CSE440: Natural Language Processing II

Dr. Farig Sadeque
Associate Professor
Department of Computer Science and Engineering
BRAC University

Lecture 3: ML Essentials

Topics

- Classification
- Feature representation
- Probability
- Naive Bayes
- ML evaluation

Classification

- Let’s see an example from “*Where’s My Mom?* by Julia Donaldson and Axel Scheffler ISBN-13: 978-0803732285”
- Context: a baby monkey is looking for his mom, and cannot find her anywhere
- A butterfly comes to help, asks the baby monkey how his mother looks like
- And the story starts

Who is the classifier here? What are the features? What is a feature?

Classifier

- Classifier = butterfly
- Features
  - bigger than baby
  - not a great gray hunk
  - no tusks, trunk
  - knees not baggy
  - tail coils around trees
  - doesn’t slither, hiss
  - no nest of eggs
  - legs > 0

Classifier

- Objects are described by properties or features
- Classifiers make predictions based on properties
- Good classifiers consider many properties jointly
- Classifiers may overfit, i.e., perform well on training data, but poorly on unseen data

We will mostly use discriminative classifier in this course– where the classifier function tries to draw (or predicts) a border to separate different types of data.

Classifier

- A classifier needs to be trained
- We train a classifier on a set of data we call training data
- We try to *fit* the classifier as best to our ability to the training data
  - Should we do this? Will learn soon
- Then we try to predict the class of a new data that is not seen during training
  - This is the part where prediction comes in

Formal definitions

A classifier, $h \in O \rightarrow Y$, is a function that maps an input object, $o \in O$ to an output category, $y \in Y$.

- Example: a student retention classifier, where
  $O = all\ UA\ students$ and
  $Y = \{will\text{-}graduate, won't\text{-}graduate\}$

A feature function, f_i, maps an input object to a (numeric) feature value representing some property of the object.

- Example: GPA feature, $f_{GPA}(Steven\ Bethard) = 4.0$

A feature vector is a representation of input data, where each element of the vector is the value of one feature:

$x = [f_1(o), f_2(o), \ldots, f_m(o)]$

Classification in matrix form

Most classification algorithms are trained on a large sample of input objects, $\hat{O}$, whose output categories are known:

$$\textit{training data} = \{(o_i, y_i) : o_i \in \hat{O}, y_i \in Y\}$$

If we have m feature functions, then we usually view this dataset as an $\mathbf{X}$ matrix and a Y vector:

$$\overbrace{\begin{bmatrix} f_1(o_1) & f_2(o_1) & f_3(o_1) & \cdots & f_m(o_1) \\ f_1(o_2) & f_2(o_2) & f_3(o_2) & \cdots & f_m(o_2) \\ f_1(o_3) & f_2(o_3) & f_3(o_3) & \cdots & f_m(o_3) \\ \vdots & \vdots & \ddots & \vdots & \\ f_1(o_n) & f_2(o_n) & f_3(o_n) & \cdots & f_m(o_n) \end{bmatrix}}^{\mathbf{X}} \quad \overbrace{\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix}}^{Y}$$

Classifier training will try to construct h such that $h(\mathbf{X}) = Y$.

Features are easy for regular ML task

- Like you have seen in the butterfly classifier, or other examples
- What features do we use when we are trying to classify a natural language data?
- Let’s see an example: BoW features

Bag-of-words features

- A bag-of-words feature representation means that for each word in the vocabulary, there is a feature function, f_i that produces the count of word i in the text.
- Example: f_{great} (great scenes great film) = 2

Documents	*Features*								
	f_{and}	f_{boxing}	f_{film}	f_{great}	f_{plot}	f_{satire}	f_{scenes}	f_{twists}	f_{worst}
worst boxing scenes	0	1	0	0	0	0	1	0	1
satire and great plot twists	1	0	0	1	1	1	0	1	0
great scenes great film	0	0	1	2	0	0	1	0	0

Bag-of-words features

How do we do it?

- Very easy
- Create a set of all possible unique words in the train data, w
- For each of the sentences create a vector v of size |w| with every element initialized to 0
- If a word i appears in that sentence, replace the 0 in v_i with the count of that word in that sentence
- That’s it

What are the issues with BoW?

