CPSC 340: Machine Learning and Data Mining

Non-Parametric Models
Fall 2018
Admin

• Course webpage: www.ugrad.cs.ubc.ca/~cs340

• Assignment 2 is out.
  – Due Friday of next week. It’s long so start early.

• Add/drop deadline is tomorrow.

• Auditing: message me on Piazza if you want to audit.
  – Bring your form to me after class.
Last Time: E-mail Spam Filtering

• Want a build a system that filters spam e-mails:

• We formulated as **supervised learning**:
  – \((y_i = 1)\) if e-mail ‘i’ is spam, \((y_i = 0)\) if e-mail is not spam.
  – \((x_{ij} = 1)\) if word/phrase ‘j’ is in e-mail ‘i’, \((x_{ij} = 0)\) if it is not.

<table>
<thead>
<tr>
<th></th>
<th>$</th>
<th>Hi</th>
<th>CPSC</th>
<th>340</th>
<th>Vicodin</th>
<th>Offer</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Spam? 1 1 0 ...
Last Time: Naïve Bayes

• We considered spam filtering methods based on naïve Bayes:

\[ p(y_i = "spam" \mid x_i) = \frac{p(x_i \mid y_i = "spam") p(y_i = "spam")}{p(x_i)} \]

• Makes conditional independence assumption to make learning practical:

\[ p(\text{hello} = 1, \text{vicodin} = 0, 340 = 1 \mid \text{spam}) \approx p(\text{hello} = 1 \mid \text{spam}) p(\text{vicodin} = 0 \mid \text{spam}) p(340 = 1 \mid \text{spam}) \]

• Predict “spam” if \( p(y_i = "spam" \mid x_i) > p(y_i = "not spam" \mid x_i) \).
  – We don’t need \( p(x_i) \) to test this.
Laplace Smoothing

• Our estimate of $p(\text{‘lactase’} = 1 \mid \text{‘spam’})$ is:

$$\frac{\# \text{spam messages with lactase}}{\# \text{spam messages}}$$

— But there is a problem if you have no spam messages with lactase:
  • $p(\text{‘lactase’} \mid \text{‘spam’}) = 0$, so spam messages with lactase automatically get through.

— Common fix is Laplace smoothing:
  • Add 1 to numerator, and 2 to denominator (binary features).
    — Acts like a “fake” spam example that has lactase, and a “fake” spam example that doesn’t.
Laplace Smoothing

- Laplace smoothing:

\[
\frac{(# \text{spam messages with lactase}) + 1}{(# \text{spam messages}) + 2}
\]

- Typically you do this for all features.
  - Helps against overfitting by biasing towards the uniform distribution.

- A common variation is to use a real number \( \beta \) rather than 1.
  - Add ‘\( \beta k \)’ to denominator if feature has ‘\( k \)’ possible values (so sums to 1).
Decision Theory

• Are we equally concerned about “spam” vs. “not spam”?
• True positives, false positives, false negatives, true negatives:

<table>
<thead>
<tr>
<th>Predict / True</th>
<th>True ‘spam’</th>
<th>True ‘not spam’</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predict ‘spam’</td>
<td>True Positive</td>
<td>False Positive</td>
</tr>
<tr>
<td>Predict ‘not spam’</td>
<td>False Negative</td>
<td>True Negative</td>
</tr>
</tbody>
</table>

• The costs mistakes might be different:
  – Letting a spam message through (false negative) is not a big deal.
  – Filtering a not spam (false positive) message will make users mad.
Decision Theory

• We can give a cost to each scenario, such as:

<table>
<thead>
<tr>
<th>Predict / True</th>
<th>True ‘spam’</th>
<th>True ‘not spam’</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predict ‘spam’</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Predict ‘not spam’</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>

• Instead of most probable label, take $y_{\text{hat}}$ minimizing expected cost:

\[
E \left[ \text{cost}(\hat{y}_i, \tilde{y}_i) \right]
\]

• Even if “spam” has a higher probability, predicting “spam” might have a higher cost.
Decision Theory Example

