Exercises on text classification with (mostly) linear models

Ion Androutsopoulos, 2024–25

Submit as a group of 2–3 members (unless specified otherwise in the lectures) a report for exercise 11 (max. 10 pages, PDF format). Include in your report all the required information, especially experimental results. Do not include code in the report, but include a link to a Colab notebook containing your code. Make sure to divide fairly the work of your group to its members and describe in your report the contribution of each member. The contribution of each member will also be checked during the oral examination of your submission. For delayed submissions, one point will be subtracted per day of delay.

1. Let $< X_1, X_2, X_3 > = < 0, 1, 0 >$ be the feature vector of an unseen object to be classified. We use a *k*-NN classifier, with two classes ($C = 0$, $C = 1$), $k = 3$, and the number of different features as the distance measure. The training dataset contains the four vectors (rows) of the table (excluding the first column). Which class will the unseen object be classified in?

d	0	1	0	C?
2	**1**	**0**	**0**	**0**
1	**0**	**1**	**1**	**0**
3	1	0	1	1
2	**1**	**1**	**1**	**1**

Answer: The first column of the table shows the distance (d) from the unseen object to the corresponding training instances. The k = 3 nearest neighbors are at distances 1 and 2. The majority class among the k nearest neighbors is C = 0. Hence, the unseen object will be classified in C = 0.

2. Two training examples are *inconsistent*, if they have the same feature values, but different "correct" class labels. If there are *no* inconsistent training examples and we evaluate a *k*-NN classifier on the same dataset we used for training, then its accuracy will be:

___ 100%
_X_ 100% if $k = 1$, but may not be 100% if $k \neq 1$
___ none of the above (the accuracy depends on the dataset).

Answer: If k = 1, every test instance will be classified in the class of its closest training instance, and the closest training instance will be itself (or a copy of itself, with the same correct class label, since there are no inconsistencies), given that the test dataset is the same as the training dataset. Hence, for k = 1 every test instance will be classified correctly and the accuracy will be 100%.

If k ≠ 1, each test instance will be classified in the majority class of its k nearest training (and test) instances, and the majority class may not be the correct one. Hence, for k ≠ 1 the accuracy may not be 100%.

3. (a) Based on the training examples of the table, the entropy of *C* is:

_X_ H(*C*) = 1 ___ H(*C*) = 0 ___ H(*C*) = ½

X	*Y*	*Z*	*C*
0	1	0	positive
0	1	1	positive
0	1	0	positive
1	0	1	positive
1	1	0	negative
0	0	1	negative
0	0	0	negative
0	0	1	negative

Answer: Based on the training examples, P(C=positive) = P(C=negative) = ½. Hence, C has two equally probable values and, therefore, its entropy is maximum. For two possible outcomes, the maximum entropy value is 1. You should be able to reach the same conclusion by computing the entropy of C using the definition of entropy.

(b) Based on the training examples of the table, the highest information gain score is:

___IG(C, X) _X_IG(C, Y) ___IG(C, Z)

Answer: The training data show that if we learn that Y = 1, it is very likely (probability ¾) that C = positive; and if we learn that Y = 0, it is very likely (probability ¾) that C = negative. Hence, knowing the value of Y reduces the entropy of C, which was initially maximum. You should be able to reach the same conclusion by computing IG(C,Y).

By contrast, the training data show that if we learn that X = 1, the probability of C = positive remains ½, equal to the probability of C = negative; and if we learn that X = 0, the probability of C = positive remains ½, equal to the probability of C = negative. Hence, whatever the value of X, learning X still leaves us with two equally probable outcomes C=positive and C=negative and a maximum entropy, i.e., knowing X does not reduce the entropy of C, which implies that IG(C,X) = 0. You should be able to reach the same conclusion by computing IG(C,X).

Similarly, IG(C,Z) = 0.

