DSCI 573: Model Selection and Feature Selection

Feature Engineering
Winter 2017
The Question I Hate the Most...

• How much data do we need?

• A difficult if not impossible question to answer.

• My usual answer: “more is better”.  
  – With the warning: “as long as the quality doesn’t suffer”.

• Another popular answer: “ten times the number of features”.
A Simple Setting: Coupon Collecting

• Assume we have a categorical variable with 50 possible values:
  – {Alabama, Alaska, Arizona, Arkansas,...}.

• Assume each category has probability of 1/50 of being chosen:
  – How many objects do we need to see before we expect to see them all?

• Expected value is ~225.

• Coupon collector problem: O(n log n) in general.
  – Gotta Catch’em all!

• Obvious sanity check, is need more samples than categories:
  – Situation is worse if they don’t have equal probabilities.
  – Typically want to see categories more than once to learn.
Feature Engineering and Feature Selection

• A sad truth about machine learning:
  – The right features usually make a bigger difference than the right model.

• Next time we’ll start talking about feature engineering.
  – Designing features that let you learn with less data.
  – This is usually application-specific, and the guidelines are fuzzy.

• Later we’ll start talking about feature selection.
  – Choosing “relevant” features among possible candidates.
  – This can also let you learn with less data.

• The above points are why deep learning is so hot:
  – If you have lots of data, deep learning tries to learn good features.
Simple Example of Feature Engineering

- Feature aggregation:
  - Combine features to form new features:

<table>
<thead>
<tr>
<th>Van</th>
<th>Bur</th>
<th>Sur</th>
<th>Edm</th>
<th>Cal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BC</th>
<th>AB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Simple Example of Feature Selection

• **Feature Selection:**
  – Remove features that are not relevant to the task.

<table>
<thead>
<tr>
<th>SID:</th>
<th>Age</th>
<th>Job?</th>
<th>City</th>
<th>Rating</th>
<th>Income</th>
</tr>
</thead>
<tbody>
<tr>
<td>3457</td>
<td>23</td>
<td>Yes</td>
<td>Van</td>
<td>A</td>
<td>22,000.00</td>
</tr>
<tr>
<td>1247</td>
<td>23</td>
<td>Yes</td>
<td>Bur</td>
<td>BBB</td>
<td>21,000.00</td>
</tr>
<tr>
<td>6421</td>
<td>22</td>
<td>No</td>
<td>Van</td>
<td>CC</td>
<td>0.00</td>
</tr>
<tr>
<td>1235</td>
<td>25</td>
<td>Yes</td>
<td>Sur</td>
<td>AAA</td>
<td>57,000.00</td>
</tr>
<tr>
<td>8976</td>
<td>19</td>
<td>No</td>
<td>Bur</td>
<td>BB</td>
<td>13,500.00</td>
</tr>
<tr>
<td>2345</td>
<td>22</td>
<td>Yes</td>
<td>Van</td>
<td>A</td>
<td>20,000.00</td>
</tr>
</tbody>
</table>

– Student ID is probably not relevant for most tasks.
A Theoretical Answer to the Data Quantity Question

• Assume have source of IID examples and fixed class of parametric models.
  • Like “all depth-5 decision trees”.
• Under some nasty assumptions, with ‘n’ training examples it holds that:
  \[E[\text{test error of best model on training set}] - (\text{best test error}) = O(1/n).\]

• You rarely know the constant factor, but this gives some guidelines:
  – Adding more data helps more on small datasets than on large datasets.
    • Going from 10 training examples to 20, difference with best possible error gets cut in half.
      – If the best possible error is 15% you might go from 20% to 17.5% (this does not mean 20% to 10%).
    • Going from 110 training examples to 120, error only goes down by ~10%.
    • Going from 1M training examples to 1M+10, you won’t notice a change.
  – Doubling the data size cuts the error in half:
    • Going from 1M training to 2M training examples, error gets cut in half.
    • If you double the data size and your test error doesn’t improve, more data might not help.
(pause)
Feature Engineering

• “...some machine learning projects succeed and some fail. What makes the difference? Easily the most important factor is the features used.”
  – Pedro Domingos

• “Coming up with features is difficult, time-consuming, requires expert knowledge. "Applied machine learning" is basically feature engineering.”
  – Andrew Ng
Feature Engineering

• Better features usually help more than a better model.

• Good features would ideally:
  – Capture most important aspects of problem.
  – Generalize to new scenarios.
  – Allow learning with few examples, be hard to overfit with many examples.

• There is a trade-off between simple and expressive features:
  – With simple features overfitting risk is low, but accuracy might be low.
  – With complicated features accuracy can be high, but so is overfitting risk.
Feature Engineering

• The best features may be **dependent on the model** you use.

• For **counting-based methods** like naïve Bayes and decision trees:
  – Need to address coupon collecting, but separate relevant “groups”.

• For **distance-based methods** like KNN:
  – Want different class labels to be “far”.

• For **regression-based methods** like linear regression:
  – Want labels to have a linear dependency on features.
Discretization for Counting-Based Methods

• For counting-based methods:
  – **Discretization**: turn continuous into discrete.

<table>
<thead>
<tr>
<th>Age</th>
<th>&lt; 20</th>
<th>&gt;= 20, &lt; 25</th>
<th>&gt;= 25</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>25</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

  – Counting age “groups” could let us **learn more quickly** than exact ages.
  • But we wouldn’t do this for a distance-based method.
Standardization for Distance-Based Methods

• Consider features with different scales:

<table>
<thead>
<tr>
<th>Egg (#)</th>
<th>Milk (mL)</th>
<th>Fish (g)</th>
<th>Pasta (cups)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>250</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>250</td>
<td>200</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.5</td>
</tr>
<tr>
<td>2</td>
<td>250</td>
<td>150</td>
<td>0</td>
</tr>
</tbody>
</table>

• Should we convert to some standard ‘unit’?
  – It doesn’t matter for counting-based methods.

