

LECTURE 6

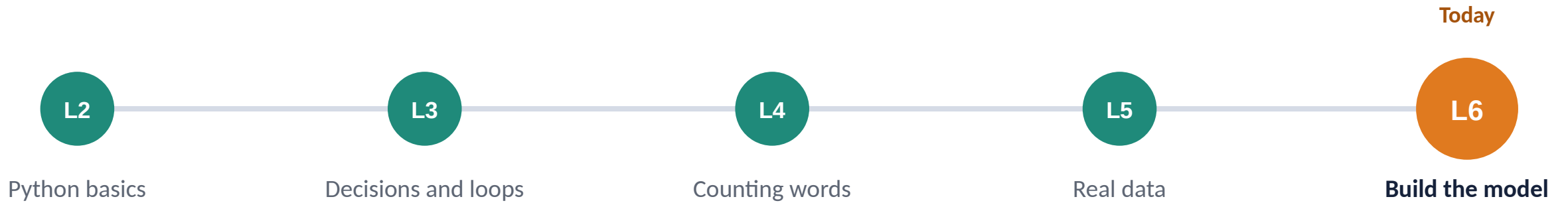
Build a real spam detector

From 5,169 messages to a working model, step by step

Today's key line of code

```
model = MultinomialNB()  
model.fit(X_train_counts, y_train)
```

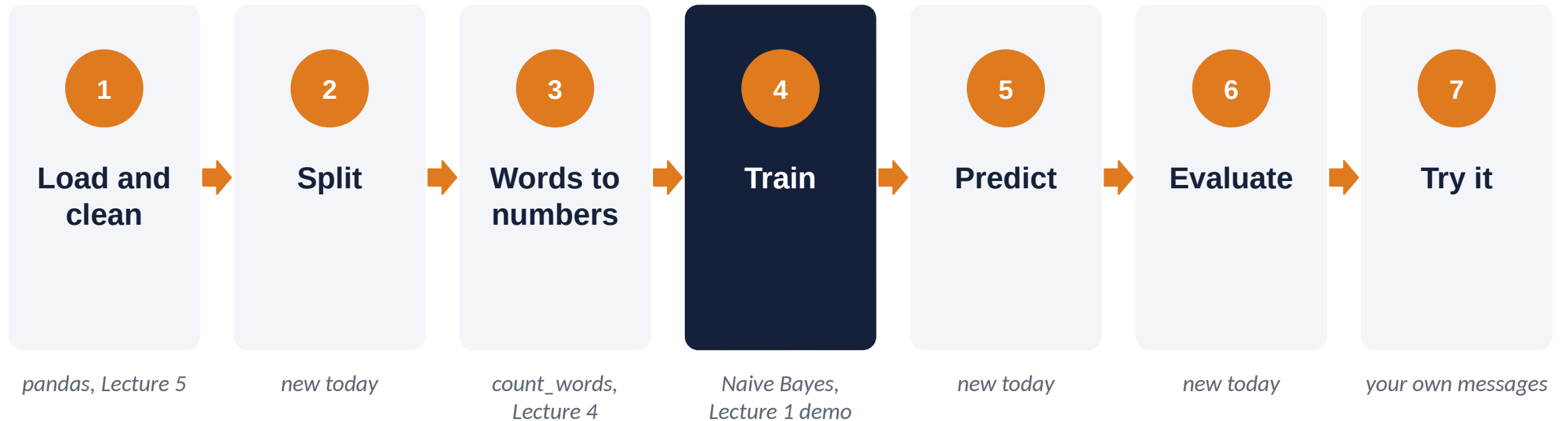
Where we are on the road to a spam detector



Today you will

- 1 Load and clean the data (Lecture 5)
- 2 Split into training and test sets
- 3 Turn words into numbers
- 4 Train a Naive Bayes model
- 5 Predict, evaluate honestly, and try your own messages

Seven steps from raw text to predictions



The same seven steps work for most classification problems: plant disease photos, speech emotion, fraud.

