Aleksei Maliutin
1224****
a*****n@student.uva.nl

Homework Assignment 3
Machine Learning 1, 18/19

2018-12-16

1 Naive Bayes Text Classification

Naive Bayes (NB) is a particular form of classification that makes strong independence assumptions regarding the features of the data, conditional on the classes (see Bishop section 4.2.3). Specifically, NB assumes each feature is independent given the class label. In contrast, when we looked at probabilistic generative models for classification in the lecture, we used a full-covariance Gaussian to model data from each class, which incorporates correlation between all the input features (i.e. they are not conditionally independent).

1. (2 points)

Write down the data likelihood, $p(\mathbf{T}, \mathbf{X}|\boldsymbol{\theta})$ without independence assumptions. Now derive the data likelihood for the general k classes naive Bayes classifier, stating where you make use of the product rule and the naive Bayes assumption. You should write the likelihood in terms of $p(x_d|\mathcal{C}_k)$, meaning you should not assume the explicit Bernoulli distribution.
ANSWER:

$$
\begin{aligned}
p(\mathbf{T}, \mathbf{X}|\boldsymbol{\theta}) &= \prod_{n=1}^{N} p(t_n, \boldsymbol{x}_n|\boldsymbol{\theta}) \\
&= \prod_{n=1}^{N} p(t_n|\boldsymbol{\theta}) \cdot p(\boldsymbol{x}_n|t_n, \boldsymbol{\theta}) \\
&= \prod_{n=1}^{N} \prod_{k=1}^{K} \Big(p(t_n = k|\boldsymbol{\theta}) \cdot \underbrace{p(\boldsymbol{x}_n|t_n = k, \boldsymbol{\theta})}_{\text{use Naive Bayes assumption}} \Big)^{\mathbb{I}(t_n=k)} \\
&= \prod_{n=1}^{N} \prod_{k=1}^{K} \Big(\pi_k \cdot \prod_{d=1}^{D} p(x_{nd}|\mathcal{C}_k, \theta_{dk}) \Big)^{\mathbb{I}(t_n=k)}
\end{aligned}
$$

2. (0.5 points)

How does the number of parameters change if you make use of the naive Bayes assumption? Why is the assumption called naive and can you think of an example in which this assumption does not hold?
ANSWER:
In general case with D inputs (features) we have $2^D - 1$ independent variables (due to the summation constraint). In contrast, using Naive Bayes assumption - the feature values are treated as independent, conditioned on the class $\mathcal{C}_k$ - we have *only* D independent parameters for each class. The name "Naive" means, that we reject the common nature of input features have some kind of correlation.
Example: Let's take a look on rent market, it is obvious that in case of input features such as Year of construction, floor and apt size the last ones has some kind of dependence on the first one, because 100 years ago there were not 30-floors houses etc. Hence, loosing that correlation makes our model "Naive"

3. (0.5 points)

Write down the data log-likelihood $\ln p(\mathbf{T}, \mathbf{X}|\boldsymbol{\theta})$ for the Bernoulli model.
ANSWER:

$$
\begin{aligned}
\ln p(\mathbf{T}, \mathbf{X}|\boldsymbol{\theta}) &= \ln \prod_{n=1}^{N} \prod_{k=1}^{K} \Big(\pi_k \cdot \prod_{d=1}^{D} p(x_{nd}|\mathcal{C}_k, \theta_{dk}) \Big)^{\mathbb{I}(t_n=k)} \\
&= \sum_{k=1}^{K} \sum_{n \in \mathcal{C}_k}^{N} \Big(\ln \pi_k + \sum_{d=1}^{D} x_{nd} \ln \theta_{dk} + (1 - x_{nd}) \ln (1 - \theta_{dk}) \Big)
\end{aligned}
$$

4. (4 points)