• It matters for distance-based methods:
  • KNN will focus on large values more than small values.
  • Often we “standardize” scales of different variables (e.g., convert everything to grams).
Non-Linear Transformations for Regression-Based

- Non-linear feature/label transforms can make things more linear:
  - Polynomial, exponential/logarithm, sines/cosines, RBFs.
Discussion of Feature Engineering

• The best feature transformations are application-dependent.  
  – It’s hard to give general advice.

• My advice: ask the domain experts.  
  – Often have idea of right discretization/standardization/transformation.

• Rest of the lecture focuses on some common situations:  
  – Text data, grouped data, and perception tasks.
Text Example 1: Language Identification

• Consider data that doesn’t look like this:

\[ X = \begin{bmatrix} 0.5377 & 0.3188 & 3.5784 \\ 1.8339 & -1.3077 & 2.7694 \\ -2.2588 & -0.4336 & -1.3499 \\ 0.8622 & 0.3426 & 3.0349 \end{bmatrix}, \quad y = \begin{bmatrix} +1 \\ -1 \\ -1 \\ +1 \end{bmatrix}, \]

• But instead looks like this:

\[ X = \begin{bmatrix} \text{Do you want to go for a drink sometime?} \\ \text{J’achète du pain tous les jours.} \\ \text{Fais ce que tu veux.} \\ \text{There are inner products between sentences?} \end{bmatrix}, \quad y = \begin{bmatrix} +1 \\ -1 \\ -1 \\ +1 \end{bmatrix}. \]

• How should we represent sentences using features?
A (Bad) Universal Representation

- Treat character in position ‘j’ of the sentence as a categorical feature.
  - “fais ce que tu veux” => $x_i = \text{[f a i s “ c e “ q u e “ t u “ v e u x .]}$

- “Pad” end of the sentence up to maximum #characters:
  - “fais ce que tu veux” => $x_i = \text{[f a i s “ c e “ q u e “ t u “ v e u x . γ γ γ γ γ γ γ γ γ ...]}$

- Advantage:
  - No information is lost, KNN can eventually solve the problem.

- Disadvantage: throws out everything we know about language.
  - Needs to learn that “veux” starting from any position indicates “French”.
    - Doesn’t even use that sentences are made of words (this must be learned).
  - High overfitting risk, you will need a lot of examples for this easy task.
Bag of Words Representation

- **Bag of words** represents sentences/documents by **word counts**:

  The *International Conference on Machine Learning* (ICML) is the leading international *academic conference* in *machine learning*

<table>
<thead>
<tr>
<th>ICML</th>
<th>International</th>
<th>Conference</th>
<th>Machine</th>
<th>Learning</th>
<th>Leading</th>
<th>Academic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- **Bag of words** **loses a ton of information/meaning**:
  - But it **easily solves language identification problem**
Universal Representation vs. Bag of Words

• Why is bag of words better than “string of characters” here?

  – It needs less data because it captures invariances for the task:
    • Most features give strong indication of one language or the other.
    • It doesn’t matter where the French words appear.

  – It overfits less because it throws away irrelevant information.
    • Exact sequence of words isn’t particularly relevant here.
Text Example 2: Word Sense Disambiguation

• Consider the following two sentences:
  – “The cat ran after the mouse.”
  – “Move the mouse cursor to the File menu.”

• Word sense disambiguation (WSD): classify “meaning” of a word:
  – A surprisingly difficult task.

• You can do ok with bag of words, but it will have problems:
  – “Her mouse clicked on one cat video after another.”
  – “We saw the mouse run out from behind the computer.”
  – “The mouse was gray.” (irreducible error without more context)
Bigrams and Trigrams

- A **bigram** is an ordered set of two words:
  - Like “computer mouse” or “mouse ran”.
- A **trigram** is an ordered set of three words:
  - Like “cat and mouse” or “clicked mouse on”.

- These give more context/meaning than bag of words:
  - Includes neighbouring words as well as order of words.
  - Trigrams are widely-used for various language tasks.

- General case is called **n-gram**.
  - Unfortunately, **coupon collecting** becomes a problem with larger ‘n’.
Text Example 3: Part of Speech (POS) Tagging

• Consider problem of finding the verb in a sentence:
  – “The MDS students jumped at the chance to hear about POS features.”

• Part of speech (POS) tagging is problem of labeling all words.
  – 45 common syntactic POS tags.
  – Current systems have ~97% accuracy (it’s easier than WSD).
  – You can achieve this by applying “word-level” classifier to each word.

• What features of a word should we use for POS tagging?
POS Features

• Logistic regression with these 19 features gives ~97% accuracy:
  – Categorical features whose domain is all words ("lexical" features):
    • The word (e.g., ”jumped” is usually a verb).
    • The previous word (e.g., “he” hit vs. “a” hit).
    • The previous previous word.
    • The next word.
    • The next next word.
  – Categorical features whose domain is combinations of letters ("stem" features):
    • Prefix of length 1 (”what letter does the word start with?”)
    • Prefix of length 2.
    • Prefix of length 3.
    • Prefix of length 4 (”does it start with JUMP?”)
    • Suffix of length 1.
    • Suffix of length 2.
    • Suffix of length 3 (”does it end in ING?”)
    • Suffix of length 4.
  – Binary features ("shape" features):
    • Does word contain a number?
    • Does word contain a capital?
    • Does word contain a hyphen?