- One feature function per word → large, sparse matrices
  - What's wrong with sparsity?
- Completely ignores word order
- But often still useful

Classwork

Construct the feature matrix for these four text messages:

- Sorry I’ll call later
- U can call me now
- U have won call now!!
- Sorry! U can not unsubscribe

Classwork

	$f_!$	$f_{'ll}$	f_I	f_{Sorry}	f_U	f_{call}	f_{can}	f_{have}	f_{later}	f_{me}	f_{not}	f_{now}	$f_{unsubscribe}$	f_{won}
Sorry I'll call later	0	1	1	1	0	1	0	0	1	0	0	0	0	0
U can call me now	0	0	0	0	1	1	1	0	0	1	0	1	0	0
U have won call now!!	2	0	0	0	1	1	0	1	0	0	0	1	0	1
Sorry! U can not unsubscribe	1	0	0	1	1	0	1	0	0	0	1	0	1	0

What now?

- So, we have learned what features are
- Now, how are we going to use these features to classify a natural language data?
- Let’s see one of the easiest types of classifier– it uses probability

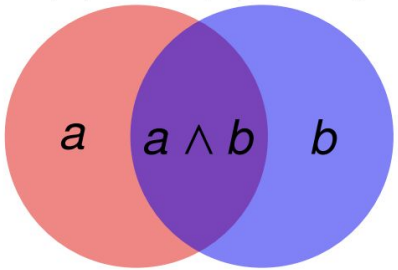
Probability review

$P(a)$ is our degree of belief in proposition a

- $P(SkyInTucson = sunny) = 0.78$

Probability rules:

$$0 \leq P(a) \leq 1$$
$$P(true) = 1$$
$$P(false) = 0$$
$$P(a \vee b) = P(a) + P(b) - P(a \wedge b)$$



Prior Probability

- The unconditional or prior probability of a proposition a is the degree of belief in that proposition given no other information

- P(DieRoll = 5) = 1/6
- P(CardDrawn = A♠) = 1/52
- P(SkyInTucson = sunny) = 286/365

Joint Probability

- The joint probability of propositions a1, . . . , an is the degree of belief in the proposition $a_1 \wedge \ldots \wedge a_n$

- P(A,♠) = P(A ∧ ♠) = 1/52

Conditional Probability

- The posterior or conditional probability of a proposition a given a proposition b is the degree of belief in a, given that we know only b

- P(Card = A♠|CardSuit = ♠) = 1/13
- P(DieRoll2 = 5|DieRoll1 = 5) =

Relation between Joint and Conditional

Product Rule

- $P(a \wedge b) = P(a|b)P(b)$ or $P(a|b) = P(a \wedge b)/P(b)$

- $P(A \wedge \spadesuit) = 1/52$
- $P(A|\spadesuit) = 1/13$
- $P(\spadesuit) = 1/4$
- $P(A|\spadesuit)P(\spadesuit) = 1/13 \cdot 1/4 = 1/52$

Bayes' Rule

$P(b|a) = P(a|b)P(b)/P(a)$

Let's derive this equation.

Purpose of Bayes' rule: Swap the conditioned and conditioning variables

Conditional independence

- Naïve Bayes classifier assumes conditional independence
- Two events A and B are conditionally independent given an event C if

$$P(A \cap B|C)=P(A|C)P(B|C)$$

- A more general version:

$$P(A_1\, A_2\, A_3 \ldots A_n|C) = \prod P(A_i|C)$$

Probabilistic classifiers

A probabilistic classifier, h, predicts the category of an object, o, as the most likely category given the object:

$$h(o) = \text{argmax}_{y \in Y} P(y|o)$$

Applying Bayes' rule yields:

$$h(o) = \text{argmax}_{y \in Y} \frac{P(o|y)P(y)}{P(o)}$$

And since $P(o)$ does not depend on y:

$$h(o) = \text{argmax}_{y \in Y} P(o|y)P(y)$$

Applying feature functions:

$$h(o) = \text{argmax}_{y \in Y} P(\begin{bmatrix} f_1(o) & f_2(o) & \cdots & f_m(o) \end{bmatrix} | y) \cdot P(y)$$

Naïve Bayes classifiers

A naïve Bayes classifier is a probabilistic classifier that assumes that the features are independent given the class:

$$
\begin{aligned}
&P(\left[\begin{array}{cccc} f_1(o) & f_2(o) & \cdots & f_m(o) \end{array}\right] | y) \\
&\quad = P(f_1(o)|y) \cdot P(f_2(o)|y) \cdot \ldots \cdot P(f_m(o)|y) \\
&\quad = \prod_{i=1}^{m} P(f_i(o)|y)
\end{aligned}
$$