Solve for the MLE estimators for θ_{dk}. Express in your own words how the result can be interpreted.
ANSWER:

$$\frac{\partial}{\partial \theta_{dk}} \ln p(\mathbf{T}, \mathbf{X}|\boldsymbol{\theta}) = \frac{\partial}{\partial \theta_{dk}} \sum_{a=1}^{K} \sum_{n \in \mathcal{C}_a} \Big(\ln \pi_a + \sum_{b=1}^{D} x_{nb} \underbrace{\ln \theta_{ab}}_{\text{depends on } \theta_{dk} \text{ if } \begin{cases} d=a \\ k=b \end{cases}} + (1 - x_{nb}) \underbrace{\ln (1 - \theta_{ab})}_{\text{depends on } \theta_{dk} \text{ if } \begin{cases} d=a \\ k=b \end{cases}} \Big)$$

$$= \sum_{n \in \mathcal{C}_k}^{N} \Big(\frac{x_{nd}}{\theta_{dk}} - \frac{1 - x_{nd}}{1 - \theta_{dk}} \Big)$$

$$= 0$$

Hence, splitting θ_{dk} and x_{nd} into different parts of equations, we are getting the following:

$$\theta_{dk} \underbrace{\sum_{n \in \mathcal{C}_k}^{N} 1}_{=N_k} = \sum_{n \in \mathcal{C}_k}^{N} x_{nd}$$

And solving it w.r.t. θ_{dk}:

$$\theta_{dk} = \frac{1}{N_k} \sum_{n \in \mathcal{C}_k}^{N} x_{nd}$$

According to it, θ_{dk} means the average number of tokens *d* per document for class *k*.

5. (1 point)

Write $p(\mathcal{C}_1|\mathbf{x})$ for the general k classes naive Bayes classifier.
ANSWER:

$$\begin{aligned} p(\mathcal{C}_1|\mathbf{x}) &= \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x})} \\ &= \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{\sum_{k=1}^{K} p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)} \\ &= \frac{\pi_1 \cdot \prod_{d=1}^{D} p(x_d|\theta_{d1})}{\sum_{k=1}^{K} \pi_k \cdot \prod_{d=1}^{D} p(x_d|\theta_{dk})} \end{aligned}$$

6. (1 point)

Write $p(\mathcal{C}_1|\mathbf{x})$ for the Bernoulli model.
ANSWER:

$$\begin{aligned} p(\mathcal{C}_1|\mathbf{x}) &= \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x})} \\ &= \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{\sum_{k=1}^{K} p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)} \\ &= \frac{\pi_1 \cdot \prod_{d=1}^{D} \theta_{d1}^{x_d} \cdot (1 - \theta_{d1})^{1-x_d}}{\sum_{k=1}^{K} \pi_k \cdot \prod_{d=1}^{D} \theta_{dk}^{x_d} \cdot (1 - \theta_{dk})^{1-x_d}} \end{aligned}$$

7. (2 points)

For the Bernoulli model, express the conditions (inequalities) of the region where $\mathbf{x}$ is predicted to be in $\mathcal{C}_1$. Provide linear inequalities in the form $\mathbf{x}^T\mathbf{a} > c$

ANSWER:
The point **x** will be predicted to be in $\mathcal{C}_1$ if $p(\mathcal{C}_1|\mathbf{x}) > p(\mathcal{C}_k|\mathbf{x}), \forall k \neq 1$. Hence, we can use formulas from previous step:

$$p(\mathcal{C}_1|\mathbf{x}) > p(\mathcal{C}_k|\mathbf{x}), \forall k \neq 1$$

$$\pi_1 \cdot \prod_{d=1}^{D} \theta_{d1}^{x_d} \cdot (1-\theta_{d1})^{1-x_d} > \pi_k \cdot \prod_{d=1}^{D} \theta_{dk}^{x_d} \cdot (1-\theta_{dk})^{1-x_d}, \forall k \neq 1$$

$$\prod_{d=1}^{D} \Big(\frac{\theta_{d1}}{\theta_{dk}}\Big)^{x_d} \cdot \Big(\frac{1-\theta_{d1}}{1-\theta_{dk}}\Big)^{-x_d} > \frac{\pi_k}{\pi_1} \cdot \prod_{d=1}^{D} \cdot \Big(\frac{1-\theta_{dk}}{1-\theta_{d1}}\Big)^{1}, \forall k \neq 1$$

