Statistical NLP
Fall 2017

Lecture 2:
Text Classification
Naive Bayes, Logistic Regression, Neural Networks

Ankur Parikh – Google
Announcements

- Homework 1 deadline was ambiguous on website. You may submit until 11:59 pm tomorrow and not use late days.

- Homework 2 (your first real homework) will be released today or tomorrow.

- My office hours: After class (7-8pm), I have reserved WW 328.
So far: language models give \( P(s) \)
- Help model fluency for various noisy-channel processes (MT, ASR, etc.)
- N-gram models don’t represent any deep variables involved in language structure or meaning
- Usually we want to know something about the input other than how likely it is (syntax, semantics, topic, etc)

Today: Text Classification
- We introduce a single new global variable
- Still a very simplistic model family
- Lets us model hidden properties of text, but only very non-local ones…
- In particular, we can only model properties which are largely invariant to word order (like topic)
Text Classification

- Automatically make a decision about inputs
  - Example: document $\rightarrow$ category
  - Example: image of digit $\rightarrow$ digit
  - Example: image of object $\rightarrow$ object type
  - Example: query + webpages $\rightarrow$ best match
  - Example: symptoms $\rightarrow$ diagnosis
  - ...

- Three main ideas
  - Representation as feature vectors / kernel functions
  - Scoring by linear functions
Text Categorization

- Want to classify documents into broad semantic topics

Obama is hoping to rally support for his $825 billion stimulus package on the eve of a crucial House vote. Republicans have expressed reservations about the proposal, calling for more tax cuts and less spending. GOP representatives seemed doubtful that any deals would be made.

California will open the 2009 season at home against Maryland Sept. 5 and will play a total of six games in Memorial Stadium in the final football schedule announced by the Pacific-10 Conference Friday. The original schedule called for 12 games over 12 weekends.

- Which one is the politics document? (And how much deep processing did that decision take?)
- One approach: bag-of-words and Naïve-Bayes models
- Begin with a labeled corpus containing examples of each class…
Secretary of State Rex Tillerson and Russian Foreign Minister Sergey Lavrov, who are expected to meet this month in New York on the sidelines of the U.N. General Assembly, charged Shannon and Ryabkov earlier this year with exploring ways to resolve bilateral disputes that are hindering broader cooperation on strategic and security issues, such as the war in Syria and the conflict in Ukraine.

It’s not often two teams that are elite at competing aspects of the game face each other. Patriots-Falcons was a great matchup to end last season, but it was a war of offense vs. offense. This won’t be like that. This will be a matchup of arguably the best quarterback in the game vs. arguably the best defense in the game.

... 

The House approved the legislation 316-90, in a vote that authorized $15.3 billion in aid for those affected by Harvey, raised the debt ceiling, and extended government funding for three months into December.
McCain called for the return of regular order in his July floor speech during debate over a bill would have repealed much of Obamacare. “The Congress must now return to regular order, hold hearings, receive input from members of both parties, and heed the recommendations of our nation’s governors so that we can produce a bill that finally provides Americans with access to quality and affordable health care,” McCain said at the time.

Beyond the winning streak, the Indians are now only one game behind the Astros for the best record in the AL, pending the outcome of Houston’s doubleheader with the A’s on Saturday. The best record in the league would secure home-field advantage through at least the ALCS. Then again, maybe home-field advantage doesn’t matter much. The Indians have actually played better on the road this year:
Generative vs. Discriminative Classifiers

- Want to learn to predict label $Y$, given input $X$

- **Generative Classifier (e.g. Naive Bayes)**
  - Assume functional form for $P(X|Y)$, $P(Y)$
  - Estimate probabilities from data (don’t forget to smooth)
  - Use Bayes rule to calculate $P(Y|X)$

- **Discriminative Classifier (e.g. Logistic Regression)**
  - Assume functional form for $P(Y|X)$
  - Estimate parameters from data (don’t forget to regularize)
  - Directly predict label by computing $P(Y|X)$
Idea: pick a topic, then generate a document using a language model for that topic.

Naïve-Bayes assumption: all words are independent given the topic.

\[ P(c, w_1, w_2, \ldots w_n) = P(c) \prod_{i} P(w_i | c) \]

We have to smooth these!

