Today’s Quiz

Q1. Consider the following dataset with only one attribute:
\[ D = \{2, 5, 10, 12, 3, 20, 30, 11, 25\} \]
Suppose that we want to cluster these data points into 2 clusters. Follow the K-means algorithm with \( K=2 \), initial centroids at 3 and 18, and distance metric \( |x - x'| \). Show you work in a table such as:

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Centroid1</th>
<th>Centroid2</th>
<th>Cluster 1</th>
<th>Cluster 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>18</td>
<td>(2, 5, 10, 3)</td>
<td>(12, 20, 30, 11, 25)</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q2. How might you modify the K-means algorithm such that it returns clusters containing (near-)equal number of data points?
### Mini-project #3

#### Public leaderboard

<table>
<thead>
<tr>
<th>#</th>
<th>Team Name</th>
<th>Score</th>
<th>Entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Duck Duck Duck</td>
<td>0.97650</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>pierthodo</td>
<td>0.96570</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>YEL</td>
<td>0.96310</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>AshjonLi</td>
<td>0.95450</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>new APS</td>
<td>0.95170</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>TeamDRA</td>
<td>0.95140</td>
<td>13</td>
</tr>
<tr>
<td>7</td>
<td>OurNeuralNetsTwerk</td>
<td>0.95130</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>Magenta</td>
<td>0.94900</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Alanyi</td>
<td>0.94100</td>
<td>14</td>
</tr>
</tbody>
</table>

#### Private leaderboard

<table>
<thead>
<tr>
<th>Team Name</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duck Duck Duck</td>
<td>0.97650</td>
</tr>
<tr>
<td>pierthodo</td>
<td>0.96510</td>
</tr>
<tr>
<td>YEL</td>
<td>0.96230</td>
</tr>
<tr>
<td>AshjonLi</td>
<td>0.95690</td>
</tr>
<tr>
<td>APS</td>
<td>0.95680</td>
</tr>
<tr>
<td>OurNeuralNetsTwerk</td>
<td>0.95410</td>
</tr>
<tr>
<td>TeamDRA</td>
<td>0.95150</td>
</tr>
<tr>
<td>Magenta</td>
<td>0.95100</td>
</tr>
<tr>
<td>Alanyi</td>
<td>0.94600</td>
</tr>
</tbody>
</table>

#### 2014:

<table>
<thead>
<tr>
<th>Team Name</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duck Duck Duck</td>
<td>0.97650</td>
</tr>
<tr>
<td>pierthodo</td>
<td>0.96510</td>
</tr>
<tr>
<td>YEL</td>
<td>0.96230</td>
</tr>
<tr>
<td>AshjonLi</td>
<td>0.95690</td>
</tr>
<tr>
<td>APS</td>
<td>0.95680</td>
</tr>
<tr>
<td>OurNeuralNetsTwerk</td>
<td>0.95410</td>
</tr>
<tr>
<td>TeamDRA</td>
<td>0.95150</td>
</tr>
<tr>
<td>Magenta</td>
<td>0.95100</td>
</tr>
<tr>
<td>Alanyi</td>
<td>0.94600</td>
</tr>
</tbody>
</table>
Online learning

- Prediction problems where data arrives in a stream:
  - Weather forecasting
  - Stock market prediction
  - Ad placement on webpages

- Need to make a prediction for each data item as it arrives.
- Algorithm receives feedback about each prediction, and adjusts its decision strategy based on this information.

- Alternative scenario: We already have entire dataset but want to learn online because there are advantages to streaming it!

Different types of online learning

- **Learning from time-series data.**
  - Examples (possibly correlated) are given in a stream, and accumulated, then the learner must make a prediction.

- **Online learning with data from stationary distribution.** *(Today’s class)*
  - Examples are given in a stream, from a fixed distribution.
  - After each example (or mini-batch), make a prediction and adjust the strategy of the learner.

- **Online learning with data from a non-stationary distribution.**
  - Examples are given in a stream, from a distribution that can change over time, possibly in response to the learner’s predictions (incl. to make it harder for the learner).
  - After each example (or mini-batch), make a prediction and adjust the strategy of the learner.
Linear learning

• Basic setup:
  - Input features: $x \in \mathbb{R}^n$
  - Output labels: $y \in \{0, 1\}$
  - Goal: Find weight vector $w$ such that $f_w(x) = w^T x$ is close to $y$.

Online linear learning

• Initialize $w$ for all feature weights.
• Repeat:
  - Get an input feature vector: $x \in \mathbb{R}^n$
  - Make a linear prediction: $f_w(x) = w^T x$
  - Observe the label: $y \in \{0, 1\}$
  - Update weight vector $w$ so that $f_w(x)$ is closer to $y$. 
Online linear learning

• Initialize \( w \) for all feature weights.