Take logarithm for both sides:

$$\sum_{d=1}^{D} x_d \cdot \underbrace{\ln\Big(\frac{\theta_{d1}(1-\theta_{dk})}{\theta_{dk}(1-\theta_{d1})}\Big)}_{=a_{1k}} > \underbrace{\ln\frac{\pi_k}{\pi_1} + \sum_{d=1}^{D} \ln\Big(\frac{1-\theta_{dk}}{1-\theta_{d1}}\Big)}_{=c_{1k},const}, \forall k \neq 1$$

$$\mathbf{x}^T \mathbf{a_{1k}} > c_{1k}, \forall k \neq 1$$

8. (0.5 points)

So far we only considered feature vectors with discrete values, is it possible to have continuous features? Argue why yes/no and if yes, what would be an example?
ANSWER:
Of course we can use not only binary representation of x_i but also continuous one, however, for modeling it we should use another kind of distribution (due to the discrete nature of Bernoulli).
Now we have a large BoW (actually, we are using each token for building it), but it could be useful make a classification task based on some key-point words AND the nearest distances to them from document (using **Word2Vec**, require a lot of preprocessing, but then the size of final model should be << comparing with default ones).
For instance, let's take a look on the next sentence: **Barcelona plays against PSV tonight**, hence, for our default version we have the following vector: $\mathbf{x} = [Barcelona = 1, play = 1, ..]$, however, if we make the same, but using **Word2Vec** and key-words - *sport, music, present, future* - we'll get $\mathbf{x} = [sport = 0.93, music = 0.15, present = 0.72, ..]$ what is easy classify as *sport event tonight* and what is more important, the required amount of memory depends only on the volume of key-words and does not depend on the collection of documents, what is great advantage during work with large collection.

2 Multi-class Logistic Regression and Multilayer Perceptrons

In class we saw the binary classification version of logistic regression. Here you will derive the gradients for the general case $K > 2$. Much of the preliminaries are in Bishop 4.3.4. This will be useful for Lab 2.

1. (3 points)

In this question we will be deriving the matrix form for the gradient of the log-likelihood.
ANSWER:

- Write down the likelihood $p(\mathbf{T}|\mathbf{\Phi}, \mathbf{w}_1, ..., \mathbf{w}_K)$. Use the entries of **T** as selectors of the correct class.

$$p(\mathbf{T}|\mathbf{\Phi}, \mathbf{w}_1, ..., \mathbf{w}_K) = \prod_{n=1}^{N} \prod_{k=1}^{K} y_k(\boldsymbol{\phi}_n)^{t_{nk}}$$

- Write down the log-likelihood $\ln p(\mathbf{T}|\mathbf{\Phi}, \mathbf{w}_1, ..., \mathbf{w}_K)$. Use the entries of **T** as selectors of the correct class.

$$\ln p(\mathbf{T}|\mathbf{\Phi}, \mathbf{w}_1, ..., \mathbf{w}_K) = \sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \cdot \ln y_k(\boldsymbol{\phi}_n)$$

- Compute the derivative $\frac{\partial y_k}{\partial \mathbf{w}_j}$.

$$
\begin{aligned}
\frac{\partial y_k}{\partial \mathbf{w}_j} &= \frac{\partial y_k(\boldsymbol{\phi})}{\partial \mathbf{w}_j} = \frac{\partial}{\partial \mathbf{w}_j} \frac{exp(a_k)}{\sum_i exp(a_i)} \\
&= \frac{exp(a_k) \cdot \frac{\partial a_k}{\partial \mathbf{w}_j} \cdot \sum_i exp(a_i) - exp(a_k) \cdot exp(a_j) \cdot \frac{\partial a_j}{\partial \mathbf{w}_j}}{\Big(\sum_i exp(a_i)\Big)^2} \\
&= \frac{exp(a_k)}{\sum_i exp(a_i)} \frac{\partial a_k}{\partial \mathbf{w}_j} - \frac{exp(a_k) \cdot exp(a_j)}{\Big(\sum_i exp(a_i)\Big)^2} \frac{\partial a_j}{\partial \mathbf{w}_j} \\
&= y_k(\boldsymbol{\phi}) \boldsymbol{\phi}^T \mathbf{I}_{kj} - y_k(\boldsymbol{\phi}) y_j(\boldsymbol{\phi}) \boldsymbol{\phi}^T \\
&= y_k(\boldsymbol{\phi}) \boldsymbol{\phi}^T \cdot (\mathbf{I}_{kj} - y_j(\boldsymbol{\phi}))
\end{aligned}
$$