Compare to a unigram language model:

\[ P(w_1, w_2, \ldots w_n) = \prod_{i} P(w_i) \]
Using NB for Classification

- We have a joint model of topics and documents

\[ P(c, w_1, w_2, \ldots w_n) = P(c) \prod_i P(w_i | c) \]

- Gives posterior likelihood of topic given a document

\[ P(c | w_1, w_2, \ldots w_n) = \frac{P(c) \prod_i P(w_i | c)}{\sum_{c'} \left[ P(c') \prod_i P(w_i | c') \right]} \]

- What about totally unknown words? How can unigram models be so terrible for language modeling, but work well for text categorization?
Model Parameters

$P(c)$

$P(w_i|c)$

$\theta_k$

$\theta_{j,k}$
Learning

• In training time, we need to learn the parameters.

\[ \theta := \{ \theta_k \}_{\forall k} \cup \{ \theta_{j,k} \}_{\forall (j,k)} \]

• Maximum likelihood:

\[ \theta := \arg\max_\theta \log \prod_{n=1}^{N} P(w^n, c|\theta) \]

• Maximum likelihood solution is the intuitive one (and can be trivially obtained by counting):

\[ \theta_k = \hat{P}(c = k) \propto \#[c = k] \]

\[ \theta_{j,k} = \hat{P}(w_i = j|c = k) \propto \#[w_i = j, c = k] \]
Learning

- Derivation for simpler case of only class probability

\[ \theta_k := \arg\max_{\theta_k} \log \prod_{n=1}^N P(c^n | \theta_k) \]

\[ := \arg\max_{\theta_k} \sum_{n=1}^N \log P(c^n | \theta_k) \]

\[ := \arg\max_{\theta_k} \sum_{n=1}^N \log \prod_{k} \theta_k^{[c^n = k]} \]

\[ := \arg\max_{\theta_k} \sum_{n=1}^N \sum_{k=1}^K \mathbb{I}[c^n = k] \log \theta_k \]

\[ := \arg\max_{\theta_k} \sum_{n=1}^N \sum_{k=1}^{K-1} \mathbb{I}[c^n = k] \log \theta_k + \mathbb{I}[c^n = K] \log(1 - \sum_{k=1}^{K-1} \theta_k) \]

Parameters are required to sum to one because they are probabilities.
Learning (cont.)

- Even simpler case where $K=2$.

\[
L(\theta) := \sum_{n=1}^{N} \mathbb{I}[c^n = 1] \log \theta_1 + \mathbb{I}[c^n = 2] \log(1 - \theta_1)
\]

- Take derivative.

\[
\frac{\partial L}{\partial \theta_1} = \sum_{n=1}^{N} \mathbb{I}[c^n = 1] \frac{1}{\theta_1} - \mathbb{I}[c^n = 2] \frac{1}{1 - \theta_1}
\]

- Setting to zero and solving gives desired result. For $K > 2$, more algebra (i.e. Lagrange Multipliers) but result is analogous.
Example: Stoplights

P(g,r,w) = 3/7  P(r,g,w) = 3/7  P(r,r,b) = 1/7

NB FACTORS:
- P(w) = 6/7
- P(r|w) = 1/2
- P(g|w) = 1/2
- P(b) = 1/7
- P(r|b) = 1
- P(g|b) = 0

P(b|r,r) = 4/10 (what happened?)
Example: Stoplights

- What does the model say when both lights are red?
  - $P(b,r,r) = (1/7)(1)(1) = 1/7 = 4/28$
  - $P(w,r,r) = (6/7)(1/2)(1/2) = 6/28 = 6/28$
  - $P(w|r,r) = 6/10!$

- We’ll guess that $(r,r)$ indicates lights are working!

- Imagine if $P(b)$ were boosted higher, to 1/2:
  - $P(b,r,r) = (1/2)(1)(1) = 1/2 = 4/8$
  - $P(w,r,r) = (1/2)(1/2)(1/2) = 1/8 = 1/8$
  - $P(w|r,r) = 1/5!$

- Changing the parameters traded accuracy for data likelihood
Example: Barometers

**Reality**

- **Raining**
  - $P(+,+,r) = 1/8$
  - $P(-,-,r) = 3/8$
- **Sunny**
  - $P(+,+,s) = 3/8$
  - $P(-,-,s) = 1/8$

