3.20

Intro

In this exercise we will compare the NBC with the full model (without any assumptions about the distribution) for class conditionals over binary data. It is expected for the full model to be more complex and accurate, because it does not make any assumptions about the distribution. However, due to its complexity, we need a huge amount of data to this expectation to hold true and the model do not overfit. Usually, the size of the dataset needed for the full model to converge is so big, that becomes unfeasible to get it in real situations. In this situation, the NBC gives better results. This happens because the model makes reasonable assumptions about the data. This assumptios led to a simpler model with fewer parameters, which will not overtfit the data. A more general view of this problem is explored in the chapter dealing with probabilistic graphic models.

As a final intro comment: since the prior is uniform, the classification depends only on the class conditional $p(x|y)$.

Solution

a)

For the full model, there is no clever trick to simplify the represantation. One could use the chain rule of probability:

$$p(x_{1:D}|y=c) = p(x_1|y=c)p(x_2|x_1, y=c)...p(x_D|x_{1:D-1}, y=c) \qquad (1)$$

to represent the full model, but this would not give any advantages over the common joint probability represantation. A good representation for the class conditional is a set of C tables which are indexed by the binary feature vector x and hold the values for $p(x|y=c)$. Thus, to get a value like $= p(x|c=2)$, we would go to table 2, convert the binary feture vector to an array index and lookup the result. The conversion between the array and the binary vector is given by $array_{idx}(x) = \sum x_i 2^i$.

Since, we don't have any simplifications each of the C tables have 2^D indexes, which corresponds to all possible values of the binary feature. Since we are bound by $\sum_x p(x|c) = 1$, one of the tables entries is not independent. Therefore, the number of features is equal to: $(2^D - 1)C = O(C2^D)$

b)

For N very small, the naive Bayes model would outperform the full model. This would happen because the full model has too many parameters and would overfit on a small dataset. On the other hand, as commented on the chapter, the naive Bayes is immune to overfit since it has only $O(CD)$ parameters.

c)

For N very large, the opposite happens. The strong hypothesis of the naive Bayes would make the model underfit. On the other hand, since there is no hypothesis constraining the full model and we have enough data, this would output a very low teset error.

d)

Naive Bayes:

runtime complexity: $O(ND)$ (the training algorithm is presented in the chapter)

Full model:

Algorithm 1 Full model training

procedure TRAINING($Dataset$)
  $p_{idx,c} = 0, N_c = 0$
  for $i = 1 : N$ **do**
    $c = y_i$
    $N_c = N_c + 1$
    $idx = binaryToIdx(x_i)$
    $p_{idx,c} = p_{idx,c} + 1$
  end for
  $p_{idx,c} = \frac{p_{idx,c}}{N_c}$
  Return $p_{idx,c}$
end procedure

Since the author says that we can convert a D-bit vector to a array index in $O(D)$ time, the runtime complexity is given by $O(ND)$. Note that we did not estimate the prior because the author informed that the class prior for this problem is uniform.

e)

Naive Bayes:

runtime complexity: $O(CD)$ (the test time algorithm is presented in the chapter)

Full model:

Algorithm 2 Full model testing

procedure TESTING($Sample$)
  $idx = binaryToIdx(x)$
  $bestCandidate = 0$
  $class = -1$
  for $c = 1 : C$ **do**
    if $bestCandidate < p_{idx,c}$ **then**
      $bestCandidate = p_{idx,c}$
      $class = c$
    end if
  end for
  Return $bestCandidate, class$
end procedure

This algorithm runs in: $O(D)+O(C) = O(max(C, D))$. Usually, the number of features is bigger than the number of classes, so the runtime complexity is equal to $O(D)$. In another words, what bounds the runtime of the algorithm is the time for converting the binary array to the table index.

f)

Naive Bayes:

In the Naive Bayes model, the class conditional features are independent. Therefore, we can calculate $p(x_v|y)$ and $p(x_v) = \sum_{y=1}^{c} p(y)p(x_v|y)$, ignoring the missing features. Nevertheless, we will need to run through all features at preprocessing or runtime to determine which features are missing. The following algorithm filters the hidden features at runtime.