<table>
<thead>
<tr>
<th>Predict / True</th>
<th>True ‘spam’</th>
<th>True ‘not spam’</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predict ‘spam’</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Predict ‘not spam’</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>

• If for a test example we have \( p(\hat{y}_i = \text{“spam”} \mid \tilde{x}_i) = 0.6 \), then:

\[
\mathbb{E} \left[ \cos^2 (\hat{y}_i = \text{“spam”}, \tilde{y}_i) \right] = p(\hat{y}_i = \text{“spam”} \mid \tilde{x}_i) \cos^2 (\hat{y}_i = \text{“spam”}, \tilde{y}_i = \text{“spam”}) \\
+ p(\hat{y}_i = \text{“not spam”} \mid \tilde{x}_i) \cos^2 (\hat{y}_i = \text{“spam”}, \tilde{y}_i = \text{“not spam”}) \\
= (0.6)(0) + (0.4)(100) = 40
\]

\[
\mathbb{E} \left[ \cos^2 (\hat{y}_i = \text{“not spam”}, \tilde{y}_i) \right] = (0.6)(10) + (0.4)(0) = 6
\]

• Even though “spam” is more likely, we should predict “not spam”.
Decision Theory Discussion

• In other applications, the costs could be different.
  – In cancer screening, maybe false positives are ok, but don’t want to miss false negatives.

• Decision theory and “darts”:

• Decision theory can help with “unbalanced” class labels:
  – If 99% of e-mails are spam, you get 99% accuracy by always predicting “spam”.
  – Decision theory approach avoids this.
  – See also precision/recall curves and ROC curves in the bonus material.
Decision Trees vs. Naïve Bayes

- **Decision trees:**
  1. Sequence of rules based on 1 feature.
  2. Training: 1 pass over data per depth.
  5. New data: might need to change tree.
  6. Accuracy: good if simple rules based on individual features work (“symptoms”).

- **Naïve Bayes:**
  1. Simultaneously combine all features.
  2. Training: 1 pass over data to count.
  5. New data: just update counts.
  6. Accuracy: good if features almost independent given label (text).
Parametric vs. Non-Parametric

- Decision trees and naïve Bayes are often not very accurate.
  - Greedy rules or conditional independence might be bad assumptions.
  - They are also parametric models.
Parametric vs. Non-Parametric

- **Parametric models:**
  - Have a fixed number of parameters: trained “model” size is $O(1)$ in terms ‘$n$’.
    - E.g., fixed-depth decision tree just stores rules.
    - E.g., naïve Bayes just stores counts.
  - You can estimate the fixed parameters more accurately with more data.
  - But eventually more data doesn’t help: model is too simple.

- **Non-parametric models:**
  - Number of parameters grows with ‘$n$’: size of “model” depends on ‘$n$’.
  - Model gets more complicated as you get more data.
  - E.g., decision tree whose depth grows with the number of examples.
K-Nearest Neighbours (KNN)

• Classical non-parametric classifier is \textit{k-nearest neighbours (KNN)}.

• To classify an example $\tilde{x}_i$:
  1. Find the ‘k’ training examples $x_i$ that are “nearest” to $\tilde{x}_i$.
  2. Classify using the \textit{most common label} of “nearest” examples.

<table>
<thead>
<tr>
<th>Egg</th>
<th>Milk</th>
<th>Fish</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.7</td>
<td>0</td>
</tr>
<tr>
<td>0.4</td>
<td>0.6</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.3</td>
<td>0.5</td>
<td>1.2</td>
</tr>
<tr>
<td>0.4</td>
<td>0</td>
<td>1.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sick?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

$\tilde{x}_i$: $\hat{y}_i$:

3 “nearest” neighbours.

Most common label is “sick”.
K-Nearest Neighbours (KNN)

• Classical non-parametric classifier is \( k \)-nearest neighbours (KNN).