• Repeat:
  - Get an input feature vector: \( x \in \mathbb{R}^n \)
  - Make a logistic prediction: \( f_w(x) = 1 / (1 + \exp(-w^T x)) \)
  - Observe the label: \( y \in \{0, 1\} \)
  - Update weight vector \( w \) so that \( f_w(x) \) is closer to \( y \).

\[
w_{k+1} = w_k + \alpha_k (y - f_w(x)) x
\]
An example

A larger example

- The Reuters News Corpus (RCV1) dataset:
  - 781K examples
  - 60M nonzero features
  - 1.1GB

- Format: label | sparse features...
  - ECAT | 45:1.2342  67:999834
  - CCAT | 9:5.6  23:678.3
Vision example

Typical sliding window approach to tracking-by-detection

Improved tracking using online-learning

Reasons for online learning

- Fast convergence to a good predictor.

- RAM efficient. You need to store only one example in RAM rather than all of them -> Entirely new scales of data are possible!

- Online Learning = Online optimization, which allows us to solve new problems (e.g., adversarial)

- Online Learning = ability to deal with drifting distributions.
Defining updates

• Define a loss function:
  
  \[ L(y, f_w(x)) \]

• Update according to :
  
  \[ w \leftarrow w + \alpha \frac{\partial L(y, f_w(x))}{\partial w} \]

• This is just **Stochastic Gradient Descent** (SGD).

• Useful for a large range of algorithms: linear, SVMs, neural nets, ...

Faster optimization?

• In optimization, stochastic gradient descent doesn’t usually perform as well as 2\textsuperscript{nd} order methods.

• For machine learning, if we optimize too well, we overfit!
Using SGD

- SGD is fast on large datasets because it exploits the redundancy in the data.
  - Update on 1 example may be very similar to update on another similar example.
  - SGD is slow near a solution because of gradient noise; learning rate must decrease. Fortunately, we don’t care!

- Sometimes worthwhile to run algorithm in online mode, even if all data is available as an initial batch. Or use a mini-batch (constant memory cost.)

- Lots of tricks available to adapt the learning rate online.

Non-convex optimization

- Generalized linear models have (generally) convex loss functions.

- SVMs have convex loss functions, but inequality constraints can make the problem difficult.

- Models with non-convex loss functions:
  - Discriminative training of mixture models.
  - Models with “products” of parameters.
  - Models with complex non-linearities after parameters.

- However: Using a “good” architecture can be more important than insisting on convexity (e.g. shallow vs “deep” NN).
Online Naïve Bayes

• Track the parameters online:
  - $\theta_1 = \frac{\text{# examples of } y=1}{\text{number of examples}}$
  - $\theta_{1,1} = \frac{\text{# examples of } x=1 \text{ and } y=1}{\text{number of examples where } y=1}$
  - $\theta_{1,0} = \frac{\text{# examples of } x=1 \text{ and } y=0}{\text{number of examples where } y=0}$

• Estimate log-likelihood ratio using online estimates of the parameters.

\[
\log \frac{P(y=1)}{P(y=0)} + \sum_j \log \frac{P(x_j | y=1)}{P(x_j | y=0)}
\]

Online versions of other ML algorithms?

• Linear discriminant analysis?
  - Similar to Naïve Bayes, use online estimate of the parameters and recompute the decision boundary.

• K-nearest neighbour?
  - Easy! Use the current set of points for lazy estimation.

• Decision trees?
  - Hard! Need to re-balance the tree based on new data.

• Logistic regression?
  - Apply online (stochastic) gradient descent.

• Support vector machines?
  - Stochastic gradient descent + variants.

• Neural networks?
  - Stochastic gradient descent + variants.
Online bagging

Bagging($T,M$)

- For each $m \in \{1,2,\ldots,M\}$,
  - $T_m = \text{Sample_With_Replacement}(T,N)$
  - $h_m = L_b(T_m)$
- Return $\{h_1,h_2,\ldots,h_M\}$

T = training set
N = # training samples
M = # base models
h = set of base models
d = current example

Online Boosting

AdaBoost($\{(x,y),\ldots,(x,y)\}, L_b, M$)

- Initialize $D_1(n) = 1/N$ for all $n \in \{1,2,\ldots,N\}$.
- Do for $m = 1,2,\ldots,M$:
  - 1. Call $L_b$ with the distribution $D_m$.
  - 2. Get back a hypothesis $h_m : X \rightarrow Y$.
  - 3. Calculate the error of $h_m$ : $\epsilon_m = \sum_n h_m(x_n) \neq y_n D_m(n)$. If $\epsilon_m > 1/2$ then set $M = m - 1$ and abort this loop.
  - 4. Set $\beta_m = \frac{\epsilon_m^{1-\epsilon_m}}{\epsilon_m}$.
  - 5. Update distribution $D_{m+1}$:
    $$D_{m+1}(n) = \frac{D_m(n)}{Z_m} \times \begin{cases} 
    \beta_m & \text{if } h_m(x_n) = y_n \\
    1 & \text{otherwise}
  \end{cases}$$
  where $Z_m$ is a normalization constant chosen so that $D_{m+1}$ is a probability distribution.