Algorithm 3 Naive Bayes with missing features

procedure TESTINGWITHMISSINGFEATURES($incompleteSample$)
  for $c = 1 : C$ **do**
    $L_c = log(\pi_c)$
    for $j = 1 : D$ **do**
      if $x_j missing$ **then** *continue*
      end if
      if $x_j = 1$ **then**
        $L_c = L_c + log\theta_{jc}$
      else$L_c = L_c + log(1 - \theta_{jc})$
      end if
    end for
  end for
  $p_c = exp(L_c - logsumexp(L_:))$
  Return $argmax_c p_c$
end procedure

The runtime complexity of this new testing algorithm remains $O(CD)$

Full model:

For the full model, we can create a preprocessing step, to guarantee that every visisble feature is at the left of all hidden feautres. This will help us sweep the indexes on the probabilities tables. This pre-processing step requires one sweep through the feature array, which have runtime complexity $O(D)$. With the sorted feature array, we can precompute the index of the visible feature and use the hidden feature vector index as an offset of this base value. This will require linear time over all the combinations of the hidden feature.

Algorithm 4 Full model with missing features

procedure TESTINGWITHMISSINGFEATURES($incompleteSample$)
  $vIndex = arrayToIndex(x_v)$
  $p_{x_v|c} = 0, p(x_v) = 0$
  for $offset = 0 : 2^h - 1$ **do**
    $idx = vIndex + offset$
    $p_{x_v|c} = p_{x_v|c} + p_{idx,c}$
  end for
  for $c = 1 : C$ **do**
    $p(x_v) = p(x_v) + p(c)p_{x_v|c}$
  end for
  $p_{c|x} = \frac{p(c)p_{x_v|c}}{p(x_v)}$ # p(c) is the uniform prior
  Return $argmax_c p_{c|x}$
end procedure

Looking the algorithm above, we reach at a runtime of $O(2^h v)$. Therefore, the total runtime is equal to $O(D) + O(2^h v) = max((v + h), 2^h v)$, which will most likely be equal to $O(2^h v)$. It is important to mention that differently from the Naive Bayes, the full model suffer a lot in terms of runtime, when we have missing features.

Conclusion

In this exercise we did a comparison between the full model and the Naive Bayes model for a generative binary classifier. We discovered that without any assumption about the distribution of the data, the number of parameters that we need to train grows exponentially with the number of features.

Comparing the case where we have little data with the case where we have a lot of data, we discovered where each of the models is most suited. Naive Bayes shines the most in the former case, because the full model will overtit due to its huge amount of parameters. On the latter case, the full model is the best choice, because Naive Bayes will sufer from underfitting due to its simplifications.

Training the model, we saw that both models have $O(ND)$ runtime. This makes sense, since the training time is determined by the number of samples, not by the number of parameters. Nevertheless, we should note that the training of the full model will most likely give birth to sparse probabilities tables, where most of the values are zero.

When testing the model, we saw that the full model have a better runtime then the Naive Bayes. This happens because the full model only require C lookups on the probabilities tables, in order to find the best class for a given input. The computation of the index can be done before the table lookups, which decouples the two steps. On the other hand, Naive Bayes requires checking every feature for every class. This coupling between the class loop and the feature loop, makes the runtime to be significantly slower.

Lastly, when we have missing features, we saw that Naive Bayes outperforms the full model. In fact, there are few models that can handle missing features in the input. For generative clasifiers, we can use the marginalization property, as we did in this exercise. However, this will not work for discriminative models. Going back to the question, the conditional class independence assumption of the NBC, makes dealing with missing features easy. We just have to add a conditional that detects if a feature is missing and in case it is, just skip to the next iteration. This minimal adjust of the algorithm does not change its running time at all. On the other hand, since the full model cannot decouple the features, we need to sweep throug all the combinations of the missing features to perform marginalization. This makes the running time of the full model to increase exponentially with the number of hidden features.