- Use the chain rule and the derivatives $\frac{\partial y_k}{\partial \mathbf{w}_j}$ to compute $\nabla_{\mathbf{w}_j} \ln p(\mathbf{T}|\boldsymbol{\Phi}, \mathbf{w}_1, ..., \mathbf{w}_K)$.

$$
\begin{aligned}
\frac{\partial \ln p(\mathbf{T}|\boldsymbol{\Phi}, \mathbf{w}_1, ..., \mathbf{w}_K)}{\partial \mathbf{w}_j} &= \sum_{n=1}^{N} \sum_{k=1}^{k} \frac{t_{nk}}{y_k(\boldsymbol{\phi}_n)} \cdot \frac{\partial y_k}{\partial \mathbf{w}_j} \\
&= \sum_{n=1}^{N} \sum_{k=1}^{k} \frac{t_{nk}}{y_k(\boldsymbol{\phi}_n)} \cdot y_k(\boldsymbol{\phi}_n) \boldsymbol{\phi}_n^T \cdot (\mathbf{I}_{kj} - y_j(\boldsymbol{\phi}_n)) \\
&= \sum_{n=1}^{N} \sum_{k=1}^{k} t_{nk} \cdot \boldsymbol{\phi}_n^T \cdot (\mathbf{I}_{kj} - y_j(\boldsymbol{\phi}_n)) \\
&= \sum_{n=1}^{N} \boldsymbol{\phi}_n^T \cdot \sum_{k=1}^{k} t_{nk} \mathbf{I}_{kj} - \sum_{n=1}^{N} y_j(\boldsymbol{\phi}_n) \boldsymbol{\phi}_n^T \cdot \sum_{k=1}^{k} t_{nk} \\
&= \sum_{n=1}^{N} \Big(t_{nj} - y_j(\boldsymbol{\phi}_n)\Big) \boldsymbol{\phi}_n^T
\end{aligned}
$$

Hence, the gradient is equal to

$$
\nabla_{\mathbf{w}_j} \ln p(\mathbf{T}|\boldsymbol{\Phi}, \mathbf{w}_1, ..., \mathbf{w}_K) = \Big(\frac{\partial \ln p(\mathbf{T}|\boldsymbol{\Phi}, \mathbf{w}_1, ..., \mathbf{w}_K)}{\partial \mathbf{w}_j}\Big)
$$

- Your gradient is now a sum, can you rewrite it as a matrix multiplication? Why is this useful?

$$
\boldsymbol{\Phi}^T (\mathbf{t}_j - \mathbf{y}_j)
$$

We can use vectorized operations from *numpy*, what leads to better performance.

2. (1 points)

Write down the negative log-likelihood, this will be our objective function. Can you recognize the function you obtain? What is the relationship between the two (the function you obtain and the original log-likelihood)? What changes in terms of weight updates during optimization?
ANSWER:

$$
\begin{aligned}
-\ln p(\mathbf{T}|\boldsymbol{\Phi}, \mathbf{w}_1, ..., \mathbf{w}_K) &= -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \cdot \ln y_k(\boldsymbol{\phi}_n) \\
&= E(\mathbf{w}_1, ..., \mathbf{w}_K)
\end{aligned}
$$

The objective function $E(\mathbf{w}_1, ..., \mathbf{w}_K)$ is **Cross-entropy** error function for the multiclass classification problem. It's equal to the negative log-likelihood, hence, maximizing log-likelihood leads to minimizing cross-entropy error, the gradient is the same, but with opposite sign. However, in case of weight updates during optimization, there are no changes, because with log-likelihood we are adding, and now we subtract the same value but with opposite sing. (+ is equal to -(-)).