**NB Model**

- **Raining?**
  - M1
  - M2

**NB FACTORS:**
- $P(s) = 1/2$
- $P(-|s) = 1/4$
- $P(-|r) = 3/4$

**PREDICTIONS:**
- $P(r,-,-) = (1/2)(3/4)(3/4)$
- $P(s,-,-) = (1/2)(1/4)(1/4)$
- $P(r|-,-) = 9/10$
- $P(s|-,-) = 1/10$

*Overconfidence!*
(Non-)Independence Issues

- **Mild Non-Independence**
  - Evidence all points in the right direction
  - Observations just not entirely independent
  - Results: Inflated Confidence, Deflated Priors
  - What to do? Boost priors or attenuate evidence

\[
P(c, w_1, w_2, \ldots, w_n)" = " P(c)^{boost > 1} \prod_i P(w_i \mid c)^{boost < 1}
\]

- **Severe Non-Independence**
  - Words viewed independently are misleading
Overview

- So far: Naïve Bayes models for classification
  - Generative models, estimating $P(X|Y)$ and $P(Y)$
  - Assumption: Features are independent given the label (often violated in practice)
  - Easy to estimate (simple counting)

- Next: Logistic Regression for classification
  - Discriminative models, estimating $P(Y|X)$ directly
  - Very flexible feature handling
  - Require numerical optimization methods
Example: Text Classification

<table>
<thead>
<tr>
<th>DOCUMENT</th>
<th>CATEGORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>... win the election ...</td>
<td>POLITICS</td>
</tr>
<tr>
<td>... win the game ...</td>
<td>SPORTS</td>
</tr>
<tr>
<td>... see a movie ...</td>
<td>OTHER</td>
</tr>
</tbody>
</table>

- Document’s source
- Document layout
Feature Representations

- Features are indicator functions $f_i$ which count the occurrences of certain patterns in the input.

Washington County jail served 11,166 meals last month - a figure that translates to feeding some 120 people three times daily for 31 days.

```
features

context:jail = 1
context:county = 1
context:feeding = 1
context:game = 0
...
local-context:jail = 1
local-context:meals = 1
...
subcat:NP = 1
subcat:PP = 0
...
object-head:meals = 1
object-head:ball = 0
```
Some Definitions

**INPUTS**
\[ x^i \]

… win the election …

**OUTPUT SPACE**
\[ y \]

SPORTS, POLITICS, OTHER

**OUTPUTS**
\[ y \]

SPORTS

**TRUE OUTPUTS**
\[ y^i \]

POLITICS

**FEATURE VECTORS**
\[ f_i(y) \]

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

*Either x is implicit, or y contains x*
One Set of Possible Definitions

- Feature Templates:
  - WordIdentity()
  - Suffix(3), Prefix(3)

- Predicates (‘Features’):
  - `word_win`, `word_election`
  - `suffix3_win`, `suffix3_ion`, …

- Feature Indices/Weights:
  - `word_win x SPORTS`, `word_win x POLITICS`, …
From Templates to Weights

- There is one weight for each predicate and output pair
- People often simply refer to this as the ‘feature vector’

\[
x \quad \ldots \text{win the election} \ldots
\]

\[
\text{“f}_{i}(x)"
\]

\[
\begin{bmatrix}
1 & 0 & 1 & 0
\end{bmatrix}
\]

\[
\text{“win”} \quad \rightarrow \quad \begin{bmatrix}
1 & 0 & 1 & 0
\end{bmatrix}
\]

\[
\text{“election”}
\]

\[
f_i(SPORTS) = [1\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]
\]

\[
f_i(POLITICS) = [0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]
\]

\[
f_i(OTHER) = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1]
\]
Non-Block Feature Vectors

- Sometimes the features of candidates cannot be decomposed
- Example: a parse tree’s features may be the productions present in the tree

\[
f_i(\text{NP} \quad \text{VP}) = [1 \quad 0 \quad 1 \quad 0 \quad 1]
\]

\[
f_i(\text{NP} \quad \text{VP}) = [1 \quad 1 \quad 0 \quad 1 \quad 0]
\]

- Different candidates will thus often share features
- We’ll return to the non-block case later
Linear Models: Scoring