A naïve Bayes classifier thus makes predictions as:

$$h(o) = \underset{y \in Y}{\operatorname{argmax}} P(y) \prod_{i=1}^{m} P(f_i(o)|y)$$

Let’s classify

- Or, in other words, train a classifier on train data and predict the class of a new data
- We will need:
  - Prior probability of a class y; P(y)
  - And conditional probability of each feature f_i $P(f_i|y)$
- Where do we get these probabilities from?
  - From the training data
  - This is the training phase
- Recall from the equation in previous slide:
  - We multiply all these probabilities to get the final probability of a test data
  - This is the test phase

Let's clear this up with an example

$$h(o) = \text{argmax}_{y \in Y} P(y) \prod_{i=1}^{m} P(f_i(o)|y)$$

Training data:	$f_!$	$f_{'ll}$	f_I	f_{Sorry}	f_U	f_{call}	f_{can}	f_{have}	f_{later}	f_{me}	f_{now}	f_{won}
+ *Sorry I'll call later*	0	1	1	1	0	1	0	0	1	0	0	0
+ *U can call me now*	0	0	0	0	1	1	1	0	0	1	1	0
− *U have won call now!!*	2	0	0	0	1	1	0	1	0	0	1	1
Testing data:												
Sorry! U can not unsubscribe	1	0	0	1	1	0	1	0	0	0	0	0

$$P(+|o) \propto P(+)P(f_!|+)P(f_{Sorry}|+)P(f_U|+)P(f_{can}|+)$$
$$P(-|o) \propto P(-)P(f_!|-)P(f_{Sorry}|-)P(f_U|-)P(f_{can}|-)$$

We have faced our first issue!

- Multiplying probability always has this type of issue: if a probability is 0, the entire multiplication becomes 0
- What should we do?
- We use a technique called smoothing

$$P(f_i(o) = k|y) = \left(\frac{1 + \text{count of } i \text{ with } y}{m + \text{count of any with } y} \right)$$

m = number of unique words in the training data

Let's try to do this example again, but now with smoothing

$$h(o) = \text{argmax}_{y \in Y} P(y) \prod_{i=1}^{m} P(f_i(o)|y)$$

$$P(f_i(o) = k|y) = \left(\frac{1 + \text{count of } i \text{ with } y}{m + \text{count of any with } y} \right)$$

Training data:	$f_!$	$f_{'ll}$	f_l	f_{Sorry}	f_U	f_{call}	f_{can}	f_{have}	f_{later}	f_{me}	f_{now}	f_{won}
+ *Sorry I'll call later*	0	1	1	1	0	1	0	0	1	0	0	0
+ *U can call me now*	0	0	0	0	1	1	1	0	0	1	1	0
– *U have won call now!!*	2	0	0	0	1	1	0	1	0	0	1	1
Testing data:												
Sorry! U can not unsubscribe	1	0	0	1	1	0	1	0	0	0	0	0

$$P(+|o) \propto P(+)P(f_!|+)P(f_{Sorry}|+)P(f_U|+)P(f_{can}|+)$$

$$P(-|o) \propto P(-)P(f_!|-)P(f_{Sorry}|-)P(f_U|-)P(f_{can}|-)$$

After the calculation

$$P(+|o) \propto \frac{2}{3} \cdot \frac{1}{22} \cdot \frac{2}{22} \cdot \frac{2}{22} \cdot \frac{2}{22} \approx 0.000023$$

$$P(-|o) \propto \frac{1}{3} \cdot \frac{3}{19} \cdot \frac{1}{19} \cdot \frac{2}{19} \cdot \frac{1}{19} \approx 0.000015$$

$$\Rightarrow \quad h(o) = +$$

Refining features

- Binarization
- Binary multinomial naïve Bayes assumes that what matters is whether or not a word occurs in a document, not its frequency in the document.
- Equivalent to removing duplicate words in each document.

$f_!$	$f_{'ll}$	f_I	f_{Sorry}
0	1	1	3
0	0	0	0
2	0	0	0
1	0	0	1

$\Rightarrow$

$f_!$	$f_{'ll}$	f_I	f_{Sorry}
0	1	1	1
0	0	0	0
1	0	0	0
1	0	0	1

Refining features

- N-gram features
- Instead of one word at a time, consider multiple words
- Bag-of-words assumption is sometimes too simplistic:
  - good vs. not good
  - like vs. didn't like
- A bag-of-n-grams feature representation has a vocabulary that includes phrases (up to) n tokens long.
  - Example: $f_{\text{great film}}$ (great scenes great film) = 1
  - This is bag-of-bigrams feature
- bag-of-words = bag-of-1-grams (a.k.a. unigrams)
- Consequences:
  - Vocabulary size grows quickly as n increases
  - Small counts; most n-grams never seen for large n
- For word n-grams, typically $n \leq 3$
- For character n-grams, typically $n \leq 10$

