

1 Learning a Spam Classifier

In assignment 4, you are to write a program that learns a spam classifier. Your program will input a set of emails labeled as spam, and a set of emails labeled as legitimate. Your program will begin by performing feature selection to convert each email message into a vector of feature values. I will denote the feature vector representing email i as \mathbf{x}_i with label y_i . Then, your program will learn a *classifier*. A classifier is a function which inputs any valid vector of feature values, and outputs a classification. In this case, the classification will be 1 if the classifier believes the input represents a spam email and 0 if it believes the input represents a legitimate email. I will represent the learned classifier by the function $f(\cdot)$ which inputs a feature vector \mathbf{x}_i and outputs a guess of \mathbf{x}_i 's label, \hat{y}_i . For any input \mathbf{x}_i , we want $f(\mathbf{x}_i) = \hat{y}_i = y_i$.

For example, suppose you decide that the only relevant features for classifying an email as spam or legitimate are the number of times that each of the words “diploma” and “guarantee” appear in the email. Then, your program would convert each input email message into a vector of length 2. The first element of this vector would be the number of times the word “diploma” appears in the given email. The second element of the vector would be the number of times the word “guarantee” appears in the given email. Suppose you do this conversion for each input message. Figure 1(a) shows an example plot of the 2D training examples. The x’s represent spam emails, and the o’s represent legitimate emails. Qualitatively, in this synthetic example, it seems that either the word “diploma” or the word “guarantee” can appear in a legitimate email, but if *both* words appear, then it is likely the email is spam. Once your program has converted the emails into feature vectors, it must learn a classifier. Suppose, for example, that the classifier shown in Figure 1(b) is learned. For any valid input feature vector, the learned classifier will classify its input as spam if it falls in the blue region of input space, and legitimate if it falls in the purple region of input space.

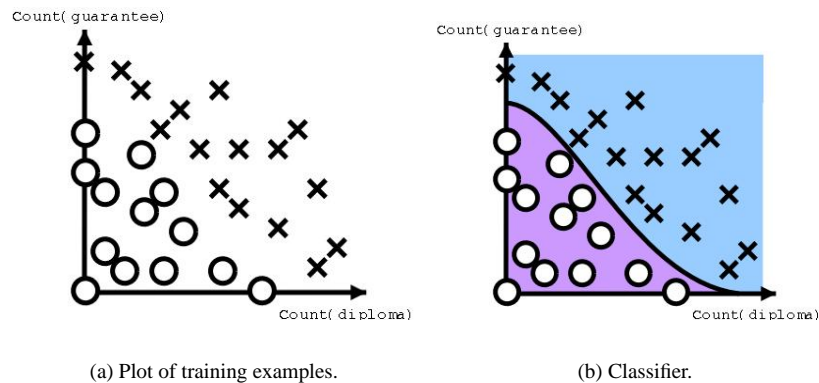


Figure 1: An example training data set and learned classifier. Input email examples are converted into 2-D feature vectors. The x -axis is the number of times the word “diploma” appears, and the y -axis is the number of times the word “guarantee” appears. The x’s represent spam emails and the o’s represent legitimate emails. An example classifier is shown in (b). Any input vector in the blue region is classified as spam, and any input vector in the purple region is classified as legitimate.

The hardest part of designing a program to learn a classifier is to make it perform well on new test data.

2 Naive Bayes

We are going to view classification in a Bayesian framework. In this, our goal is to estimate the probability of each label y_i for any input \mathbf{x}_i , $P(Y = y_i | \mathbf{X} = \mathbf{x}_i)$. Here, \mathbf{X} is the vector of random variables representing the input and Y is the random variable representing the label. Our classifier chooses the label with the highest probability:

$$f(\mathbf{x}_i) = \hat{y}_i = \operatorname{argmax}_y P(Y = y | \mathbf{X} = \mathbf{x}_i).$$

We use Bayes’ rule to rewrite this probability as:

$$P(Y = y | \mathbf{X} = \mathbf{x}_i) = \frac{P(\mathbf{X} = \mathbf{x}_i | Y = y)P(Y = y)}{P(\mathbf{X} = \mathbf{x}_i)}.$$

Since \mathbf{x}_i is fixed, which y maximizes this probability does not depend on $P(\mathbf{X} = \mathbf{x}_i)$. So, we can ignore this term:

$$P(Y = y | \mathbf{X} = \mathbf{x}_i) \propto P(\mathbf{X} = \mathbf{x}_i | Y = y)P(Y = y).$$

So, to find y which maximizes $P(Y = y|\mathbf{X} = \mathbf{x}_i)$, we estimate $P(\mathbf{X} = \mathbf{x}_i|Y = y)$ and $P(Y = y)$ for each label y and the input feature vector \mathbf{x} .