3. (2 points)

Write a mini-batch gradient algorithm for logistic regression using this objective function. Let B be the batch size, n_B the number of batches and $E_n = -\sum_{n=1}^{N} t_{nk} \cdot \ln y_k(\boldsymbol{\phi}_n)$the objective evaluated on data point n. Make

sure to include indices for time, to define the learning rate and to check what dimension the weights you are updating have, transposing the gradient if necessary.
ANSWER:
First of all, we should notice the main difference of mini-batch comparing with SGD: on each iteration we are working with batch of data, then we aggregate the gradient over that mini-batch and update weights.
Terminate conditions: T - max number of iteration or ϵ - converge ratio.
Hyper-parameters: η - learning rate, *B* - batch size, n_B - the number of batches.

(1) Initialize $\mathbf{w}_1, ..., \mathbf{w}_K$

(2) Initialize η

(3) Initialize $t = 0, \epsilon = +\infty$

(4) For *batch* from 1 to n_B

a) Initialize empty list *Es*

b) For *c* from 1 to B

i. Calculate $\triangledown E_c(\mathbf{w}^{(\tau)})$ (it's a row-vector) and add it to list *Es*

ii. *t++* - increase the number of elapsed iterations

c) Aggregate items from *Es* - depends on implementation, we can simply sum or take the average. For instance, $\triangledown \hat{E}(\mathbf{w}^{(\tau)}) = np.sum(Es, axis = 0)$ (it's also a row-vector)

d) Update widths:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \eta \cdot \triangledown^T \hat{E}(\mathbf{w}^{(\tau)})$$

NOTE: we use transpose here, because gradient in our notation is row-vector.

e) (Optional) Decrease η

f) (Optional) Earlier termination: if $\epsilon > distance(\mathbf{w}^{(\tau)}, \mathbf{w}^{(\tau+1)})$ or $t > T$

(5) Return current $\mathbf{w}_1, ..., \mathbf{w}_K$

Name one advantage of mini-batch gradient descent over stochastic gradient descent with single data points. How does it compare to full batch gradient descent?
ANSWER:
Comparing with SGD, mini-batch is more efficient, because we don't walk a lot in directions of the noise datapoint (it's eliminated via aggregation step, 4c). Also, it almost does not have an oscillating behaviour around the minimum. Moreover, from implementation point of view, we can use vectorized computation for batches.
On the other hand, comparing with full batch gradient descent, the most important advantage is that it fits in memory: all training data and algorithm implementations. Additionally, mini-batch provides higher update frequency, because we don't need to wait the evaluation on the whole dataset.

4. (1 point)

Consider now the Multilayer Perceptron in Fig. 1, with input features of dimension 2, 2 hidden units and 3 output classes. For which weight values and type of activation functions can we fall back to the multiclass logistic regression case?

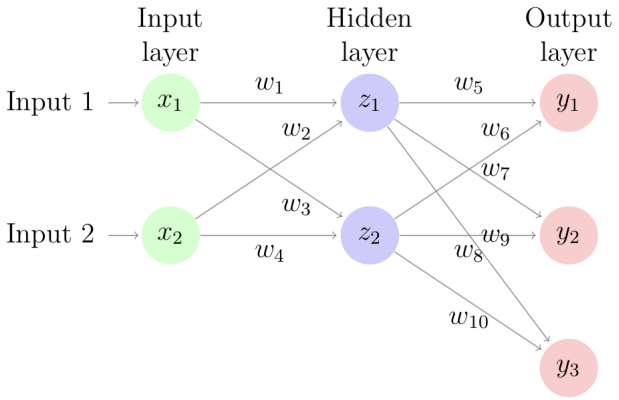

Figure 1: Neural Network

ANSWER:
NOTE: Figure above is not exactly right, in my view, because it does not include x_0 - the bias.
The activation functions on the hidden layer should be Identity activation function.
Activation for the output is a **SoftMax**.

