Machine Learning with Large Datasets

Naïve Bayes classification

Barnabás Póczos

Chain Rule & Bayes Rule

Chain rule:

$$P(X,Y) = P(X|Y)P(Y) = P(Y|X)P(X)$$

Bayes rule:

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}$$

Bayes rule is important for reverse conditioning.

The Naïve Bayes Classifier

The Bayes Classifier

Let $X = (X_1, \ldots, X_d)$ be a d-dimensional feature and $Y \in \{1, 2, \ldots, K\}$ its class label.

The Bayes classification rule:

$$\begin{aligned} f_{Bayes}(X) &= \arg\max_Y \; P(Y \mid X_1, \ldots, X_d) \\ &= \arg\max_Y \frac{P(X_1, \ldots, X_d \mid Y) P(Y)}{P(X_1, \ldots, X_d)} \\ &= \arg\max_Y P(X_1, \ldots, X_d \mid Y) P(Y) \end{aligned}$$

The difficulty is that we have to learn these probabilities from the training dataset.

Naïve Bayes Assumption

Naïve Bayes assumption: Features X_1 and X_2 are conditionally independent given the class label Y:

$$P(X_1, X_2|Y) = P(X_1|Y)P(X_2|Y)$$

More generally: $$P(X_1 ... X_d|Y) = \prod_{i=1}^{d} P(X_i|Y)$$

How many parameters to estimate on the l.h.s. & r.h.s?

(X is composed of d binary features, e.g. presence of word "earn" in a text. Y has K possible class labels)

$(2^d-1)K$ vs $(2-1)dK$

Naïve Bayes Classifier

Given:

- Class prior $P(Y)$
- d conditionally independent features $X_1,\ldots,X_d$ given the class label Y
- For each X_i, we have the conditional likelihood $P(X_i|Y)$

Decision rule:

$$\begin{aligned} f_{NB}(\mathbf{x}) &= \arg\max_y \; P(x_1,\ldots,x_d \mid y)P(y) \\ &= \arg\max_y \prod_{i=1}^{d} P_i(x_i|y)P(y) \end{aligned}$$

Naïve Bayes Algorithm for discrete features

Training Data: $\{(X^{(j)}, Y^{(j)})\}_{j=1}^{n}$ $\quad X^{(j)} = (X_1^{(j)}, \ldots, X_d^{(j)})$

n d dimensional features + class labels

$f_{NB}(\mathbf{x}) = \arg\max_y \prod_{i=1}^{d} P_i(x_i|y)P(y)$ **We need to estimate these probabilities!**

Estimate them with Relative Frequencies!

For Class Prior $\quad \widehat{P}(y) = \frac{\{\#j : Y^{(j)} = y\}}{n}$

For Likelihood $\quad \frac{\widehat{P}_i(x_i, y)}{\widehat{P}(y)} = \frac{\{\#j : X_i^{(j)} = x_i, Y^{(j)} = y\}/n}{\{\#j : Y^{(j)} = y\}/n}$

NB Prediction for test data: $X = (x_1, \ldots, x_d)$

$$Y = \arg\max_y \widehat{P}(y) \prod_{i=1}^{d} \frac{\widehat{P}_i(x_i, y)}{\widehat{P}(y)}$$

Subtlety: Insufficient training data

What if you never see a training instance where $X_1 = a$ when $Y = b$?

$$Y = \arg\max_y \widehat{P}(y) \prod_{i=1}^{d} \frac{\widehat{P}_i(x_i, y)}{\widehat{P}(y)}$$

For example,

there is no X_1='Earn' when Y='SpamEmail' in our dataset.

$$\Rightarrow P(X_1 = a, Y = b) = 0 \Rightarrow P(X_1 = a | Y = b) = 0$$

$$\boxed{\Rightarrow P(X_1 = a, X_2 ... X_d | Y = b) = P(X_1 = a | Y = b) \prod_{i=2}^{d} P(X_i | Y = b) = 0}$$

Thus, no matter what the values $X_2, \ldots, X_d$ take:

$$P(Y = b \mid X_1 = a, X_2, \ldots, X_d) = 0$$

What now???

