner* is any hypothesis class capable of achieving better than random error.

Lemma 1. The VC-dimension of $L(B, T)$ is

$$d_{\text{VC}}(L(B, T)) = O(T d_{\text{VC}}(B) \log(T d_{\text{VC}}(B))) \tag{20}$$

Proof. See Chapter 10, Lemma 10.3 of *Understanding Machine Learning: From Theory to Algorithms.* □

Problem 8. Decision trees are some of the most commonly "boosted" models.

1. Provide a tight upper bound on the VC dimension for an ensemble of decision trees.

2. If you increase the number of decision trees T in the ensemble, then how should you adjust the number of nodes k in the decision trees?

3. If you increase the number of nodes k in the base decision trees, then how should you adjust the number of decision trees in the ensemble T?

Nearest Neighbor Methods

Fact 3. The VC-dimension of k-nearest neighbor methods is infinite.

Theorem 7 (informal)**.** Let h be the 1-NN hypothesis. Then for "well behaved" data distributions,

$$E_{\text{out}}(h) \leq 2E_{\text{out}}(f) + 4\sqrt{d}N^{-\frac{1}{d+1}}. \tag{21}$$

Proof. See Theorem 19.3 of *Understanding Machine Learning: From Theory to Algorithms.* □

3 Multiclass Classification

In the multiclass setting, we let let c be the number of classes, and the output space $\mathcal{Y} = [c]$.

Fact 4 (informal)**.** The generalization error of a multiclass classification problem is $\sqrt{c}$ times the generalization error of the equivalent binary classification problem. That is, with high probability

$$E_{\text{out}}(g) \leq E_{\text{in}}(g) + O\left(\sqrt{\frac{c \cdot d_{\text{VC}} \cdot \log N}{N}}\right). \tag{22}$$

Technically, this result doesn't hold under certain edge cases. The *Natarajan dimension* is a formal generalization of the VC dimension for the multiclass setting, and this generalization bounds based on the Natarajan dimension are the most technically correct way to talk about the generalization error of multiclass classification. Unfortunately, the Natarajan dimension is more mathematically sophisticated than the VC dimension. Fortunately, appealing to the Natarajan dimension is not needed in practice. For real-world purposes, Eq 22 is useful for decision making, and it's what you should use to complete the following problems.

Problem 9. You have an image classification problem where you are determining whether an image contains a dog or not. You have found that logistic regression with the 2nd degree polynomial kernel provides a good balance between training and generalization error for your dataset of $d = 100$ and $N = 1000$. You are considering extending your classifier to predict the breed of dog, where there would be 80 different breeds to select from. How many training data points would you expect to need in order to get similar E_{out}?