In a linear model, each feature index is associated with a weight

\[
\mathbf{f}_i(\text{POLITICS}) = [\ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0] \\
\mathbf{f}_i(\text{SPORTS}) = [\ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0] \\
\mathbf{w} = [\ 1 \ 1 \ -1 \ -2 \ 1 \ -1 \ 1 \ -2 \ -2 \ -1 \ -1 \ 1]
\]

We compare hypotheses on the basis of their linear scores:

\[
\text{score}(\mathbf{x}^i, \mathbf{y}, \mathbf{w}) = \mathbf{w}^\top \mathbf{f}_i(\mathbf{y})
\]

\[
\mathbf{f}_i(\text{POLITICS}) = [\ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0] \\
\mathbf{w} = [\ 1 \ 1 \ -1 \ -2 \ 1 \ -1 \ 1 \ -2 \ -2 \ -1 \ -1 \ 1]
\]

\[
\text{score}(\mathbf{x}^i, \text{POLITICS}, \mathbf{w}) = 1 \times 1 + 1 \times 1 = 2
\]
The linear prediction rule:

$$\text{prediction}(\mathbf{x}^i, \mathbf{w}) = \arg\max_{y \in \mathcal{Y}} \mathbf{w}^\top \mathbf{f}_i(y)$$

$$\text{score}(\mathbf{x}^i, \text{SPORTS}, \mathbf{w}) = 1 \times 1 + (-1) \times 1 = 0$$
$$\text{score}(\mathbf{x}^i, \text{POLITICS}, \mathbf{w}) = 1 \times 1 + 1 \times 1 = 2$$
$$\text{score}(\mathbf{x}^i, \text{OTHER}, \mathbf{w}) = (-2) \times 1 + (-1) \times 1 = -3$$

$$\text{prediction}(\mathbf{x}^i, \mathbf{w}) = \text{POLITICS}$$

We’ve said nothing about where weights come from!
If more than two classes:
- Highest score wins
- Boundaries are more complex
- Harder to visualize

There are other ways: e.g. reconcile pairwise decisions, error
The challenge in feature engineering

- How expressive are linear models? Is it possible to design a linear model that will always get zero training error?

- Yes! Because I can define arbitrarily expressive features.

\[
prediction(x^i, w) = \arg \max_{y \in \mathcal{Y}} w^\top f_i(y)
\]

Can be arbitrarily expressive in theory.
The challenge in feature engineering

- However, if features are too expressive, will not generalize well to dev/test set.

- **Challenge**: Design features that can generalize well.
What if I try to pick the features (such that after learning weights on the training set), it minimizes the test error?

This is also bad. My “design/development” choices are causing overfitting.
Better Experimental Setup

Hold Out Set:

Cross Validation:
Learning Classifier Weights

- Two broad approaches to learning weights

- Generative: work with a probabilistic model of the data, weights are (log) local conditional probabilities
  - Advantages: learning weights is easy, smoothing is well-understood, backed by understanding of modeling

- Discriminative: set weights based on some error-related criterion
  - Advantages: error-driven, often weights which are good for classification aren’t the ones which best describe the data
How to pick weights?

- **Goal:** choose “best” vector \( w \) given training data
  - For now, we mean “best for classification”

- **The ideal:** the weights which have greatest test set accuracy / F1 / whatever
  - But, don’t have the test set
  - Must compute weights from training set

- **Maybe we want weights which give best training set accuracy?**
  - Hard discontinuous optimization problem
  - May not (does not) generalize to test set
Linear Models: Maximum Entropy

- Maximum entropy (logistic regression)
  - Use the scores as probabilities:
    \[
    P(y|x, w) = \frac{\exp(w^T f(y))}{\sum_{y'} \exp(w^T f(y'))} \quad \text{Make positive Normalize}
    \]

- Maximize the (log) conditional likelihood of training data
  \[
  L(w) = \log \prod_i P(y_i|x_i, w) = \sum_i \log \left( \frac{\exp(w^T f_i(y_i))}{\sum_y \exp(w^T f_i(y))} \right)
  = \sum_i \left( w^T f_i(y_i) - \log \sum_y \exp(w^T f_i(y)) \right)
  \]
Derivative for Maximum Entropy

\[ L(w) = \sum_i \left( w^\top f_i(y^i) - \log \sum_y \exp(w^\top f_i(y)) \right) \]

\[ \frac{\partial L(w)}{\partial w_n} = \sum_i \left( f_i(y^i)_n - \sum_y P(y|x_i)f_i(y)_n \right) \]