Case Study: Text Classification

Case Study: Text Classification

- Classify e-mails
  - Y = {Spam,NotSpam}
- Classify news articles
  - Y = {what is the topic of the article?

What about the features **X**?

The text!

X_i represents i^{th} word in document

Article from rec.sport.hockey

```
Path: cantaloupe.srv.cs.cmu.edu!das-news.harvard.e
From: xxx@yyy.zzz.edu (John Doe)
Subject: Re: This year's biggest and worst (opinic
Date: 5 Apr 93 09:53:39 GMT

I can only comment on the Kings, but the most
obvious candidate for pleasant surprise is Alex
Zhitnik. He came highly touted as a defensive
defenseman, but he's clearly much more than that.
Great skater and hard shot (though wish he were
more accurate). In fact, he pretty much allowed
the Kings to trade away that huge defensive
liability Paul Coffey. Kelly Hrudey is only the
biggest disappointment if you thought he was any
good to begin with. But, at best, he's only a
mediocre goaltender. A better choice would be
Tomas Sandstrom, though not through any fault of
his own, but because some thugs in Toronto decided
```

NB for Text Classification

P(**X**|Y) is huge!!!

- An article is a list of 1000 words, $\mathbf{X}=\{X_1,\ldots,X_{1000}\}$
- X_i represents the i^{th} word in the document, i.e., the domain of X_i is entire vocabulary, e.g., Webster Dictionary (or more).
- K different topics (classes)
  $X_i \in \{1,\ldots,50000\} \Rightarrow K50000^{1000}$ parameters….

NB assumption helps a lot!!!

- $P(X_i=x_i|Y=y)$ is the probability of observing word x_i at the i^{th} position in a document on topic y $\Rightarrow$ 1000K(50000-1) parameters to learn

$$h_{NB}(\mathbf{x}) = \arg\max_y P(y) \prod_{i=1}^{LengthDoc=1000} P_i(x_i|y)$$

Bag of words model

Typical additional assumption – **Position in document doesn't matter**: $P(X_i=x_i|Y=y) = P(X_k=x_i|Y=y)$

- "Bag of words" model – order of words on the page ignored
- Sounds really silly, but often works very well! $\Rightarrow$ K50000 parameters

$$\prod_{i=1}^{LengthDoc} P_i(x_i|y) = \prod_{w=1}^{W} P(w|y)^{count_w}$$

$$Y = \arg\max_y \widehat{P}(y) \prod_{w=1}^{W} \widehat{P}(w|y)^{count_w}$$

When the lecture is over, remember to wake up the person sitting next to you in the lecture room.

Bag of words model

Typical additional assumption – **Position in document doesn't matter**: $P(X_i=x_i|Y=y) = P(X_k=x_i|Y=y)$

- "Bag of words" model – order of words on the page ignored
- Sounds really silly, but often works very well! $\Rightarrow$ K(50000-1) parameters

$$\prod_{i=1}^{LengthDoc} P_i(x_i|y) = \prod_{w=1}^{W} P(w|y)^{count_w}$$

$$Y = \arg\max_y \widehat{P}(y) \prod_{w=1}^{W} \widehat{P}(w|y)^{count_w}$$

in is lecture lecture next over person remember room sitting the the the to to up wake when you

Bag of words approach


the world of
TOTAL

All About The Company
Global Activities
Corporate Structure
TOTAL's Story
Upstream Strategy
Downstream Strategy
Chemicals Strategy
TOTAL Foundation
Homepage

all about the
company

Our energy exploration, production, and distribution operations span the globe, with activities in more than 100 countries.

At TOTAL, we draw our greatest strength from our fast-growing oil and gas reserves. Our strategic emphasis on natural gas provides a strong position in a rapidly expanding market.

Our expanding refining and marketing operations in Asia and the Mediterranean Rim complement already solid positions in Europe, Africa, and the U.S.