4. Confirm that *IG(X; C) = IG(C; X)* (slide 9).

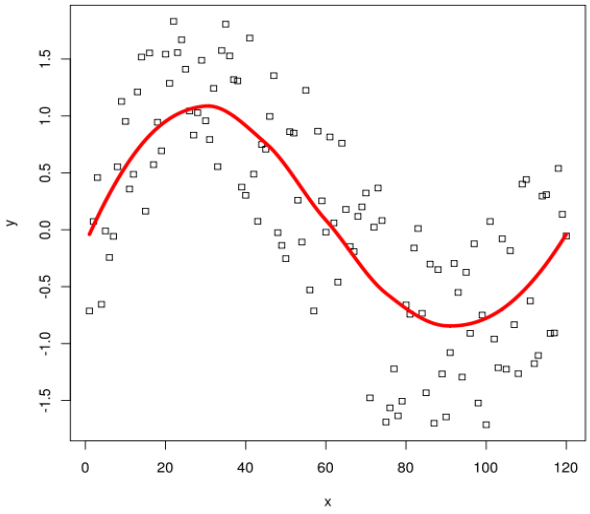
5. (a) Explain why supervised machine learning algorithms must be evaluated on different data than those used for training.

Answer: If evaluated on the training dataset, an algorithm that simply memorizes the training instances and their correct class labels would obtain an accuracy score of 100%, without this score being indicative of how well it would perform on unseen instances. More generally, a supervised learning algorithm often performs better (due to overfitting) on the training dataset than on unseen data.

(b) A researcher submitted a conference paper that described a named entity recognizer (NER) that used supervised machine learning. The paper reported experiments with different feature sets. For each feature set, the NER had been trained on a training corpus (the same for all feature sets) and had been tested on a separate test corpus (the same for all feature sets). The paper listed test F1 results for each feature set, which were used to select the best feature set. With the best feature set, the NER of the paper had better F1 (and other) scores on the test corpus, compared to other NER systems that had been trained and evaluated on the same training and test corpora, respectively. However, the reviewers rejected the paper, noting that the test results were unreliable and that the comparison to the other systems was wrong, because the test set was in effect also used during training. Were the reviewers right or not? Explain why. If they were right, what should the researcher do to address the problem?

Answer: The reviewers were right to complain, because the researcher selected the feature set that led to the best results on the test dataset, i.e., the test dataset was used for feature selection, which in effect is part of training (or at least configuring) the system. The chosen feature set may contain features that work well for the particular test set (especially if the test set is small), but may not work that well on other test sets; in effect, the system may have overfitted the particular test set. The researcher should select the feature set that leads to the best results on a held-out part of the training set (or perform a cross-validation on the training set to select the feature set), and then evaluate the NER (with the chosen feature set) on the test set.

6. The dots of the figure are a sample from a population that actually follows the function $y = f(x)$, whose curve is the solid line.[1] Due to noise (e.g., introduced by the measurements), the dots are not exactly on the solid line. We wish to learn from the sample a function $y = h(x)$ that will approximate the (unknown to us) true $f(x)$.



(a) Explain why linear regression would not lead to a good $h(x)$.

Answer: Linear regression learns straight lines (more generally, planes or hyperplanes). In our case, the target curve $y = f(x)$ cannot be approximated well by a single straight line.

(b) How could we learn a better $h(x)$ using k-NN regression (or variant)? What would we do during training? After training, given an x, how would we obtain $y = h(x)$;

Answer: During training, we would simply store all the coordinates (x, y) of the instances (dots) of the sample. After training, given an x', we would retrieve the k (training) instances of the sample whose x values are closest to x' and we would return their average y. We could also assign weights to the y values of the k neighbors (e.g., inversely proportional to the distance from the neighbor's x to x').