• To classify an example \( \tilde{x}_i \):
  1. Find the ‘\( k \)’ training examples \( x_i \) that are “nearest” to \( \tilde{x}_i \).
  2. Classify using the most common label of “nearest” examples.
K-Nearest Neighbours (KNN)

- Classical non-parametric classifier is **k-nearest neighbours (KNN)**.
- To classify an example $\tilde{x}_i$:
  1. Find the ‘$k$’ training examples $x_i$ that are “nearest” to $\tilde{x}_i$.
  2. Classify using the most common label of “nearest” examples.
K-Nearest Neighbours (KNN)

- Classical non-parametric classifier is \textit{k-nearest neighbours (KNN)}.
- To classify an example $\tilde{x}_i$:
  1. Find \textit{the ‘k’} training examples $x_i$ that are “nearest” to $\tilde{x}_i$.
  2. Classify using the \textit{most common label} of “nearest” examples.
K-Nearest Neighbours (KNN)

• Classical non-parametric classifier is \textit{k-nearest neighbours (KNN)}.

• To classify an example $\tilde{x}_i$:
  1. Find the ‘$k$’ training examples $x_i$ that are “nearest” to $\tilde{x}_i$.
  2. Classify using the most common label of “nearest” examples.
K-Nearest Neighbours (KNN)

• Assumption:
  – Examples with similar features are likely to have similar labels.

• Most common distance function is Euclidean distance:
  \[ \| x_i - \tilde{x}_i \| = \sqrt{\sum_{j} (x_{ij} - \tilde{x}_{ij})^2} \]
  – \( x_i \) is features of training example ‘i’, and \( \tilde{x}_i \) is features of test example ‘i’.

• With a small ‘n’, KNN model will be very simple.
• Model gets more complicated as ‘n’ increases.
  – Starts to detect subtle differences between examples.
Consistency of KNN

• KNN has appealing **consistency** properties:
  – As ‘n’ goes to $\infty$, KNN test error is **less than twice best possible error**.
    • For fixed ‘k’ and binary labels (under mild assumptions).

• Stone’s Theorem: KNN is “**universally consistent**”.
  – If k/n goes to zero and ‘k’ goes to $\infty$, **converges to the best possible error**.
    • First algorithm shown to have this property.

• Does Stone’s Theorem violate the no free lunch theorem?
  – No: it requires a continuity assumption on the labels.
  – Consistency says nothing about finite ‘n’ (see "**Dont Trust Asymptotics**").
Parametric vs. Non-Parametric Models

![Diagram showing test error versus number of examples 'n'. The parametric model has a higher error initially but converges faster than the non-parametric model. The gap may never go to zero.](Image)
Parametric vs. Non-Parametric Models

The graph illustrates the relationship between the number of examples and test error for both parametric and non-parametric models. The parametric model (blue line) shows a faster decrease in error as the number of examples increases, while the non-parametric model (green line) has a more gradual decrease. The gap between the two models can go to zero with sufficient data.

The graph highlights the advantage of parametric models in terms of faster convergence and potentially better performance with more data.
Curse of Dimensionality

• “Curse of dimensionality”: problems with high-dimensional spaces.
  – Volume of space grows exponentially with dimension.
    • Circle has area $O(r^2)$, sphere has area $O(r^3)$, 4d hyper-sphere has area $O(r^4)$, ...
  – Need exponentially more points to ‘fill’ a high-dimensional volume.
    • “Nearest” neighbours might be really far even with large ‘n’.

• KNN is also problematic if features have very different scales.

• Nevertheless, KNN is really easy to use and often hard to beat!
KNN Implementation

• There is **no training** phase in KNN ("lazy" learning).
  – You just store the training data.
  – **Non-parametric** because the size of the model is $O(nd)$, the size of ‘X’.

• But **predictions are expensive**: $O(nd)$ to classify 1 test example.
  – Tons of work on reducing this cost (we’ll discuss this later).

• There are also alternatives to Euclidean distance...
Defining “Distance” with “Norms”

• A common way to define the “distance” between examples:
  – Take the “norm” of the difference between feature vectors.
    \[
    \| x_i - \tilde{x}_i \|_2 = \sqrt{\sum_{j=1}^{d} (x_{ij} - \tilde{x}_{ij})^2}
    \]

• Norms are a way to measure the “length” of a vector.
  – The most common norm is the “L2-norm” (or “Euclidean norm”):
    \[
    \| r \|_2 = \sqrt{\sum_{j=1}^{d} r_j^2}
    \]
  – Here, the “norm” of the difference is the standard Euclidean distance.
L2-norm, L1-norm, and L∞-Norms.