Step 1: Load and clean

- Same loading code as Lecture 5; in the lab use "sms.tsv" instead of url
- Drop duplicates so test messages are truly new
- 5,169 unique messages remain
- X = the messages (input), y = the labels (answer)

Code

```
import pandas as pd
url = ("https://raw.githubusercontent.com/"
      "justmarkham/pycon-2016-tutorial/"
      "master/data/sms.tsv")
df = pd.read_csv(url, sep="\t", header=None,
                 names=["label", "message"])
df = df.drop_duplicates()
X = df["message"]
y = df["label"]
print(df.shape)
```

Output

```
(5169, 2)
```

Step 2: Split into study and exam sets

- The model studies the training set (80%)
- We grade it on the test set (20%) it has never seen
- `stratify=y` keeps the same spam share in both
- `random_state=42` makes the split repeatable

Code

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42,
    stratify=y)
print(len(X_train), len(X_test))
```

Output

```
4135 1034
```

Step 3: What "words to numbers" means

Models only understand numbers. CountVectorizer builds a table: one row per message, one column per word, each cell a count.

Message	win	free	prize	now	lunch	class
Win a free prize, win now	2	1	1	1	0	0
Free lunch now	0	1	0	1	1	0
See you in class	0	0	0	0	0	1

This is count_words() from Lecture 4, done for every message and arranged as a table. Word order is lost: it is a “bag of words”.

Step 3: Words to numbers, in code

- **fit_transform** on training data: learn the vocabulary, then count
- **transform** on test data: count only, using that vocabulary
- 7,680 different words learned
- Never fit on the test set: that would peek at the exam

Code

```
from sklearn.feature_extraction.text import (
    CountVectorizer)

vec = CountVectorizer()
X_train_counts = vec.fit_transform(X_train)
X_test_counts = vec.transform(X_test)
print(X_train_counts.shape)
```

Output

```
(4135, 7680)
```

Step 3, up close: fit learns, transform reuses

- **fit** reads the training messages and builds the vocabulary
- **transform** counts words using that vocabulary only
- "tomorrow" was never seen in training, so it is ignored
- "a" is dropped: by default, one-letter words are skipped

Code

```
demo = CountVectorizer()  
demo.fit(["win a free prize", "see you in class"])  
print(demo.get_feature_names_out())  
print(demo.transform(["free class tomorrow"]).toarray())
```

Output

```
['class' 'free' 'in' 'prize' 'see' 'win' 'you']  
[[1 1 0 0 0 0 0]]
```

Step 4: Train the model

- Multinomial Naive Bayes: the model from Lecture 1's demo
- `fit` = learn from examples: counts words per label
- Training takes well under a second
- This is our `spam_score` idea, with proper probabilities

Code

```
from sklearn.naive_bayes import MultinomialNB

model = MultinomialNB()
model.fit(X_train_counts, y_train)
print("Training done!")
```

Output

```
Training done!
```

Step 5: Predict messages it has never seen

- `predict` labels all 1,034 test messages
- Put truth and prediction side by side
- Here 5 of 6 are right; one spam slipped through
- Next: count mistakes over the whole test set

Code

```
predictions = model.predict(X_test_counts)
check = pd.DataFrame({"message": X_test.str[:28],
                      "truth": y_test,
                      "predicted": predictions})
print(check.sample(6, random_state=42))
```

Output

	message	truth	predicted
2599	Okie... Thanx...	ham	ham
1207	As a SIM subscriber, you are	spam	spam
5147	Get your garden ready for su	spam	spam
3332	How much it will cost approx	ham	ham
788	Ever thought about living a	spam	ham
1972	Enjoy ur life. . Good night	ham	ham

Step 6: Evaluate, and compare with doing nothing

Code

```
from sklearn.metrics import accuracy_score

acc = accuracy_score(y_test, predictions)
baseline = (y_test == "ham").mean()
print(f"Model accuracy:  {acc:.1%}")
print(f"Always say ham:  {baseline:.1%}")
```

Output

```
Model accuracy:  98.2%
Always say ham:  87.3%
```

98.2%

Our model

87.3%

Always says "ham"

Error rate falls from 12.7% to 1.8%

Step 6: Which mistakes does it make?

Code

```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, predictions,
                       labels=["ham", "spam"]))
```

Output