7. Derive the weights update rule of least squares linear regression, when using stochastic gradient descent.

Answer: With stochastic gradient descent, the weights of least squares linear regression are updated as follows:

$$\vec{w} \leftarrow \vec{w} - \eta \cdot \nabla_{\vec{w}} E_i(\vec{w}), \text{ where } E_i(\vec{w}) = \frac{1}{2}\left[f_{\vec{w}}(\vec{x}^{(i)}) - y^{(i)}\right]^2$$

and:

$$\nabla_{\vec{w}} E_i(\vec{w}) = \langle \frac{\partial E_i(\vec{w})}{\partial w_0}, \frac{\partial E_i(\vec{w})}{\partial w_1}, \cdots, \frac{\partial E_i(\vec{w})}{\partial w_l}, \cdots, \frac{\partial E_i(\vec{w})}{\partial w_n} \rangle$$

For $l \in \{0, \ldots, n\}$:

$$\frac{\partial E_i(\vec{w})}{\partial w_l} = \left[f_{\vec{w}}(\vec{x}^{(i)}) - y^{(i)}\right] \cdot x_l^{(i)}$$

Hence:

$$\nabla_{\vec{w}} E_i(\vec{w}) = \left[f_{\vec{w}}(\vec{x}^{(i)}) - y^{(i)}\right] \cdot \langle x_1^{(i)}, \cdots, x_l^{(i)}, \cdots x_n^{(i)} \rangle =$$

$$= \left[f_{\vec{w}}(\vec{x}^{(i)}) - y^{(i)}\right] \cdot \vec{x}^{(i)}$$

and the weights update rule is:

$$\vec{w} \leftarrow \vec{w} - \eta \cdot \left[f_{\vec{w}}(\vec{x}^{(i)}) - y^{(i)}\right] \cdot \vec{x}^{(i)}$$

8. Derive the weights update rule of logistic regression, when using batch gradient ascent.

[1] Figure from http://en.wikipedia.org/wiki/Local_regression.

Answer: With batch gradient ascent, the weights of logistic regression are updated as follows (using natural logarithms):

$$\vec{w} \leftarrow \vec{w} + \eta \cdot \nabla_{\vec{w}} l(\vec{w})$$

where:

$$l(\vec{w}) = \sum_{i=1}^{m} y^{(i)} \ln P\left(c_+ \middle| \vec{x}^{(i)}; \vec{w}\right) + \left(1 - y^{(i)}\right) \ln P\left(c_- \middle| \vec{x}^{(i)}; \vec{w}\right) \quad (1)$$

and:

$$P(c_+|\vec{x}; \vec{w}) = 1/\left(1 + e^{-\vec{w}\vec{x}}\right) \text{ (2)}$$
$$P(c_-|\vec{x}; \vec{w}) = e^{-\vec{w}\vec{x}}/\left(1 + e^{-\vec{w}\vec{x}}\right) \text{ (3)}$$

Using (2) and (3) in (1), we obtain:

$$l(\vec{w}) = \sum_{i=1}^{m} y^{(i)} ln\left(1/\left(1 + e^{-\vec{w}\vec{x}^{(i)}}\right)\right) + \left(1 - y^{(i)}\right) ln\left(e^{-\vec{w}\vec{x}^{(i)}}/\left(1 + e^{-\vec{w}\vec{x}^{(i)}}\right)\right) =$$

$$= \sum_{i=1}^{m} \cancel{-y^{(i)} ln\left(1 + e^{-\vec{w}\vec{x}^{(i)}}\right)} + ln\left(e^{-\vec{w}\vec{x}^{(i)}}\right) - ln\left(1 + e^{-\vec{w}\vec{x}^{(i)}}\right) - y^{(i)} ln\left(e^{-\vec{w}\vec{x}^{(i)}}\right)$$
$$\cancel{+ y^{(i)} ln\left(1 + e^{-\vec{w}\vec{x}^{(i)}}\right)}$$
$$= \sum_{i=1}^{m} ln\left(e^{-\vec{w}\vec{x}^{(i)}}\right) - ln\left(1 + e^{-\vec{w}\vec{x}^{(i)}}\right) - y^{(i)} ln\left(e^{-\vec{w}\vec{x}^{(i)}}\right)$$