Refining features

- Lexicon features
- Sometimes there isn't enough training data.
  - Example: sentiment training data that doesn't include wretched, dreadful, genial, etc.
- A lexicon is a list of words (not sentences or documents) that have been annotated for a task.
  - Example: the LIWC lexicon labels words for the categories Positive emotion, Family etc.
- Lexicons are typically used to produce count features.
  - Example: f_{posemo} (great scenes beautiful film) = 2 if
  - great and beautiful are tagged as posemo in the lexicon

Rule-based features

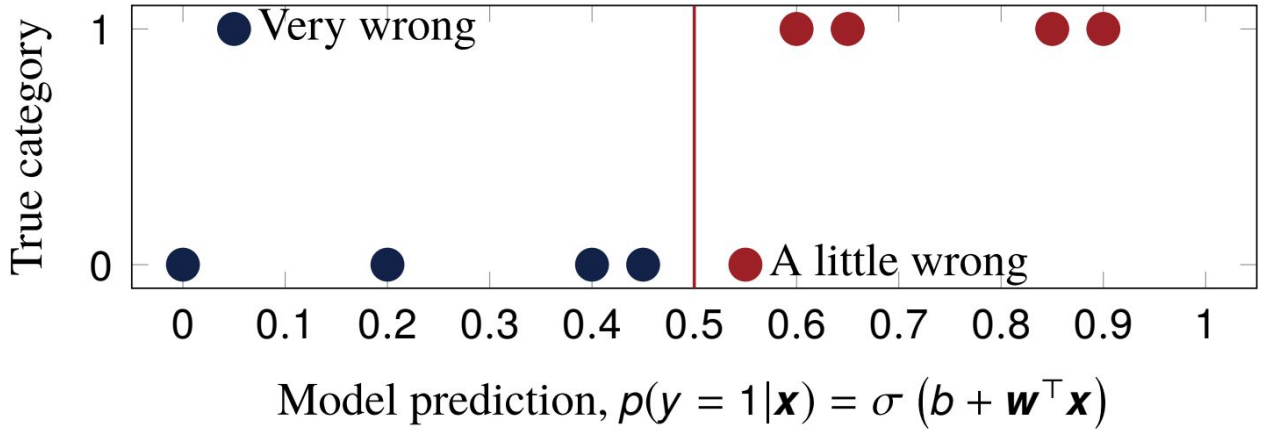
- There are no limits on the algorithm that a feature function may apply.
- Examples:
  - How many times does [$][\d,]{7} match?
  - What proportion of first line’s characters are capitals?
  - What is the ratio of text to image area (in HTML)?

Probabilistic learners are simple

- In everyday life, the learners we use make mistakes and then learn from mistakes
  - Let’s say we want to draw a line to separate spans vs. not spams (linear regression classifier)
  - The line splits the spams and not spams into two regions
  - If a spam falls in the non-spam region, the classifier made a mistake
  - This line is called a “decision boundary”
- How do we do that?
  - We can penalize the model for making mistakes
  - How can we penalize? Should we penalize the same amount for every mistake it makes?
  - Enter the idea of *Loss*

Loss

How "wrong" is this classifier?



Classification errors: $Cost_{0/1}(\boldsymbol{w}, b; \boldsymbol{X}, \boldsymbol{y}) = \frac{2}{10}$

But not all errors are equal

- Errors near the decision boundary are "less bad"

Cross entropy loss

Our estimated $p(y = 1|\boldsymbol{x}) = \sigma\left(b + \boldsymbol{w}^\top \boldsymbol{x}\right) = \hat{y}$ is good if:

- $\hat{y}$ is high when the true label is $y = 1$
- $\hat{y}$ is low when the true label is $y = 0$

So a good classifier would maximize:

$$p(y|\boldsymbol{x}) = \hat{y}^y (1 - \hat{y})^{(1-y)}$$

Or, equivalently, minimize:

$$\begin{aligned} L_{CE}(\boldsymbol{w}, b; \boldsymbol{x}, y) &= -\log p(y|\boldsymbol{x}) \\ &= -\log\left(\hat{y}^y (1 - \hat{y})^{(1-y)}\right) \\ &= -\left(y \log \hat{y} + (1 - y) \log(1 - \hat{y})\right) \\ &= -\big(y \log \sigma\left(b + \boldsymbol{w}^\top \boldsymbol{x}\right) + \\ &\qquad (1 - y) \log\left(1 - \sigma\left(b + \boldsymbol{w}^\top \boldsymbol{x}\right)\right)\big) \end{aligned}$$