```
[[898  5]
 [ 14 117]]
```

- Rows: the truth. Columns: the prediction.
- 5 normal messages wrongly flagged
- 14 of 131 spam missed: about 1 in 9
- Which mistake is worse for a user?

	Predicted ham	Predicted spam
Truly ham	898 Normal messages let through	5 False positive: real message sent to spam
Truly spam	14 False negative: spam slipped through	117 Spam caught

Watch it: from split to an honest score

WATCH FOR

- stratify keeps the spam share equal in both parts
- The test set stays hidden during training
- Each test message lands in one matrix cell
- Accuracy compared with always saying ham

MACHINE LEARNING, STEP BY STEP

From train/test split to an honest score

Why we hide some messages from the model,
and how the confusion matrix and accuracy are counted.

Click the video to play, click again to pause. Silent: talk the class through it.

Step 7: Try your own messages

- New text goes through the same `vec.transform`
- `predict_proba` gives [ham, spam] probabilities
- Classic prize spam: caught with near certainty
- A modern parcel scam slips through: it is not like 2011 spam

Code

```
tests = [  
    "Congratulations! You won a free prize. Call now!",  
    "Are we still meeting at the library tomorrow?",  
    "Your parcel is waiting, pay the delivery fee"]  
new_counts = vec.transform(tests)  
print(model.predict(new_counts))  
print(model.predict_proba(new_counts).round(2))
```

Output

```
['spam' 'ham' 'ham']  
[[0.    1.  ]  
 [1.    0.  ]  
 [0.83  0.17]]
```

Look inside: what did it learn?

- For each word: how much more likely in spam than ham
- Top clues: claim, prize, guaranteed, www
- "150p" is a UK price code, a sign of the data's origin
- Nobody typed these rules; they came from the data

Code

```
import numpy as np

words = vec.get_feature_names_out()
lp = model.feature_log_prob_
diff = lp[1] - lp[0]
top = np.argsort(diff)[-10:][::-1]
print(list(words[top]))
```

Output

```
['claim', 'prize', '150p', 'tone', 'co', 'www', '18',  
'guaranteed', '500', '1000']
```

Honest limits of our model

1

Old, foreign data

Mostly UK and Singapore messages from around 2011.
New scam styles are unseen.

2

One small test

1,034 test messages, one random split. Another split gives a slightly different score.

3

Words only

It ignores word order, the sender and links. "FR33" tricks can still work.

4

Mistakes have costs

A real message lost in the spam folder can matter more than one spam let through.

The whole pipeline, start to finish

Everything from today in one cell. This short program is a complete, real machine learning project.

Code

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score

url = ("https://raw.githubusercontent.com/justmarkham/"
       "pycon-2016-tutorial/master/data/sms.tsv")
df = pd.read_csv(url, sep="\t", header=None,
                 names=["label", "message"]).drop_duplicates()
X_tr, X_te, y_tr, y_te = train_test_split(
    df["message"], df["label"], test_size=0.2,
    random_state=42, stratify=df["label"])
vec = CountVectorizer()
model = MultinomialNB().fit(vec.fit_transform(X_tr), y_tr)
pred = model.predict(vec.transform(X_te))
print(f"Accuracy: {accuracy_score(y_te, pred):.1%}")
```

Output

Accuracy: 98.2%

Two classic machine learning mistakes

Read the last line of an error first. It names the problem.

What you typed	What Python says	What it means
<pre>model.predict(["Win cash now"])</pre>	<pre>ValueError: Expected 2D array, got 1D array instead:</pre>	The model needs numbers. Use <code>model.predict(vec.transform([...]))</code> .
<pre>vec2 = CountVectorizer() X_bad = vec2.fit_transform(X_test) model.predict(X_bad)</pre>	<pre>ValueError: X has 3320 features, but MultinomialNB is expecting 7680 features as input.</pre>	A new vocabulary no longer matches the model. Fit only on training data; use <code>vec.transform</code> for new text.

Interactive lab: Spam Model Lab

In the lab you can

1 Split

Change test_size, random_state and stratify; see every message.