Total count of feature n in correct candidates

Expected count of feature n in predicted candidates
The optimum parameters are the ones for which each feature’s predicted expectation equals its empirical expectation. The optimum distribution is:

- Always unique (but parameters may not be unique)
- Always exists (if features counts are from actual data).

\[
\frac{\partial L(w)}{\partial w_n} = \sum_i \left( f_i(y^i)_n - \sum_y P(y|x_i)f_i(y)_n \right)
\]

The weight for the “context-word:jail and cat:prison” feature:

- Actual = 1
- Empirical = 1.2
Motivation for maximum entropy:
- Connection to maximum entropy principle (sort of)
- Might want to do a good job of being uncertain on noisy cases...
- ... in practice, though, posteriors are pretty peaked

Regularization (smoothing)

\[
\max_w \sum_i \left( w^T f_i(y^i) - \log \sum_y \exp(w^T f_i(y)) \right) - k \|w\|^2
\]
Derivative for Maximum Entropy

\[ L(w) = -k||w||^2 + \sum_i \left( w^T f_i(y^i) - \log \sum_y \exp(w^T f_i(y)) \right) \]

\[ \frac{\partial L(w)}{\partial w_n} = -2kw_n + \sum_i \left( f_i(y^i)_n - \sum_y P(y|x_i)f_i(y)_n \right) \]

Big weights are bad

Total count of feature \( n \) in correct candidates

Expected count of feature \( n \) in predicted candidates
Unconstrained Optimization

- The maxent objective is an unconstrained optimization problem

\[ L(w) \]

- Basic idea: move uphill from current guess
- Gradient ascent / descent follows the gradient incrementally
- At local optimum, derivative vector is zero
- Will converge if step sizes are small enough, but not efficient
Once we have a function $f$, we can find a local optimum by iteratively following the gradient.

For convex functions, a local optimum will be global.

Basic gradient ascent isn’t very efficient, but there are simple enhancements which take into account previous gradients: conjugate gradient, L-BFGs.

There are special-purpose optimization techniques for maxent, like iterative scaling, but they aren’t better.
The MaxEnt objective is nicely behaved:
- Differentiable (so many ways to optimize)
- Concave (so no local optima)

\[ f(\lambda a + (1 - \lambda)b) \geq \lambda f(a) + (1 - \lambda)f(b) \]

Concavity guarantees a single, global maximum value because any higher points are greedily reachable
Linear Models: Naïve-Bayes

- Naïve-Bayes is a linear model, where:

\[ x^i = d_1, d_2, \cdots d_n \]

\[ \{ v_j \}_{j=1}^{\mid V \mid} \]

words in vocabulary

\[ \{ \#v_j \}_{j=1}^{\mid V \mid} \]

counts of each word

\[
\begin{align*}
    f_i(y) &= [1, \#v_1, \#v_2, \ldots \log \#v_{\mid V \mid}] \\
    w &= [\log P(y), \log P(v_1|y), \log P(v_2|y), \ldots \log P(v_{\mid V \mid}|y)]
\end{align*}
\]
Linear Models: Naïve-Bayes

- Derivation:

\[
score(x_i, y, w) = w^\top f_i(y) = \log P(y) + \sum_k \#v_k \log P(v_k | y) = \log \left( P(y) \prod_k P(v_k | y)^{\#v_k} \right) = \log \left( P(y) \prod_{d \in x^i} P(d | y) \right) = \log P(x^i, y)\]
Because of regularization, the more common prefixes have larger weights even though entire-word features are more specific.

**Local Context**

<table>
<thead>
<tr>
<th>State</th>
<th>Prev</th>
<th>Cur</th>
<th>Next</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word</td>
<td>at</td>
<td>Grace</td>
<td>Road</td>
</tr>
<tr>
<td>Tag</td>
<td>IN</td>
<td>NNP</td>
<td>NNP</td>
</tr>
<tr>
<td>Sig</td>
<td>x</td>
<td>Xx</td>
<td>Xx</td>
</tr>
</tbody>
</table>