• The three most common norms: **L2-norm, L1-norm, and L∞-norm.**
  
  – Definitions of these norms with two-dimensions:

  \[ \| r \|_2 = \sqrt{r_1^2 + r_2^2} \]
  \[ \| r \|_1 = |r_1| + |r_2| \]
  \[ \| r \|_\infty = \max \{|r_1|, |r_2|\} \]

  – Notation: we often leave out the “2” for the L2-norm: \[ \| r \| = \| r \|_2 \]
Norms in d-Dimensions

- We can generalize these common norms to d-dimensional vectors:

  \[ \|r\|_2 = \sqrt{\sum_{j=1}^{d} r_j^2} \quad \|r\|_1 = \sum_{j=1}^{d} |r_j| \quad \|r\|_\infty = \max_j \{|r_j|\} \]

  - Example, in 3-dimensions:
    \[ \|r\|_2 = \sqrt{r_1^2 + r_2^2 + r_3^2} \]
    \[ \|r\|_1 = |r_1| + |r_2| + |r_3| \]
    \[ \|r\|_\infty = \max(|r_1|, |r_2|, |r_3|) \]

  - Notation: \( \|r\|_2^2 = (\|r\|_2)^2 = (\sqrt{\sum_{j=1}^{d} r_j^2})^2 = \sum_{j=1}^{d} r_j^2 = \sum_{j=1}^{d} |r_j|^2 = \sum_{j=1}^{d} r_j^2 \)

- These norms place different “weights” on large values:
  - \( L_1 \): all values are equal.
  - \( L_2 \): bigger values are more important (because of squaring).
  - \( L_\infty \): only biggest value is important.
Norms as Measures of Distance

• By taking norm of difference, we get a “distance” between vectors:

\[ \| r - s \|_2 = \sqrt{(r_1 - s_1)^2 + (r_2 - s_2)^2} \]

\[ = \| r - s \| \text{ "Euclidean distance"} \]

\[ \| r - s \|_1 = |r_1 - s_1| + |r_2 - s_2| \]

"Number of blocks you need to walk to get from \( r \) to \( s \)."

\[ \| r - s \|_\infty = \max \{ |r_1 - s_1|, |r_2 - s_2| \} \]

"Most number of blocks in any direction you would have to walk."
3 Defining Properties of Norms

• A “norm” is any function satisfying the following 3 properties:
  
  1. Only ‘0’ has a ‘length’ of zero.
  
  2. Multiplying ‘r’ by constant ‘α’ multiplies length by |α|
     • “If be will twice as long if you multiply by 2”: ||αr|| = |α| • ||r||.
     • Implication is that norms cannot be negative.
  
  3. Length of ‘r+s’ is not more than length of ‘r’ plus length of ‘s’:
     • “You can’t get there faster by a detour”.
     • “Triangle inequality”: ||r + s|| ≤ ||r|| + ||s||.
KNN Distance Functions

• Most common KNN distance functions: norm($x_i - x_j$).
  – L1-, L2-, and Linf-norm.
  – Weighted norms (if some features are more important): $d \sum_{j=1}^{d} v_j \cdot \lvert x_j \rvert$
  – “Mahalanobis” distance (takes into account correlations).

• But we can consider other distance/similarity functions:
  – Jaccard similarity (if $x_i$ are sets).
  – Edit distance (if $x_i$ are strings).
  – Metric learning (learn the best distance function).
Summary

• Decision theory allows us to consider costs of predictions.
• Non-parametric models grow with number of training examples.
• K-Nearest Neighbours: simple non-parametric classifier.
  • Appealing “consistency” properties.
  • Suffers from high prediction cost and curse of dimensionality.

• Next Time:
  • Fighting the fundamental trade-off and Microsoft Kinect.
Other Performance Measures

• Classification error might be wrong measure:
  – Use weighted classification error if have different costs.
  – Might want to use things like Jaccard measure: \( \frac{TP}{TP + FP + FN} \).