The gradient is:

$$\nabla_{\vec{w}} l(\vec{w}) = \langle \frac{\partial l(\vec{w})}{\partial w_0}, \frac{\partial l(\vec{w})}{\partial w_1}, \cdots, \frac{\partial l(\vec{w})}{\partial w_l}, \cdots, \frac{\partial l(\vec{w})}{\partial w_n} \rangle$$

and for $l \in \{0, \ldots, n\}$:

$$\frac{\partial l(\vec{w})}{\partial w_l} = -\sum_{i=1}^{m} \frac{e^{-\vec{w}\vec{x}^{(i)}} \cdot x_l^{(i)}}{e^{-\vec{w}\vec{x}^{(i)}}} - \frac{e^{-\vec{w}\vec{x}^{(i)}} \cdot x_l^{(i)}}{1 + e^{-\vec{w}\vec{x}^{(i)}}} - y^{(i)} \frac{e^{-\vec{w}\vec{x}^{(i)}} \cdot x_l^{(i)}}{e^{-\vec{w}\vec{x}^{(i)}}}$$
$$= -\sum_{i=1}^{m} x_l^{(i)} - x_l^{(i)} \frac{e^{-\vec{w}\vec{x}^{(i)}}}{1 + e^{-\vec{w}\vec{x}^{(i)}}} - y^{(i)} x_l^{(i)}$$
$$= -\sum_{i=1}^{m} \left(\frac{1 + e^{-\vec{w}\vec{x}^{(i)}} - e^{-\vec{w}\vec{x}^{(i)}}}{1 + e^{-\vec{w}\vec{x}^{(i)}}} - y^{(i)} \right) x_l^{(i)}$$
$$= \sum_{i=1}^{m} \left[y^{(i)} - P\left(c_+ \middle| \vec{x}^{(i)}\right) \right] x_l^{(i)}$$

Hence, the weights update rule is:

$$w_l \leftarrow w_l + \eta \cdot \sum_{i=1}^{m} \left[y^{(i)} - P\left(c_+ \middle| \vec{x}^{(i)}\right) \right] x_l^{(i)}$$

9. We trained a logistic regression classifier on the training instances (dots) of the figure. There are two classes (black, white) and two features (x_1, x_2). If we evaluate the classifier on the training (same) instances, will it classify all the training instances correctly? If yes, why? If not, what can we do to help the classifier classify correctly all the training instances?

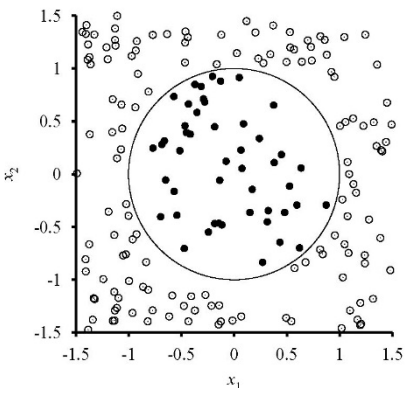


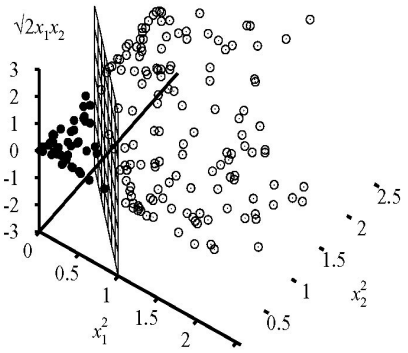
Figure from the book of Russel and Norvig.

Answer: Logistic regression classifiers are linear separators. They learn a straight line (in our case) or a hyperplane (more generally), and then classify unseen instances by computing whether or not they fall above or below the straight line (or hyperplane). The instances (dots) of the figure are not linearly separable (no straight line separates all the black from all the white dots). Hence, a logistic regression classifier cannot learn to separate them all (classify them all correctly), even if it is trained on the same instances.

By using more features, however, the instances may become linearly separable. For example, with the following transformation:

$$\vec{F}(\vec{x}) = \left\langle x_1^2, x_2^2, \sqrt{2}x_1x_2 \right\rangle$$

the instances are moved to a new, 3-dimensional vector space where they are linearly separable (2nd figure, also from the book of Russel & Norvig). Hence, a logistic regression classifier could learn to classify them all correctly, provided that we apply the transformation above.