$$W_1 = \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$
$$W_2 = \begin{bmatrix} w_5 & w_6 \\ w_7 & w_8 \\ w_9 & w_{10} \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \\ \mathbf{w}_3 \end{bmatrix}$$

5. (3 points)

Consider again the network in Fig. 1. We now consider the case where the activation function on the hidden layer is a ReLU, the activation for the output is a Softmax and we consider again the cross-entropy loss E.
ANSWER:

- Compute the forward pass for the data point $x = (0.3, 0.7)^T$ with label $y = (0, 0, 1)^T$. Evaluate the loss.

$$z^{in} = W_1 \cdot x = \begin{bmatrix} 0.729 \\ 0.412 \end{bmatrix}$$
$$z^{out} = ReLU(z^{in}) = \begin{bmatrix} 0.729 \\ 0.412 \end{bmatrix}$$
$$y^{in} = W_2 \cdot z^{out} = \begin{bmatrix} 0.446 \\ 0.726 \\ 0.932 \end{bmatrix}$$
$$y^{out} = SoftMax(y^{in}) = \begin{bmatrix} 0.253 \\ 0.335 \\ 0.412 \end{bmatrix}$$

And finally calculate the cross-entropy loss:

$$E = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \cdot \ln y_k = -\Big(0 \cdot \ln y_1 + 0 \cdot \ln y_2 + 1 \cdot \ln y_3\Big)$$
$$= -\ln 0.412 = \mathbf{0.887}$$

- Compute the derivative $\frac{\partial E}{\partial w_5}$. Write down the formula first and then compute the numerical result.

$$\frac{\partial E}{\partial w_5} = \sum_i \frac{\partial E}{\partial y_i^{out}} \cdot \frac{\partial y_i^{out}}{\partial y_1^{in}} \cdot \frac{\partial y_1^{in}}{\partial w_5}$$

Taking into account $y = (0, 0, 1)$

the first two terms of sum eliminated:

$$\begin{aligned}
&= -\frac{1}{y_3^{out}} \cdot \frac{\partial y_3^{out}}{\partial y_1^{in}} \cdot \frac{\partial y_1^{in}}{\partial w_5} \\
&= -\frac{1}{y_3^{out}} \cdot (-y_1^{out} \cdot y_3^{out}) \cdot \frac{\partial (z_1 w_5 + z_2 w_6)}{\partial w_5} \\
&= y_1^{out} \cdot z_1 \\
&= \mathbf{0.184}
\end{aligned}$$

- The other derivatives are given by... Perform a weight update with learning rate 0.05.

$$\mathbf{W}^{\tau+1} = \mathbf{W}^{\tau} - \eta \mathbf{W}_{upd}$$

After calculation we are getting the new weights matrices:

$$W_1 = \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} = \begin{bmatrix} 0.402 & 0.875 \\ 0.583 & 0.347 \end{bmatrix}$$

$$W_2 = \begin{bmatrix} w_5 & w_6 \\ w_7 & w_8 \\ w_9 & w_{10} \end{bmatrix} = \begin{bmatrix} 0.111 & 0.865 \\ 0.808 & 0.303 \\ 0.792 & 0.912 \end{bmatrix}$$

- Compute the loss again, what happens?

$$y^{out} = SoftMax(y^{in}) = \begin{bmatrix} 0.250 \\ 0.330 \\ 0.420 \end{bmatrix}$$

And the cross-entropy loss now is

$$\begin{aligned}
E &= -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \cdot \ln y_k = -\Big(0 \cdot \ln y_1 + 0 \cdot \ln y_2 + 1 \cdot \ln y_3\Big) \\
&= -\ln 0.420 = \mathbf{0.867}
\end{aligned}$$

As we see, it has decreased, in other words, our neural network started the adaptation (training) to current data.