Our growing specialty chemicals sector adds balance and profit to the core energy business.

aardvark	0
about	2
all	2
Africa	1
apple	0
anxious	0
...	
gas	1
...	
oil	1
…	
Zaire	0

Twenty news groups results

Given 1000 training documents from each group
Learn to classify new documents according to which newsgroup it came from

comp.graphics	misc.forsale
comp.os.ms-windows.misc	rec.autos
comp.sys.ibm.pc.hardware	rec.motorcycles
comp.sys.mac.hardware	rec.sport.baseball
comp.windows.x	rec.sport.hockey
alt.atheism	sci.space
soc.religion.christian	sci.crypt
talk.religion.misc	sci.electronics
talk.politics.mideast	sci.med
talk.politics.misc	
talk.politics.guns	

Naïve Bayes: 89% accuracy

Multinomial Naïve Bayes

Naïve Bayes assumption

$$P(X_1, \ldots, X_d|Y) = \prod_{i=1}^{d} P(X_i|Y)$$

In multinomial Naïve Bayes we assume that

$$P(X_1, \ldots, X_d|Y) = \frac{(X_1 + \ldots + X_d)!}{X_1! \cdots X_d!} \prod_{i=1}^{d} (p_{i|Y})^{X_i} \quad s.t. \sum_{i=1}^{d} p_{i|Y} = 1$$

that is, the conditional distribution is multinomial.

For each class label Y, we can estimate the parameters $p_{1|Y}, \ldots p_{d|Y}$ from the training set.

Multinomial Naïve Bayes

Multinomial Naive Bayes Demo

In [1]:
```
import numpy as np
from sklearn.naive_bayes import MultinomialNB
```

First create input-output pairs: (X,y). In this case we will have 3 instances each having 4 features. The features have to be nonnegative. E.g. Count numbers

In [2]:
```
X=np.array([[20,20,31,32],[20,33,17,30],[10,12,13,15]]) # shape: num_of_instances * num_of_features
X
```

Out[2]:
```
array([[20, 20, 31, 32],
       [20, 33, 17, 30],
       [10, 12, 13, 15]])
```

In [3]:
```
y = np.array([0, 1, 1])# class labels of the 3 instances. Can be 0,1,2,... K
```

In [4]:
```
clf = MultinomialNB(alpha=0, class_prior=None, fit_prior=True)
```

Training a multinomial naive bayes classifier is only 1 line with sklearn:

In [5]:
```
clf.fit(X, y)
```

Out[5]:
```
MultinomialNB(alpha=0, class_prior=None, fit_prior=True)
```

Now let us test the classifier on a test instance:

In [6]:
```
Xtest=[13,10,19,20]
Xtest
```

Out[6]:
```
[13, 10, 19, 20]
```

In [7]:
```
print(clf.predict(Xtest)) # the predicted class label is 0
```

```
[0]
```

These are the predicted class probabilities:

Multinomial Naïve Bayes (continued)

In [8]:
```
clf.predict_proba(Xtest)
```

Out[8]:
```
array([[ 0.95422538,  0.04577462]])
```

Now let us see how to calculate these probabilities from scratch

First let us count the features for each class.

In [9]:
```
clf.feature_count_    # shape: num_of_classes * num_of_features
```

Out[9]:
```
array([[ 20.,  20.,  31.,  32.],
       [ 30.,  45.,  30.,  45.]])
```

normalize the rows to have sum 1

In [10]:
```
np.exp(clf.feature_log_prob_) #P(x_i|y) i=1,2,3,4
```

Out[10]:
```
array([[ 0.19417476,  0.19417476,  0.30097087,  0.31067961],
       [ 0.2       ,  0.3       ,  0.2       ,  0.3       ]])
```

Take the log of each entry to calculate the log probabilities

In [11]:
```
clf.feature_log_prob_ #logP(x_i|y) i=1,2,3,4
```

Out[11]:
```
array([[-1.63899671, -1.63899671, -1.20074178, -1.16899309],
       [-1.60943791, -1.2039728 , -1.60943791, -1.2039728 ]])
```

Now calculate the class priors. First count the number of each class in the training set

In [12]:
```
clf.class_count_
```

Out[12]:
```
array([ 1.,  2.])
```

normalize the class count to get the class prior

In [13]:
```
np.exp(clf.class_log_prior_)
```

Out[13]:
```
array([ 0.33333333,  0.66666667])
```

Multinomial Naïve Bayes (continued)

and the class log prior

```
In [14]: clf.class_log_prior_

Out[14]: array([-1.09861229, -0.40546511])
```