**Feature Weights**

<table>
<thead>
<tr>
<th>Feature Type</th>
<th>Feature</th>
<th>PERS</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous word</td>
<td>at</td>
<td>-0.73</td>
<td>0.94</td>
</tr>
<tr>
<td>Current word</td>
<td>Grace</td>
<td>0.03</td>
<td>0.00</td>
</tr>
<tr>
<td>Beginning bigram</td>
<td>&lt;G</td>
<td>0.45</td>
<td>-0.04</td>
</tr>
<tr>
<td>Current POS tag</td>
<td>NNP</td>
<td>0.47</td>
<td>0.45</td>
</tr>
<tr>
<td>Prev and cur tags</td>
<td>IN NNP</td>
<td>-0.10</td>
<td>0.14</td>
</tr>
<tr>
<td>Previous state</td>
<td>Other</td>
<td>-0.70</td>
<td>-0.92</td>
</tr>
<tr>
<td>Current signature</td>
<td>Xx</td>
<td>0.80</td>
<td>0.46</td>
</tr>
<tr>
<td>Prev state, cur sig</td>
<td>O-Xx</td>
<td>0.68</td>
<td>0.37</td>
</tr>
<tr>
<td>Prev-cur-next sig</td>
<td>x-Xx-Xx</td>
<td>-0.69</td>
<td>0.37</td>
</tr>
<tr>
<td>P. state - p-cur sig</td>
<td>O-x-Xx</td>
<td>-0.20</td>
<td>0.82</td>
</tr>
<tr>
<td>Total:</td>
<td></td>
<td>-0.58</td>
<td>2.68</td>
</tr>
</tbody>
</table>
## Some Empirical Results

From Zhang & Oles 2001

<table>
<thead>
<tr>
<th></th>
<th>Naive Bayes</th>
<th>Logistic Reg</th>
<th>SVM</th>
<th>Mod SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>earn</td>
<td>96.6</td>
<td>98.4</td>
<td>98.1</td>
<td>98.1</td>
</tr>
<tr>
<td>acq</td>
<td>91.7</td>
<td>95.2</td>
<td>95.3</td>
<td>94.5</td>
</tr>
<tr>
<td>money-fx</td>
<td>70.0</td>
<td>75.2</td>
<td>74.4</td>
<td>74.5</td>
</tr>
<tr>
<td>grain</td>
<td>76.6</td>
<td>88.4</td>
<td>89.6</td>
<td>90.6</td>
</tr>
<tr>
<td>crude</td>
<td>84.1</td>
<td>85.9</td>
<td>84.8</td>
<td>84.0</td>
</tr>
<tr>
<td>trade</td>
<td>52.3</td>
<td>72.9</td>
<td>73.4</td>
<td>74.8</td>
</tr>
<tr>
<td>interest</td>
<td>68.2</td>
<td>78.1</td>
<td>75.9</td>
<td>74.7</td>
</tr>
<tr>
<td>ship</td>
<td>76.4</td>
<td>81.9</td>
<td>82.4</td>
<td>83.8</td>
</tr>
<tr>
<td>wheat</td>
<td>58.1</td>
<td>88.2</td>
<td>88.9</td>
<td>89.6</td>
</tr>
<tr>
<td>corn</td>
<td>52.4</td>
<td>88.7</td>
<td>86.2</td>
<td>86.7</td>
</tr>
</tbody>
</table>
A Note on Features: TF/IDF

- More frequent terms in a document are more important:
  \[ f_{ij} = \text{frequency of term } i \text{ in document } j \]

- May want to normalize term frequency (tf) by dividing by the frequency of the most common term in the document:
  \[ tf_{ij} = \frac{f_{ij}}{\max_i \{f_{ij}\}} \]

- Terms that appear in many different documents are less indicative:
  \[ df_i = \text{document frequency of term } i = \text{number of documents containing term } i \]
  \[ idf_i = \text{inverse document frequency of term } i = \log_2 (N/ df_i) \]
  \[ N= \text{total number of documents} \]

- An indication of a term’s discrimination power.
- Log used to dampen the effect relative to tf.
- A typical combined term importance indicator is tf-idf weighting:
The perceptron algorithm
- Different learning algorithm as opposed to gradient descent.
- Iteratively processes the training set, reacting to training errors
- Can be thought of as trying to drive down training error

The (online) perceptron algorithm:
- Start with zero weights
- Visit training instances one by one
  - Try to classify
  - If correct, no change!
  - If wrong: adjust weights
The (online) perceptron algorithm:

For each example:

\[ \hat{y}_i = \arg \max_{y \in \mathcal{Y}} w^T f_i(y) \]

If incorrect, update weights as follows:

\[ w \leftarrow w - f_i(\hat{y}_i) \]
\[ w \leftarrow w + f_i(y_i) \]
Based on two ideas:

- Represent words with continuous vectors as opposed to discrete features
- Use highly nonlinear, non-convex functions to build expressive models on top of these input vectors.
Natural Language Inference

- Let’s take a different problem as a running example.
- Natural language inference: Determine entailment/contradiction relationships between a premise and a hypothesis.

<table>
<thead>
<tr>
<th>Premise</th>
<th>Bob is in his room, but because of the thunder and lightning outside, he cannot sleep.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hypothesis 1</td>
<td>Bob is awake.</td>
</tr>
<tr>
<td>Hypothesis 2</td>
<td>It is sunny outside.</td>
</tr>
<tr>
<td>Hypothesis 3</td>
<td>Bob has a big house.</td>
</tr>
</tbody>
</table>
Notation

\[ a = (a_1, \ldots a_n) \] premise

\[ b = (b_1, \ldots b_n) \] hypothesis

\[ y \in \{n, c, e\} \] labels

Goal is to learn a function \( f : (a, b) \rightarrow \hat{y} \in \{n, c, e\} \)

estimated label, that we want to be close to \( y \)
Start with words!

But each word is represented by a dense vector, called a **word embedding**. For this talk, let us pretend these embeddings are given to us a priori.

\[
\text{thunder} = (0.5, 0.75, -0.6, -0.1, 0.2, \ldots) \\
\text{sleep} = (0.6, -0.5, -0.3, 0.9, 0.4, \ldots)
\]
Word Embeddings

- Similar words (ideally) have similar vectors

https://www.tensorflow.org/
A Simple Neural Net

- A next step can be to take combine the word embeddings for each word to construct an embedding for a given sentence.
- For now, let us just average the embeddings.

\[
\bar{a} = \frac{a_1 + \ldots + a_n}{n}
\]

\[
\bar{b} = \frac{b_1 + \ldots + b_n}{n}
\]
A Simple Neural Net

- We can now take our mean embeddings for each sentence and pass them through some complicated non-linear function. It can look something like this:

$$z_c := \tanh(W_2^c \times \tanh(W_1^c \times [\bar{a}; \bar{b}] + b_1^c) + b_2^c)$$

This is usually called a feed forward neural network.
A Simple Neural Net

- We can now take our mean embeddings for each sentence and pass them through some complicated non-linear function. It can look something like this:

\[
z_c := \tanh(W_2^c \times \tanh(W_1^c \times [\bar{a}; \bar{b}] + b_1^c) + b_2^c)
\]

This is usually called a feed forward neural network.
A Simple Neural Net

- Now get scores for all the classes, and normalize to get the probability of each class

\[ z_c := \text{tanh}(W_c^2 \times \text{tanh}(W_c^1 \times [\bar{a}; \bar{b}] + b_1^c) + b_2^c) \]

\[ z_e := \text{tanh}(W_e^2 \times \text{tanh}(W_e^1 \times [\bar{a}; \bar{b}] + b_1^e) + b_2^e) \]

\[ z_n := \text{tanh}(W_n^2 \times \text{tanh}(W_n^1 \times [\bar{a}; \bar{b}] + b_1^n) + b_2^n) \]

\[ P(z_c|a, b) = \frac{\exp(z_c)}{\exp(z_c) + \exp(z_e) + \exp(z_n)} \]
A Simple Neural Net

- Lots of complicated stuff happened, but it's all differentiable! Yay!!

\[
\bar{a} = \frac{a_1 + \ldots + a_n}{n} \quad \bar{b} = \frac{b_1 + \ldots + b_n}{n}
\]

\[
z_c := \tanh(W^c_2 \times \tanh(W^c_1 \times [\bar{a}; \bar{b}] + b^c_1) + b^c_2)
\]

\[
z_e := \tanh(W^e_2 \times \tanh(W^e_1 \times [\bar{a}; \bar{b}] + b^e_1) + b^e_2)
\]

\[
z_n := \tanh(W^n_2 \times \tanh(W^n_1 \times [\bar{a}; \bar{b}] + b^n_1) + b^n_2)
\]

\[
P(z_c|a, b) = \frac{\exp(z_c)}{\exp(z_c) + \exp(z_e) + \exp(z_n)}
\]