Cross entropy loss

- This cross-entropy loss, $L_{CE}(\mathbf{w}, b; \mathbf{x}, y)$, measures how bad **w** and b are on the single example (**x**, y).
- An ML model runs through each of the training examples and tries to reduce this cross entropy loss as it goes along– and thus learns the pattern in the examples
- Sometimes, a model goes through all training examples multiple times to reduce the loss further
    - Each of these run is called an epoch

ML best practices

- Data splitting
- Underfitting and overfitting
- Performance metrics
- Statistical significance

Data splitting

- Train classifier on data E_{train}
- Test classifier on E_{test}

```
>>> from sklearn.feature_extraction.text import TfidfVectorizer
>>> from sklearn.model_selection import train_test_split
>>> corpus = [
... 'This is the first document.',
... 'This document is the second document.',
... 'And this is the third one.',
... 'Is this the first document?',
... ]
>>> vectorizer = TfidfVectorizer()
>>> X = vectorizer.fit_transform(corpus)
>>> y = [1, 0, 0, 1]
>>> X_train, X_test, y_train, y_test = train_test_split(
... corpus, y, test_size=0.25, random_state=42)
```

Data splitting

- Never peak into the test data!
  - Not even accidentally

Train:

x_1	x_2	$f(x)$
1	*a*	*true*
0	*b*	*false*
0	*c*	*false*
0	*b*	*false*

Test:

x_1	x_2	$f(x)$
1	*a*	*true*
0	*b*	*false*
0	*c*	*false*
1	*c*	*true*

$$h_1(x) = \begin{cases} true & \text{if } x_1 = 1 \\ false & \text{otherwise} \end{cases} \qquad h_2(x) = \begin{cases} true & \text{if } x_2 = a \\ false & \text{otherwise} \end{cases}$$

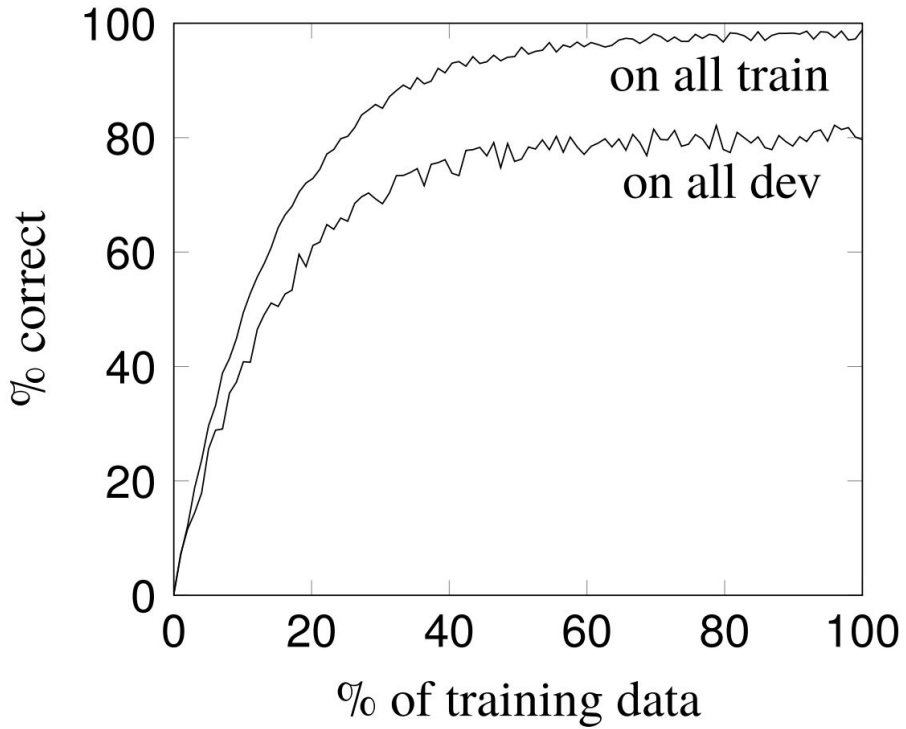
- What are the accuracies of these models?
- How do you improve them?