Now let us go back to the feature log probabilities that we already calculated:

```
In [15]: np.exp(clf.feature_log_prob_.T)

Out[15]: array([[ 0.19417476,   0.2        ],
                [ 0.19417476,   0.3        ],
                [ 0.30097087,   0.2        ],
                [ 0.31067961,   0.3        ]])
```

```
In [16]: clf.feature_log_prob_.T

Out[16]: array([[-1.63899671, -1.60943791],
                [-1.63899671, -1.2039728 ],
                [-1.20074178, -1.60943791],
                [-1.16899309, -1.2039728 ]])
```

Calculate the inner product of this matrix with the features of the test instance:

```
In [17]: np.dot(Xtest, clf.feature_log_prob_.T)

Out[17]: array([-83.89088004, -87.62119733])
```

If we add the class log prior to this we get the join log likelihood:

```
In [18]: joint_log_likelihood = clf.class_log_prior_ + np.dot(Xtest, clf.feature_log_prob_.T)
```

```
In [19]: joint_log_likelihood

Out[19]: array([-84.98949233, -88.02666244])
```

Multinomial Naïve Bayes (continued)

And from this the joint likelihood:

```
In [20]: np.exp(joint_log_likelihood)
Out[20]: array([  1.22894505e-37,   5.89530453e-39])
```

These numbers are pretty small... and all that left is to normalize them to get a prob distribution on the class labels:

```
In [21]: np.exp(joint_log_likelihood)/sum(np.exp(joint_log_likelihood))
Out[21]: array([ 0.95422538,  0.04577462])
```

These are indeed the same numbers that clf.predict_proba(Xtest) provided

The above normalization, however, can easily underflow because we need to deal with very small numbers. A better approach is to normalize in the log space

```
In [22]: log_normalizing_xtest = np.log(np.sum(np.exp(joint_log_likelihood)))

In [23]: log_normalizing_xtest
Out[23]: -84.942636938259014

In [24]: log_prob_pred_xtest=joint_log_likelihood - log_normalizing_xtest

In [25]: log_prob_pred_xtest
Out[25]: array([-0.04685539, -3.0840255 ])

In [26]: np.exp(log_prob_pred_xtest)
Out[26]: array([ 0.95422538,  0.04577462])
```

This is indeed the same as clf.predict_proba(Xtest) provides.

```
In [ ]:
```

Multinomial Naïve Bayes Application

Naive Bayes on MNIST dataset

Based on the code of Bikramjot Hanzra, CMU SCS

The first step is to download the handwritten image dataset.

In [1]:
```
%pylab inline
# Fetch the MNIST handwritten digit dataset
from sklearn.datasets import fetch_mldata
mnist = fetch_mldata('MNIST original', data_home="../data")
```

```
Populating the interactive namespace from numpy and matplotlib
```

Let's display some data :)

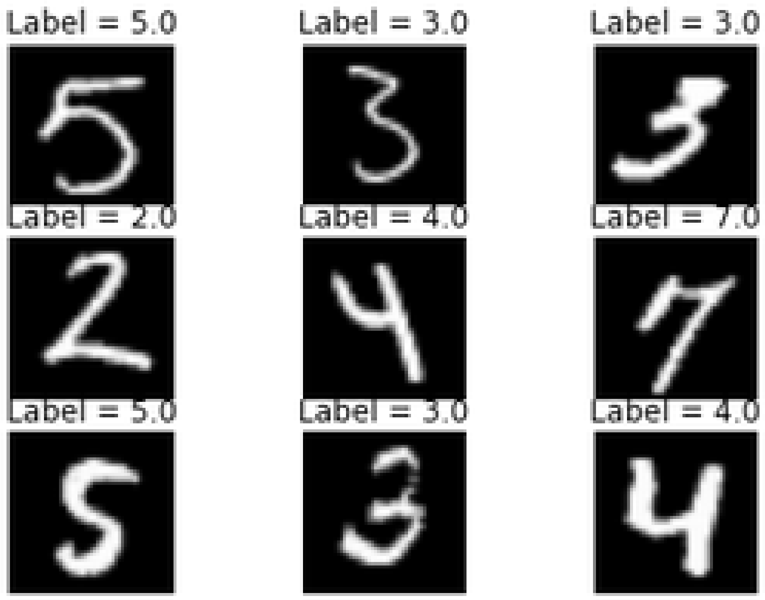
Now let's explore the data.