10. A student trained a logistic regression classifier with (batch) gradient ascent. To speed up the training, the student used a large constant η value in the weights update rule, hoping that this way the gradient ascent would need fewer steps. However, the gradient ascent no longer converged; it seemed to be oscillating between two values. Why did this happen?

Answer: Because of the large η value, the gradient ascent possibly went over the maximum (overshot it), then made another (large) step towards the opposite direction (overshooting again the maximum) etc. Reducing the value of η at each step would probably have helped in this case.

11. Develop a sentiment classifier for a kind of text of your choice (e.g., tweets, product or movie reviews). Use an existing sentiment analysis dataset with at least two classes (e.g., positive/negative or positive/negative/neutral).[2] The classes should be mutually exclusive, i.e., this is a single-label multi-class classification problem. You may use Boolean, TF, or TF-IDF features corresponding to words or *n*-grams, and/or other features. You may apply any feature selection (or dimensionality reduction) method you consider appropriate. You may also want to try using centroids of pre-trained word embeddings.[3] You can write your own code to produce feature vectors, perform feature selection (or dimensionality reduction) and train the classifier (e.g., using SGD, in the case of logistic regression), or you can use existing

[2] See, for example, the Large Movie Review Dataset (http://ai.stanford.edu/~amaas/data/sentiment/), the Cornell Movie Review Data (http://www.cs.cornell.edu/people/pabo/movie-review-data/, included in NLTK), or the Twitter Sentiment Analysis Dataset (https://www.kaggle.com/c/twitter-sentiment-analysis2/data, you need to create a Kaggle account).

[3] Pre-trained word embeddings are available, for example, from http://nlp.stanford.edu/projects/glove/, https://fasttext.cc/docs/en/crawl-vectors.html, https://code.google.com/archive/p/word2vec/.

implementations.[4] You should experiment with at least logistic regression (or multinomial logistic regression, if you have more than two classes) and optionally (if you are keen and have free time) additional learning algorithms (e.g., *k*-NN). Make sure that you use separate training, development, and test subsets. Tune the feature set and hyper-parameters (e.g., regularization weight λ in logistic regression) on the development subset. Include experimental results of a baseline majority classifier, i.e., a classifier that always assigns the most frequent class of the training data. Include in your report:

- Precision, recall, F1, precision-recall AUC scores, for each class and classifier, separately for the training, development, and test subsets. Use three separate tables for the training, development, and test results. In each table, use a separate row for each classifier (or baseline), and show the precision, recall, F1, PR-AUC scores of the classes in columns (four columns per class).
- Macro-averaged precision, recall, F1, precision-recall AUC scores (all computed by averaging the corresponding scores of the previous bullet over the classes), for each classifier, separately for the training, development, and test subsets.[5] Show these results by adding four more columns to the tables of the previous bullet.
- For each classifier, two learning curves (like slides 53, 56) showing macro-averaged F1 computed on (i) the increasingly larger (e.g., 10%, 20%, etc.) subset of the training set the classifier has encountered, (ii) the (always entire) development subset. Show a separate diagram for each classifier, with two curves in each diagram.
- A short description of the methods and datasets you used, including statistics about the datasets (e.g., average document length, number of training/dev/test documents, vocabulary size) and a description of the preprocessing steps that you performed.

12. [Optional] (a) Let $< X_1, X_2, X_3 > = < 0, 1, 0 >$ be the feature vector of a text to be classified. We use a multivariate Bernoulli Naive Bayes classifier, with two classes ($C = 0$, $C = 1$), and the training data of the table. What will be the decision of the classifier? Use Laplace estimates for $P(X_i|C)$. All the features X_i are Boolean.