Data splitting: improving a model

- Bad idea
  - Train on the train data
  - Evaluate on test data
  - Go back to the training phase and tune parameters
- What will it do?
- Better idea
  - Split data into three parts
    - Train on the major chunk of the data (E_{train})
    - Tune on a very small piece of data (E_{val})
    - Test on another small chunk of data (E_{test})

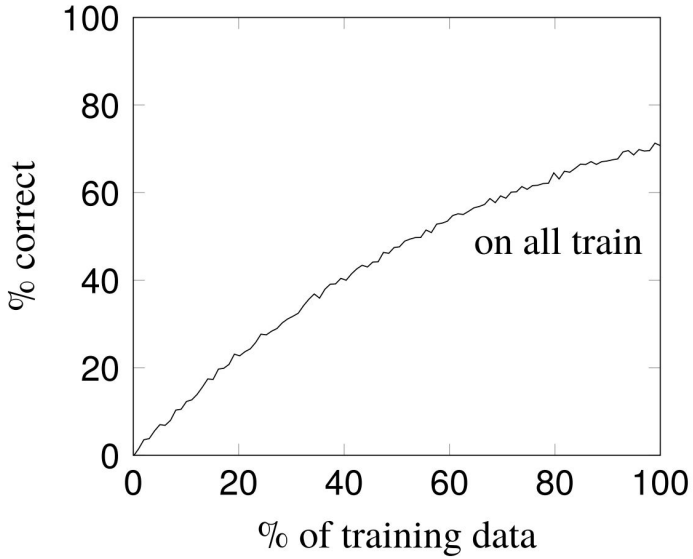
Underfitting and Overfitting

Learning Curve

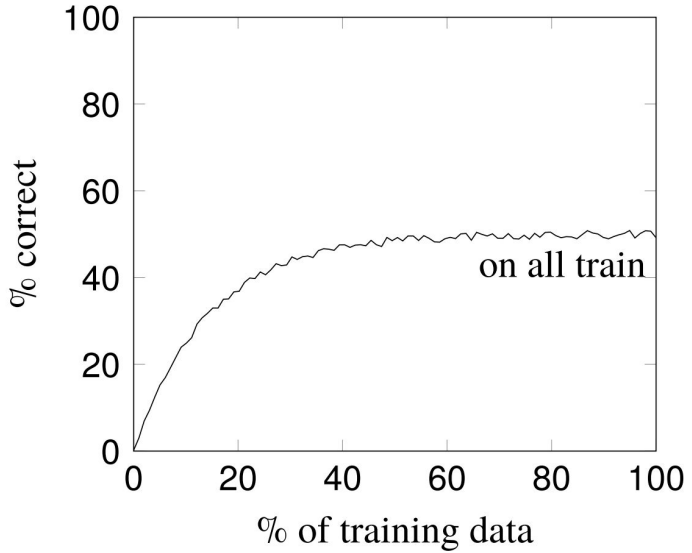