In [2]:
```
# Display the number of samples
print "(Number of samples, No. of pixels) = ", mnist.data.shape

# Display 9 number randomly selectly
for c in range(1, 10):
    subplot(3, 3,c)
    i = randint(mnist.data.shape[0])
    im = mnist.data[i].reshape((28,28))
    axis("off")
    title("Label = {}".format(mnist.target[i]))
    imshow(im, cmap='gray')
```

```
(Number of samples, No. of pixels) =  (70000, 784)
```

Multinomial Naïve Bayes Application



Split the data into training and testing data

In [3]:
```
# Split the data into training and test data
from sklearn.cross_validation import train_test_split
x_train, x_test, y_train, y_test = train_test_split(mnist.data, mnist.target, test_size=0.05, random_state=42)

# Which is same as
# x_train = mnist.data[:split]
# y_train = mnist.target[:split]
# x_test = mnist.data[split:]
# y_test = mnist.target[split:]
```

In [4]:
```
x_train.shape
```

Out[4]:
```
(66500, 784)
```

Multinomial Naïve Bayes Application

Prepare the classifier

In [6]:

```
# Create the Multinomial Naive Bayes Classifier
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB()
```

Perform the predictions and display the results

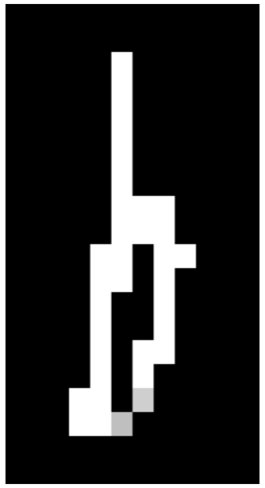
In [7]:

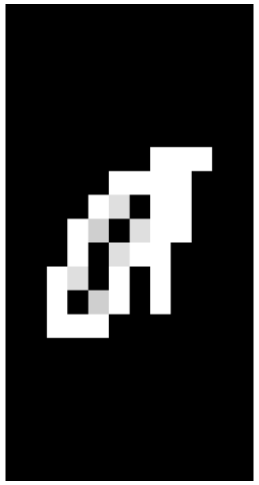
```
# Perform the predictions
clf.fit(x_train,y_train)
# Perform the predictions
y_predicted = clf.predict(x_test)
# Calculate the accuracy of the prediction
from sklearn.metrics import accuracy_score
print "Accuracy = {} %".format(accuracy_score(y_test, y_predicted)*100)
```

```
Accuracy = 81.7142857143 %
```

What if features are continuous?

Character recognition: X_i is the pixel intensity of the i^{th} pixel





Gaussian Naïve Bayes (GNB):

$$P(X_i = x \mid Y = y_k) = \frac{1}{\sigma_{ik}\sqrt{2\pi}} \; e^{\frac{-(x-\mu_{ik})^2}{2\sigma_{ik}^2}}$$

Different mean and variance for each class k and each pixel i.

Sometimes assume variance

- is independent of Y (i.e., σ_i),
- or independent of X_i (i.e., σ_k)
- or both (i.e., σ)