2 Words to numbers

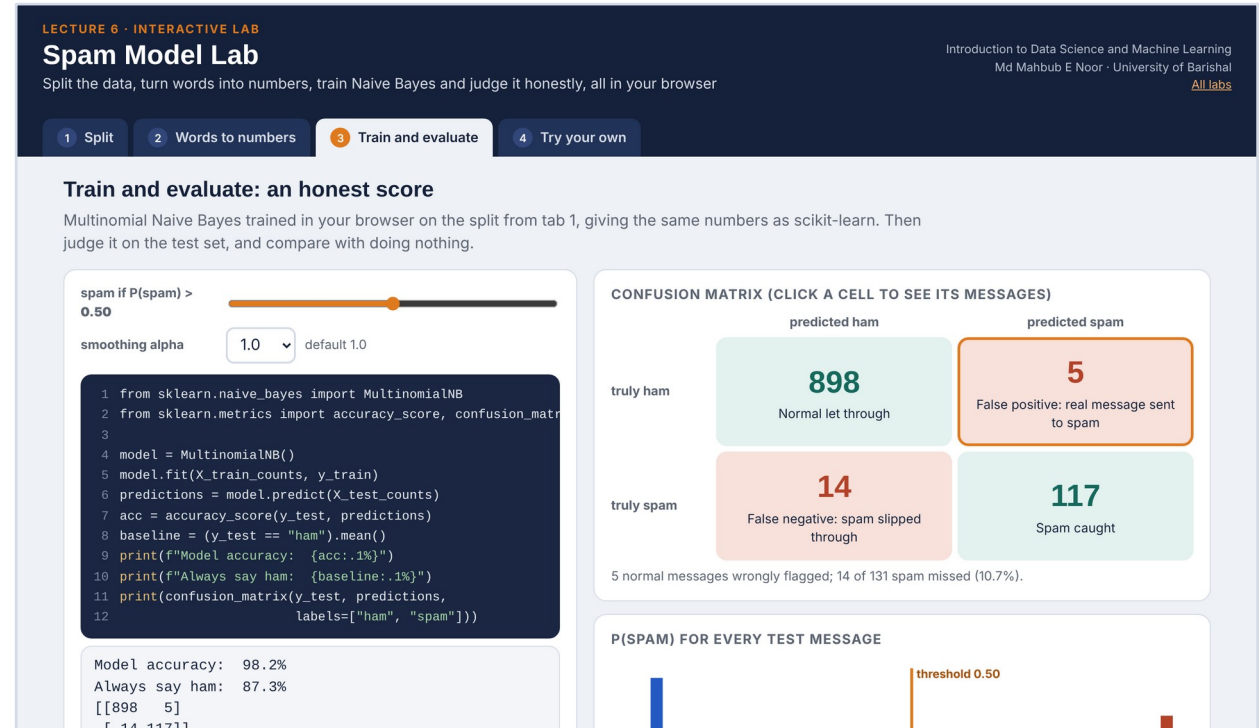
Build the bag-of-words table that CountVectorizer makes.

3 Train and evaluate

Confusion matrix you can click, and a threshold slider.

4 Try your own

Word-by-word evidence for any message you type.



Open the lab ►

Works offline in any browser

Labs_open_in_browser/
L6_Spam_Model_Lab.html

Homework 6: mini-project

Due: [date, about two weeks from today]

1 Rebuild it

Type the full pipeline in your own notebook, one step per cell, with a text cell explaining each step in your own words.

2 Baseline and mistakes

Report accuracy, the always-ham baseline, and the confusion matrix. Explain each cell of the matrix in one sentence.

3 Experiment

Try `CountVectorizer(stop_words="english")` and `random_state=7`. What changes? Keep a small table of results.

4 Fool the model

Write 3 spam messages designed to slip through. Did they? Explain why they worked or failed.

5 Local test

Collect 10 SMS (5 spam, 5 normal), remove all personal details, and test them. How many are right? Why might this differ from the test score?

6 Reflect

One paragraph: would you trust this filter on your own phone? What would you change first?

Submit on [LMS / Google Classroom]: your notebook file `HW6_<your ID>.ipynb`, or a Colab link set to "Anyone with the link can view".

Recap: Lecture 6 in 90 seconds

RECAP

Lecture 6 in 90 seconds

A short recap of today's lecture. Pause any time.

Click to play or pause. Watch it again at home to revise.