X_1	X_2	X_3	C
1	0	0	0
0	1	1	0
1	0	1	1
1	1	1	1

Answer:

$$P(C = 1|X_1 = 0, X_2 = 1, X_3 = 0) \cong$$
$$\frac{P(C = 1)}{P(X_1 = 0, X_2 = 1, X_3 = 0)} \cdot P(X_1 = 0|C = 1) \cdot P(X_2 = 1|C = 1) \cdot P(X_3 = 0|C = 1)$$
$$\cong \frac{1/2}{P(X_1 = 0, X_2 = 1, X_3 = 0)} \cdot \frac{0+1}{2+2} \cdot \frac{1+1}{2+2} \cdot \frac{0+1}{2+2}$$
$$= \frac{1/2}{P(X_1 = 0, X_2 = 1, X_3 = 0)} \cdot \frac{1}{4} \cdot \frac{2}{4} \cdot \frac{1}{4}$$

[4] Consider scikit-learn (http://scikit-learn.org/stable/), Weka (http://www.cs.waikato.ac.nz/ml/weka/), LIBSVM (http://www.csie.ntu.edu.tw/~cjlin/libsvm/).

[5] In *single-label* multi-class classification (often also called simply multi-class classification), micro-averaged precision, micro-averaged recall, and micro-averaged F1 are all equal to accuracy. Check https://towardsdatascience.com/multi-class-metrics-made-simple-part-ii-the-f1-score-ebe8b2c2ca1.

$$P(C=0|X_1=0,X_2=1,X_3=0) \cong$$
$$\frac{P(C=0)}{P(X_1=0,X_2=1,X_3=0)} \cdot P(X_1=0|C=0) \cdot P(X_2=1|C=0) \cdot P(X_3=0|C=0)$$
$$\cong \frac{1/2}{P(X_1=0,X_2=1,X_3=0)} \cdot \frac{1+1}{2+2} \cdot \frac{1+1}{2+2} \cdot \frac{1+1}{2+2}$$
$$= \frac{1/2}{P(X_1=0,X_2=1,X_3=0)} \cdot \frac{2}{4} \cdot \frac{2}{4} \cdot \frac{2}{4}$$

Hence, the decision will be C = 0.

13. [Optional] Let <X_1, X_2, X_3, X_4> = <b, d, b, a> be the feature vector of an object to be classified. We use a multivariate Bernoulli Naive Bayes classifier, with three classes (C = 1, C = 2, C = 3), and the training data of the table. What will be the decision of the classifier? Use Laplace estimates for $P(X_i|C)$. Each feature X_i has four possible values: a, b, c, d.

X_1	X_2	X_3	X_4	C
a	b	b	a	1
b	a	a	b	1
b	b	a	b	1
a	a	b	b	2
a	b	b	b	2
b	a	b	a	2
c	d	d	c	3
d	c	c	d	3
d	d	c	d	3

Answer: From the training data, we see that $P(C=1) = P(C=2) = P(C=3) = 3/9$*, i.e., the priori probabilities do not influence the decision.*

Using Laplace estimates, we obtain:

$$P(X_1=b|C=1) = \frac{2+1}{3+4} = \frac{3}{7} \qquad P(X_2=d|C=1) = \frac{0+1}{3+4} = \frac{1}{7}$$
$$P(X_3=b|C=1) = \frac{1+1}{3+4} = \frac{2}{7} \qquad P(X_4=a|C=1) = \frac{1+1}{3+4} = \frac{2}{7}$$

and:

$$P(X_1=b|C=2) = \frac{1+1}{3+4} = \frac{2}{7} \qquad P(X_2=d|C=2) = \frac{0+1}{3+4} = \frac{1}{7}$$
$$P(X_3=b|C=2) = \frac{3+1}{3+4} = \frac{4}{7} \qquad P(X_4=a|C=2) = \frac{1+1}{3+4} = \frac{2}{7}$$

and:

$$P(X_1=b|C=3) = \frac{0+1}{3+4} = \frac{1}{7} \qquad P(X_2=d|C=3) = \frac{2+1}{3+4} = \frac{3}{7}$$
$$P(X_3=b|C=3) = \frac{0+1}{3+4} = \frac{1}{7} \qquad P(X_4=a|C=3) = \frac{0+1}{3+4} = \frac{1}{7}$$