Example: GNB for classifying mental states

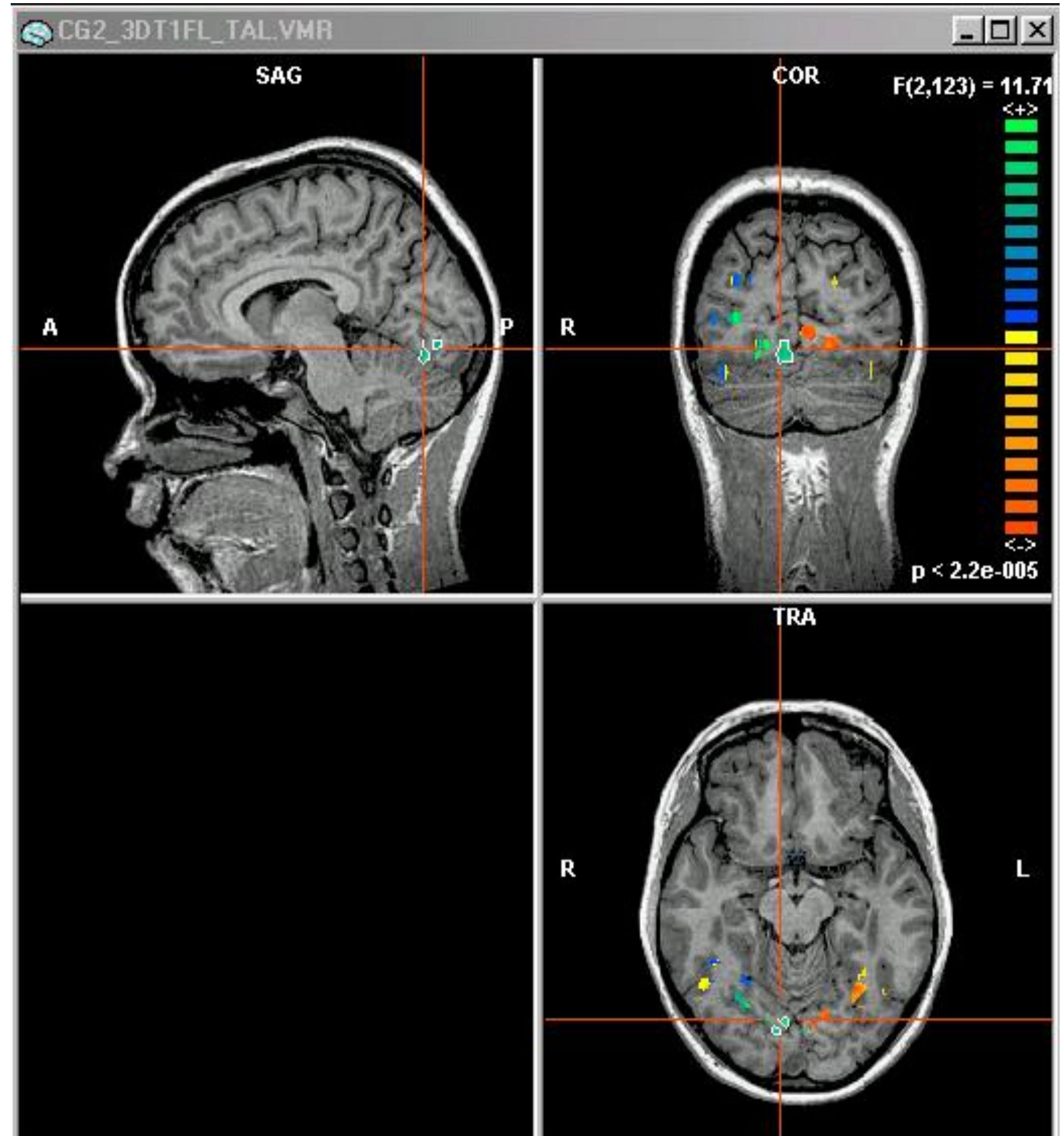
~1 mm resolution

~2 images per sec.

15,000 voxels/image

non-invasive, safe

measures Blood Oxygen Level Dependent (BOLD) response



[Mitchell et al.]