- Output the final hypothesis: $h_{f_{\text{Ada}}}(x) = \arg\max_y \sum_{h_m} h_m(x) \neq y \log \frac{1}{\beta_m}$.
Online Boosting

http://ti.arc.nasa.gov/m/profile/oza/files/ozru01a.pdf

**OnlineBoosting**

- **OnlineBoosting**($h_m$, **OnlineBase**, $d$)
  - Set the example’s “weight” $\lambda_d = 1$.
  - For each base model $h_m$, ($m \in \{1, 2, \ldots, M\}$) in the ensemble.
    - 1. Set $k$ according to $\text{Poisson} (\lambda_d)$.
    - 2. Do $k$ times
      - $h_m = \text{OnlineBase} (h_m, d)$
    - 3. If $h_m (d)$ is the correct label,
      - $\lambda_m^+ \leftarrow \lambda_m^+ + \lambda_d$
      - $\lambda_d \leftarrow \lambda_d \left( \frac{\lambda_m^+}{\lambda_m^+ + \lambda_d} \right)$
    - else
      - $\lambda_m^- \leftarrow \lambda_m^- + \lambda_d$
      - $\lambda_d \leftarrow \lambda_d \left( \frac{\lambda_m^-}{\lambda_m^- + \lambda_d} \right)$

To classify new examples:

- For each $m \in \{1, 2, \ldots, M\}$
  - Calculate $\varepsilon_m = \frac{\lambda_m^+}{\lambda_m^+ + \lambda_m^-}$ and $\beta_m = \frac{\varepsilon_m}{1 - \varepsilon_m}$
  - Return $h(x) = \text{argmax}_{x \in C} \sum_{m \in \text{base sets}} \beta_m \log \frac{1}{\beta_m}$.

---

**Online Boosting diagram:** Each row represents a new example which is passed sequentially through each weak learner. Weights (indicated by the size of the boxes) are increased when examples are misclassified by previous learners.

---

**Analysis**

- **Objective for learning from streams of data:**
  - As the number of examples grow, the online version converges (in probability) to the same solution as the offline version.

- **Methods we discussed assume the distribution generating the data does not change.**

- **Much literature on online learning assumes data may come from a changing distribution, including by an adversary.**
  - As your predictions change, the distribution of data changes also.
Measuring performance

• **Offline**: Train on train set, then measure performance on test set.

• **Online**: **Progressive validation**
  - On timestep $t$, let $l_t = L(y_t, f_w(x_t))$
  - Report loss $L = E_t[l_t]$

• **Bound the number of errors**:
  Let $D$ be a distribution over $x, y$.
  Let $l'_t = E_{(x,y)\sim D} L(y, f_w(x))$
  **Theorem**: For all probability distributions $D(x, y)$, for all online learning algorithms, with probability $1 - \delta$, we have $|L - E_t[l'_t]| \leq \sqrt{\ln(2/\delta) / (2T)}$

Regret bounds

• A good online algorithm should quickly find a hypothesis $h \in H$ that makes few mistakes, compared to the best $h^* \in H$ it could have found given the same data sequence.

• Regret of algorithm using hypothesis $h \in H$, for not having instead used $h^* \in H$, when running on sequence of $T$ examples:
  $$\text{Regret}_T(h^*) = \sum_{t=1:T} L(h(x_t), y_t) - \sum_{t=1:T} L(h^*(x_t), y_t)$$

• Regret of the algorithm over the full hypothesis class:
  $$\text{Regret}_T(H) = \max_{h \in H} \text{Regret}_T(h^*)$$

• Learner wants to have lowest possible regret with respect to $H$. At the very least, regret should be sublinear in $T$. 
Final notes

- Significant material for today’s slides taken from:
  - http://courses.cs.washington.edu/courses/cse599s/12sp/

- Mini-project #3 peer reviews due Nov. 25.

- Midterm:
  - Friday November 20, 11:30am-1pm.
  - Divided in 2 rooms. Last name A-K: Burnside 1B36
    Last name L-Z: MAASS 10
  - Closed book. No calculator.
  - You can bring 1 page (2-sided) of hand-written notes.
  - Covers lectures 1-21.
  - Short answer questions, similar to practice questions from Nov.11.
  - Practice midterm from last year will be posted on discussion board.

- Course evaluations now available on Minerva. Please fill out!