To estimate $P(Y = y)$ using the training data, we estimate the frequencies in the training data. We count the number of training examples with label y , for each y . We approximate

$$P(Y = y) = \frac{\text{\# of examples with label } = y}{\text{total \# of examples}}.$$

The difficult part is to estimate $P(\mathbf{X} = \mathbf{x}_i|Y = y)$, which is called the likelihood, particularly when \mathbf{X} is high dimensional. To make this simpler, we will make the *naive* assumption that the individual features of \mathbf{X} are conditionally independent, given the label. Mathematically, this means that:

$$P(\mathbf{X} = \mathbf{x}_i|Y = y) = \prod_{j=1}^d P(X_j = x_{ij}|Y = y).$$

For example, we are assuming that if an input is spam, then the number of times the word “diploma” appears does not depend on the number of times the word “guarantee” appears. This is a reasonable assumption for our synthetic data set, since the x ’s are distributed fairly evenly in the blue region. Thus, we are *not* assuming that count(diploma) and count(guarantee) are independent. Instead, we are assuming that for all spam emails, count(diploma) and count(guarantee) are independent. Similarly, we are assuming for all legitimate emails, count(diploma) and count(guarantee) are independent.

So, to estimate $P(\mathbf{X} = \mathbf{x}_i|Y = y)$, we must estimate $P(X_j = x_{ij}|Y = y)$ for each feature j . To do this, we again estimate the frequencies in the training data. We count the number of times the j^{th} feature is equal to x_{ij} in all training examples with label y . We approximate:

$$P(X_j = x_{ij}|Y = y) = \frac{\text{\# examples with feature } j = x_{ij} \text{ and label } y}{\text{total \# of examples with label } y}.$$

Here is the Naive Bayes algorithm for learning a classifier:

```

Learned_Counts = NB_Learner({(x_i, y_i)}_{i=1}^N)
1) For each label y, feature j, and feature value x_j:
   Initialize Feature_Count[y, j, x_j] ← 0.
2) For each label y:
   Initialize Label_Count[y] ← 0.
3) For i = 1, ..., N:
   3-1) Label_Count[y_i]++.
   3-2) For j = 1, ..., d: Feature_Count[y_i, j, x_{ij}]++.
4) Return Learned_Counts ← Feature_Count, Label_Count.

```

Here is the classifier learned:

```

ŷ = f(x, Learned_Counts)
1) For each label y, compute:
   l_y ← -(d - 1) log Label_Count[y] + ∑_{j=1}^d log Feature_Count[y, j, x_j].
2) Return argmax_y l_y.

```

Where did the logs in step 1) of the classifier come from? We want to compare

$$p_y = \text{LabelCount}[y]^{-(d-1)} \prod_{j=1}^d \text{FeatureCount}[y, j, x_j].$$

Actually performing this multiplication will result in floating point overflow if there are a large number of features. We thus compute and compare the log of p_y for each y . Comparing the logs is valid because the log function is a monotonically increasing function. So, $a < b$ if and only if $\log a < \log b$. Computing the logs can prevent underflow because the log of a product is the sum of logs: $\log(ab) = \log a + \log b$. We therefore modified our classification algorithm to compute and compare $\log p_y = l_y$.

Why didn’t we include the “total # of examples” term in the calculation? Because we are only searching for which label maximizes the criterion, not the value of the criterion itself. We will be off by a constant factor for each label y .

What if none of the training instances with target value y have feature $j = x_j$? Then $\text{Feature.Count}[y, j, x_j] = 0$, so our approximation of $P(X_j = x_j|Y = y) = 0$. This means that $p_y = 0$ for any input example with feature $j = x_j$. The typical solution is to add a small constant number of “virtual” examples for each feature value.

$$P(X_j = x_{ij}|Y = y) = \frac{(\text{\# examples with feature } j = x_{ij} \text{ and label } y) + c}{(\text{total \# of examples with label } y) + c(\text{\# of possible values for feature } j)}.$$