Underfitting

Insufficient training data



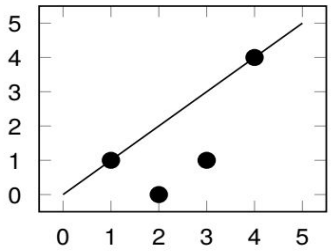
Model is too simple



Addressing underfitting

x_1	$f(x)$	$h(x)$
1	1	1
2	0	2
3	1	3
4	4	4

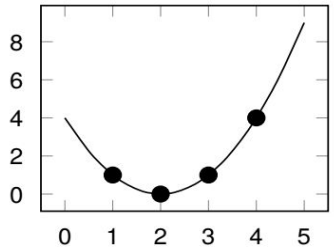
$h(x) = x_1$



More complex model

x_1	$f(x)$	$h(x)$
1	1	1
2	0	0
3	1	1
4	4	4

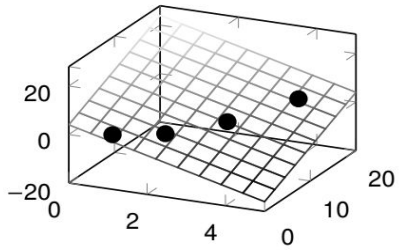
$h(x) = x_1^2 - 4x_1 + 4$



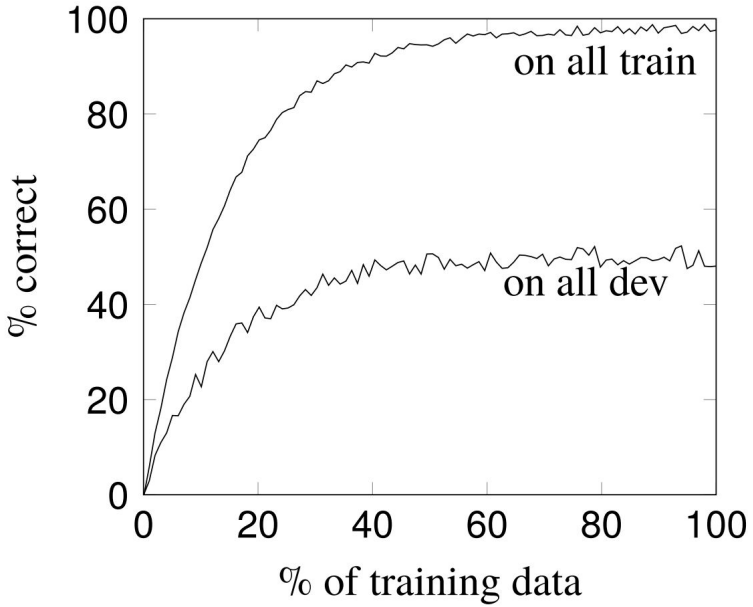
More features

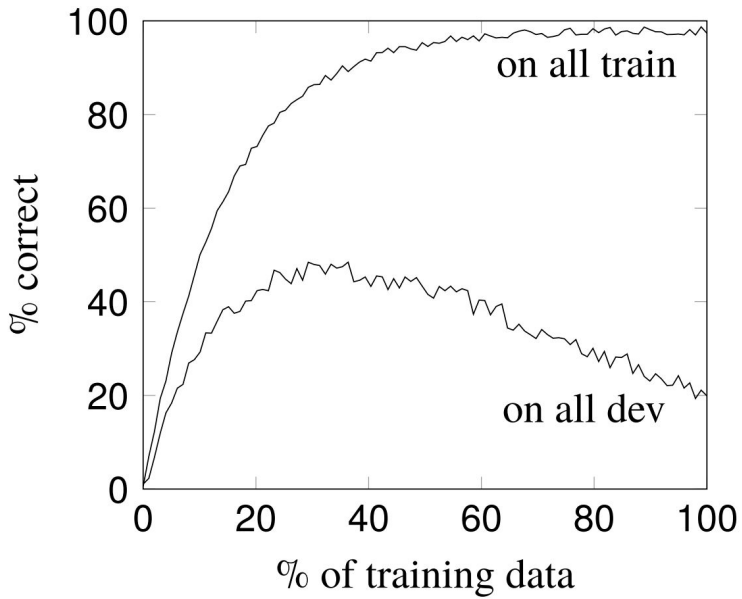
x_1	x_2	$f(x)$	$h(x)$
1	1	1	1
2	4	0	0
3	9	1	1
4	16	4	4