• Often, we report **precision** and **recall** (want both to be high):
  – Precision: “if I classify as spam, what is the probability it actually is spam?”
    • Precision = \( \frac{TP}{TP + FP} \).
    • High precision means the filtered messages are likely to really be spam.
  – Recall: “if a message is spam, what is probability it is classified as spam?”
    • Recall = \( \frac{TP}{TP + FN} \)
    • High recall means that most spam messages are filtered.
Precision-Recall Curve

- Consider the rule \( p(y_i = \text{‘spam’} \mid x_i) > t \), for threshold ‘t’.
- Precision-recall (PR) curve plots precision vs. recall as ‘t’ varies.

http://pages.cs.wisc.edu/~jdavis/davisgoadrichcamera2.pdf
ROC Curve

• Receiver operating characteristic (ROC) curve:
  – Plot true positive rate (recall) vs. false positive rate (FP/FP+TN).
  – Diagonal is random, perfect classifier would be in upper left.
  – Sometimes papers report area under curve (AUC).
    • Reflects performance for different possible thresholds on the probability.

http://pages.cs.wisc.edu/~jdavis/davisgoadrichcamera2.pdf
More on Unbalanced Classes

• With unbalanced classes, there are many alternatives to accuracy as a measure of performance:
  – Two common ones are the Jaccard coefficient and the F-score.

• Some machine learning models don’t work well with unbalanced data. Some common heuristics to improve performance are:
  – Under-sample the majority class (only take 5% of the spam messages).
  – Re-weight the examples in the accuracy measure (multiply training error of getting non-spam messages wrong by 10).
  – Some notes on this issue are here.
More on Weirdness of High Dimensions

• In high dimensions:
  – Distances become less meaningful:
    • All vectors may have similar distances.
  – Emergence of “hubs” (even with random data):
    • Some datapoints are neighbours to many more points than average.

– Visualizing high dimensions and sphere-packing
Vectorized Distance Calculation

- To classify ‘t’ test examples based on KNN, cost is $O(ndt)$.
  - Need to compare ‘n’ training examples to ‘t’ test examples, and computing a distance between two examples costs $O(d)$.

- You can do this slightly faster using fast matrix multiplication:
  - Let $D$ be a matrix such that $D_{ij}$ contains:
    $$
    \|x_i - x_j\|^2 = \|x_i\|^2 - 2x_i^\top x_j + \|x_j\|^2
    $$
    where ‘i’ is a training example and ‘j’ is a test example.
  - We can compute $D$ in Julia using:
    ```
    D = X.^2*ones(d,t) + ones(n,d)*(Xtest').^2 - 2*X*Xtest';
    ```
  - And you get an extra boost because Julia uses multiple cores.
Squared/Euclidean-Norm Notation

We’re using the following conventions:

The subscript after the norm is used to denote the $p$-norm, as in these examples:

\[
\|x\|_2 = \sqrt{\sum_{j=1}^{d} w_j^2}.
\]
\[
\|x\|_1 = \sum_{j=1}^{d} |w_j|.
\]

If the subscript is omitted, we mean the 2-norm:

\[
\|x\| = \|x\|_2.
\]

If we want to talk about the squared value of the norm we use a superscript of "2":

\[
\|x\|_2^2 = \sum_{j=1}^{d} w_j^2.
\]
\[
\|x\|_1^2 = \left(\sum_{j=1}^{d} |w_j|\right)^2.
\]

If we omit the subscript and have a superscript of "2", we’re taking about the squared L2-norm:

\[
\|x\|^2 = \sum_{j=1}^{d} w_j^2.
\]
Lp-norms

• The $L_1$, $L_2$, and $L_\infty$-norms are special cases of Lp-norms:

$$\|x\|_p = \left(\sum_{j=1}^{d} x_j^p\right)^{\frac{1}{p}}$$

• This gives a norm for any (real-valued) $p \geq 1$.
  – The $L_\infty$-norm is limit as ‘$p$’ goes to $\infty$.

• For $p < 1$, not a norm because triangle inequality not satisfied.