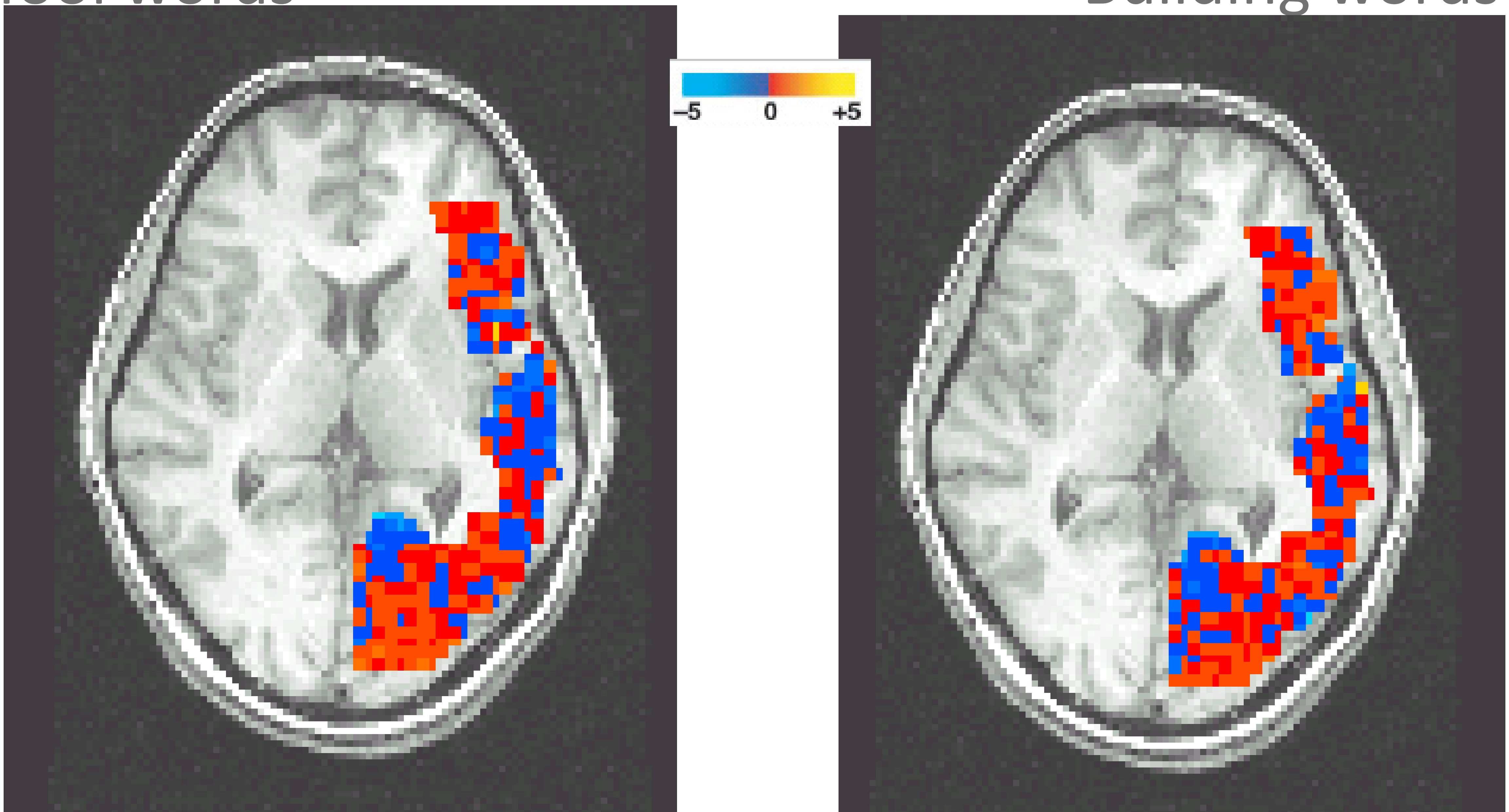
Learned Naïve Bayes Models – Means for P(BrainActivity | WordCategory)

Pairwise classification accuracy: 78-99%, 12 participants

[Mitchell et al.]

Tool words

Building words



What you should know…

Naïve Bayes classifier

- What's the assumption
- Why we use it
- How do we learn it
- Why is Bayesian (MAP) estimation important

Text classification

- Bag of words model

Gaussian NB

- Features are still conditionally independent
- Each feature has a Gaussian distribution given class

Thanks for your attention

☺

References

Many slides are taken from

- Tom Mitchel
  http://www.cs.cmu.edu/~tom/10701_sp11/slides
- Alex Smola
- Aarti Singh
- Eric Xing
- Xi Chen
- http://www.math.ntu.edu.tw/~hchen/teaching/StatInference/notes/lecture2.pdf
- Wikipedia