Can be thought of as one big function f
More generally,

- A feed forward network can be thought of successively applying linear transformations and non-linear activation functions.

\[ z = F_1(W_1(F_2(\ldots F_n(W_n[a; b] + b_n)\ldots + b_2) + b_1) \]

- Common choices of activation:
  - sigmoid
  - tanh
  - ReLU = \( \max(x, 0) \)
Learning in Neural Networks

- As long as the activation functions are differentiable, it is possible to compute gradient via chain rule (Backpropagation).

\[ z = F_1(W_1(F_2(.....F_n(W_n[a:b] + b_n).... + b_2) + b_1) \]

- Can then use gradient descent to optimize the objective.
- Existing computing libraries like Tensorflow / Torch automatically compute the gradient for you.
- Learning problem is highly non-convex, it is not guaranteed you will get to the optimal solution.
From MaxEnt to Neural Nets

In NLP, a maxent classifier is normally written as:

\[
P(c \mid d, \lambda) = \frac{\exp \sum_i \lambda_i f_i(c, d)}{\sum_{c' \in C} \exp \sum_i \lambda_i f_i(c', d)}
\]

Supervised learning gives us a distribution for datum \(d\) over classes in \(C\)

Vector form:

\[
P(c \mid d, \lambda) = \frac{e^{\lambda^T f(c, d)}}{\sum_{c'} e^{\lambda^T f(c', d)}}
\]

Such a classifier is used as-is in a neural network ("a softmax layer")

- Often as the top layer: \(J = \text{softmax}(\lambda \cdot x)\)

But for now we’ll derive a two-class logistic model for one neuron
From MaxEnt to Neural Nets

Vector form: \[ P(c \mid d, \lambda) = \frac{e^{\lambda^T f(c, d)}}{\sum_{c'} e^{\lambda^T f(c', d)}} \]

Make two class:

\[ P(c_1 \mid d, \lambda) = \frac{e^{\lambda^T f(c_1, d)}}{e^{\lambda^T f(c_1, d)} + e^{\lambda^T f(c_2, d)}} = \frac{e^{\lambda^T f(c_1, d)}}{e^{\lambda^T f(c_1, d)} + e^{\lambda^T f(c_2, d)}} \cdot \frac{e^{-\lambda^T f(c_1, d)}}{e^{-\lambda^T f(c_1, d)}} \]

\[ = \frac{1}{1 + e^{\lambda^T[f(c_2, d) - f(c_1, d)]}} = \frac{1}{1 + e^{-\lambda^T x}} \quad \text{for } x = f(c_1, d) - f(c_2, d) \]

\[ = f(\lambda^T x) \]

for \( f(z) = 1/(1 + \exp(-z)) \), the logistic function – a sigmoid non-linearity.
But that’s a neuron!

\[ h_{w,b}(x) = f(w^T x + b) \]

\[ f(z) = \frac{1}{1 + e^{-z}} \]

**b**: We can have an “always on” feature, which gives a class prior, or separate it out, as a bias term.

*w, b* are the parameters of this neuron i.e., this logistic regression model.
Neural Net = Several MaxEnt Models

If we feed a vector of inputs through a bunch of logistic regression functions, then we get a vector of outputs ...

But we don’t have to decide ahead of time what variables these logistic regressions are trying to predict!
Neural Net = Several MaxEnt Models

... which we can feed into another logistic regression function

It is the training criterion that will direct what the intermediate hidden variables should be, so as to do a good job at predicting the targets for the next layer, etc.
Classification Conclusions

- Naïve Bayes:
  - Generative models, estimating $P(X|Y)$ and $P(Y)$
  - Assumption: Features are independent given the label (often violated in practice)
  - Easy to estimate (simple counting)

- Logistic Regression / Maximum Entropy:
  - Discriminative models, estimating $P(Y|X)$ directly
  - Very flexible feature handling
  - Require numerical optimization methods
  - Converges slower, but to better point than Naïve Bayes

- Neural Networks:
  - A collection of neurons (MaxEnt models) arranged hierarchically (more later)