$h(x) = -4x_1 + x_2 + 4$



Overfitting

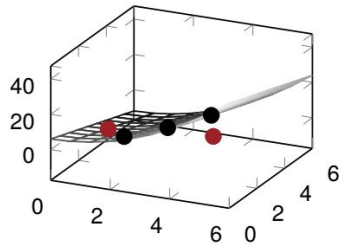




Addressing overfitting

x_1	x_2	$f(x)$	$h(x)$
2	1	7	7
3	2	9	9
4	3	13	13
1	2	3	1
5	1	15	28

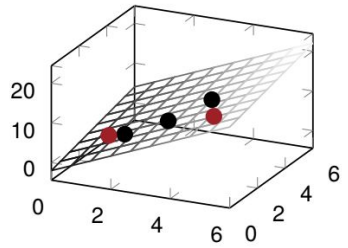
$h(x) = x_1^2 - 3x_2 + 6$



Simpler model

x_1	x_2	$f(x)$	$h(x)$
2	1	7	5
3	2	9	9
4	3	13	13
1	2	3	3
5	1	15	14

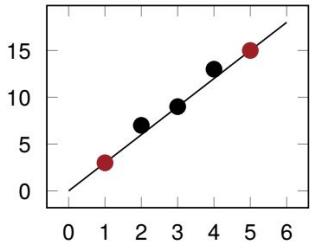
$h(x) = 3x_1 + x_2 - 2$



Fewer features

x_1	$f(x)$	$h(x)$
2	7	6
3	9	9
4	13	12
1	3	3
5	15	15

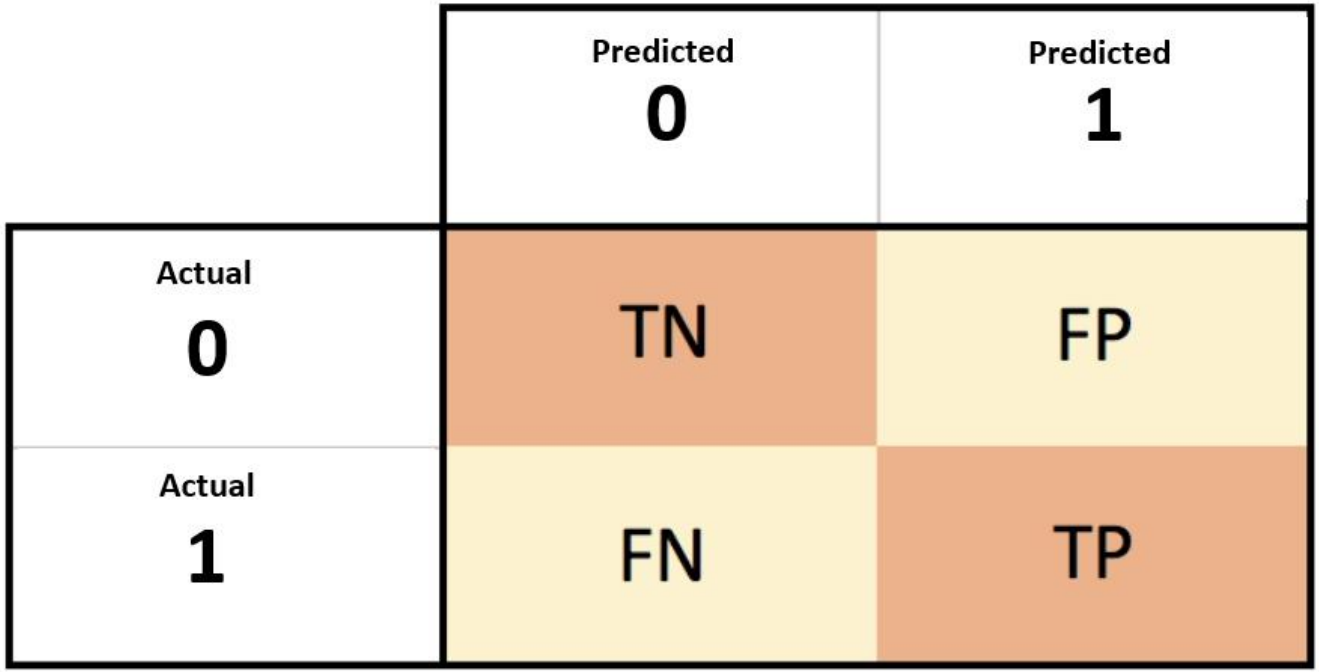
$h(x) = 3x_1$



Performance metrics

- Accuracy is a bad performance metrics
  - the distribution of categories is unbalanced, or
  - you care about one category more than the others
  - Example: detecting spam in text messages
  - Example: diagnosing depression from tweets
- What should we do?
- Use other metrics: precision, recall, F-1 score

Confusion matrix



Precision, recall and F-1 score

Precision = ratio between true + (or -) and predicted + (or -)

Recall = ration between true (or -) and actual + (or -)

F-1 score: Harmonic mean of precision and recall

There are other task-specific metrics as well:

- BLEU: machine translation
- WER: ASR
- Cosine similarity: semantic similarity
- Euclidean distance: spell checking
- F-latency: early detection

More than one category of interest

- Sometimes we care about more than one category in classification
  - In sentiment analysis we care about both positive and negative sentiments
- Use
  - Macro-average: our regular average
  - Micro-average: combine-and-calculate
- Let’s calculate macro- and micro-average precision for this case:

  We predicted 51 sentences as positive and and 42 as negative, where 37 of them were actually positive and 18 were actually negative

  - What is macro-precision?
  - What is the micro-precision?

Statistical Significance

- On some test set T, classifier A gets .992 F1, B gets .991 F1.
  - Is A actually better than B?
- Competing hypotheses:
  - A is generally better than B
  - A was better than B by chance on T, but would not be better than B in general, i.e., the null hypothesis (H_0)
  - We would like to reject the null hypothesis if we want to establish that A is actually better than B
- How to do this? One way can be:
  - We can do some sort of statistical significance test (Student's t-test) using multiple samples
  - If the test value is less than than a preset critical value (p-value), we reject the null hypothesis
  - How to get the samples? Many ways (e.g. cross-validation)