Hence:

$$P(X_1=b|C=1) \cdot P(X_2=d|C=1) \cdot P(X_3=b|C=1) \cdot P(X_4=a|C=1) = \frac{3 \cdot 1 \cdot 2 \cdot 2}{7 \cdot 7 \cdot 7 \cdot 7}$$
$$P(X_1=b|C=2) \cdot P(X_2=d|C=2) \cdot P(X_3=b|C=2) \cdot P(X_4=a|C=2) = \frac{2 \cdot 1 \cdot 4 \cdot 2}{7 \cdot 7 \cdot 7 \cdot 7}$$
$$P(X_1=b|C=3) \cdot P(X_2=d|C=3) \cdot P(X_3=b|C=3) \cdot P(X_4=a|C=3) = \frac{1 \cdot 3 \cdot 1 \cdot 1}{7 \cdot 7 \cdot 7 \cdot 7}$$

Therefore, the decision is $C = 2$.

14. [Optional] We use a multivariate Bernoulli Naive Bayes classifier, with two classes ($C = 0$, $C = 1$) and m Boolean (binary) features $X_1, \ldots, X_m$, which decides $C = 1$ if:

$$P(C = 1) \cdot \prod_{i=1}^{m} P(X_i = x_i | C = 1) \geq K$$

and $C = 0$ otherwise. K is a constant. Show that this classifier is a linear separator. Hint: If we represent by t_i the outcome denoted by $X_i = 1$ (e.g., the occurrence of a particular word), then:

$$P(X_i = x_i | C = 1) = P(t_i | C = 1)^{x_i} \cdot [1 - P(t_i | C = 1)]^{1 - x_i}$$

Recall, also, that $\log(a \cdot b) = \log a + \log b$ and $\log a^b = b \cdot \log\ a$.

Answer: The classifier decides C = 1 if and only if (iff):

$$log[P(C = 1) \cdot \prod_{i=1}^{m} P(X_i = x_i | C = 1)]\ \geq log\ K$$

which is equivalent to:

$$log\ P(C = 1) + \sum_{i=1}^{m} log\ P(X_i = x_i | C = 1) \geq log\ \ K$$

and:

$$log\ P(C = 1) + \sum_{i=1}^{m} log\{P(t_i | C = 1)^{x_i} \cdot [1 - P(t_i | C = 1)^{1 - x_i}]\} \geq log\ \ K$$

and:

$$log\ P(C = 1) + \sum_{i=1}^{m} x_i \cdot log\ P(t_i | C = 1) + (1 - x_i) \cdot log[1 - P(t_i | C = 1)] \geq log\ \ K$$

Setting $K_1 = log\ P(C = 1)$, $K_{2,i} = log\ P(t_i | C = 1)$, $K_{3,i} = log\ P[1 - P(t_i | C = 1)]$, $K_4 = log\ K$, *the previous criterion becomes:*

$$K_1 + \sum_{i=1}^{m} [x_i \cdot K_{2,i} + (1 - x_i) \cdot K_{3,i}] \geq K_4$$

which is equivalent to:

$$K_1 + \sum_{i=1}^{m} K_{3,i} - K_4 + \sum_{i=1}^{m} x_i \cdot (K_{2,i} - K_{3,i})\ \geq 0$$

Setting $w_0 = K_1 + \sum_{i=1}^{m} K_{3,i} - K_4$ *and* $w_i = (K_{2,i} - K_{3,i})$, *the criterion for C = 1 becomes:*

$$w_0 + \sum_{i=1}^{m} w_i \cdot x_i\ \geq 0$$

Hence, the classifier is a linear separator.

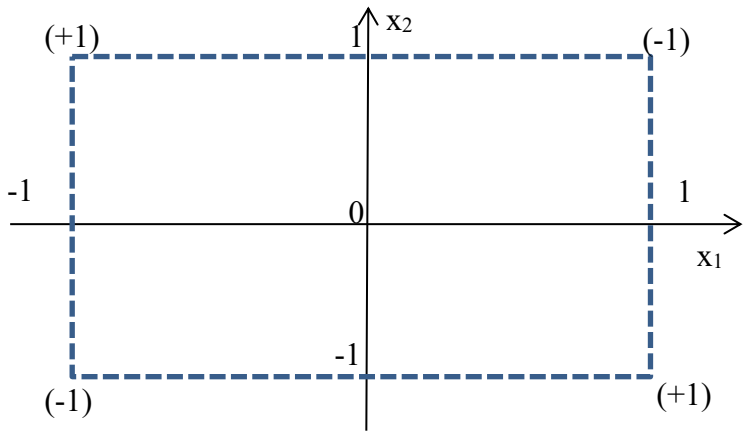
15. [Optional] Show that an SVM can compute the XOR function. Use 1, –1 to represent *true* and *false*, respectively. For example, given the input $< -1, 1 >$, the output should be 1; and given $< -1, -1 >$, the output should be –1. Assume that the SVM transforms each input $\vec{x} = < x_1, x_2 >$ of the original vector space to $\vec{F}(\vec{x}) = < x_1, x_1 x_2 >$ (in the new vector space). Plot the points $< x_1, x_1 x_2 >$ of the new vector space that correspond to all the possible inputs $< x_1, x_2 >$ of XOR. Draw the separator (straight line) that the SVM would learn in the new

vector space. What is the margin in the new vector space? Draw the corresponding separating lines in the original vector space. *(Simplified exercise 20.12 of Russel & Norvig, 2nd Edition.)*

Answer: The truth table of XOR is the following.

x_1	x_2	XOR(x_1, x_2)
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1

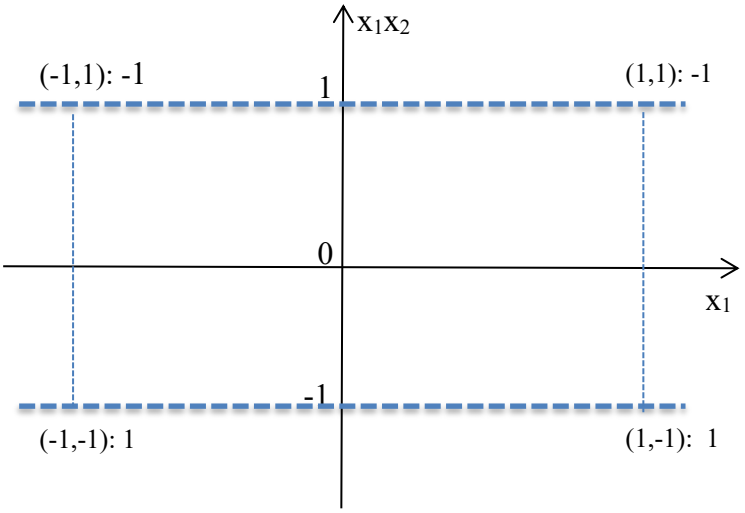
We wish to separate the two input vectors <x_1, x_2> with positive XOR(x_1, x_2) from the two input vectors with negative XOR(x_1, x_2). In the original vector space, the four vectors are not linearly separable, as shown below:



We extend the truth table to include the new feature $x_1 x_2$ of $\vec{F}(\vec{x}) = < x_1, x_1 x_2 >$:

x_1	x_2	x_1x_2	XOR(x_1, x_2)
-1	-1	1	-1
-1	1	-1	1
1	-1	-1	1
1	1	1	-1

The transformed four vectors $< x_1, x_1 x_2 >$ are linearly separable in the new vector space, as shown below. The transformed vectors with negative XOR(x_1, x_2) are now on the half-plane $x_1 x_2 \geq 1$ and the transformed vectors with positive XOR(x_1, x_2) are now on the half-plane $x_1 x_2 \leq -1$. The maximum margin linear separator is the straight line $x_1 x_2 = 0$ and its margin is 2.



The half-plane $x_1 x_2 \geq 0$ (negative class) of the new vector space corresponds to the shaded areas of the original vector space. The half-plane $x_1 x_2 \leq 0$ (positive class) of the new vector space corresponds to the rest of the original vector space.