Advanced Topics in Machine Learning: Part I

John Shawe-Taylor and Steffen Grünewalder
UCL

Second semester
2010

General Course Information

- Two main parts
  - Part I: JS-T and SG on delayed/partial feedback active learning
  - Part II: Massi Pontil on multi-task and multi-kernel learning
- Lectures:
  - Wednesdays 9-11 in Room G01 of the Charles Bell Building
  - 13-14 in Room B06 of Drayton Building
- Exam/coursework:
  - Written Examination (2.5 hours, 50%): The examination rubric is: There will be two sections: section A and B, each with two questions. You should answer just one question from each section.
  - Coursework Section (2 pieces, 50%)
- To pass this course, students must obtain an average of at least 50% when the coursework and exam components of the course are weighted together

General Course Information

- Two main parts
  - Part I: JS-T and SG on delayed/partial feedback active learning
  - Part II: Massi Pontil on multi-task and multi-kernel learning
- Lectures:
  - Wednesdays 9-11 in Room G01 of the Charles Bell Building
  - 13-14 in Room B06 of Drayton Building
- Exam/coursework:
  - Written Examination (2.5 hours, 50%): The examination rubric is: There will be two sections: section A and B, each with two questions. You should answer just one question from each section.
  - Coursework Section (2 pieces, 50%)
- To pass this course, students must obtain an average of at least 50% when the coursework and exam components of the course are weighted together

General Course Information

- Two main parts
  - Part I: JS-T and SG on delayed/partial feedback active learning
  - Part II: Massi Pontil on multi-task and multi-kernel learning
- Lectures:
  - Wednesdays 9-11 in Room G01 of the Charles Bell Building
  - 13-14 in Room B06 of Drayton Building
- Exam/coursework:
  - Written Examination (2.5 hours, 50%): The examination rubric is: There will be two sections: section A and B, each with two questions. You should answer just one question from each section.
  - Coursework Section (2 pieces, 50%)
- To pass this course, students must obtain an average of at least 50% when the coursework and exam components of the course are weighted together

General Course Information

- Two main parts
  - Part I: JS-T and SG on delayed/partial feedback active learning
  - Part II: Massi Pontil on multi-task and multi-kernel learning
- Lectures:
  - Wednesdays 9-11 in Room G01 of the Charles Bell Building
  - 13-14 in Room B06 of Drayton Building
- Exam/coursework:
  - Written Examination (2.5 hours, 50%): The examination rubric is: There will be two sections: section A and B, each with two questions. You should answer just one question from each section.
  - Coursework Section (2 pieces, 50%)
- To pass this course, students must obtain an average of at least 50% when the coursework and exam components of the course are weighted together

General Course Information

- Two main parts
  - Part I: JS-T and SG on delayed/partial feedback active learning
  - Part II: Massi Pontil on multi-task and multi-kernel learning
- Lectures:
  - Wednesdays 9-11 in Room G01 of the Charles Bell Building
  - 13-14 in Room B06 of Drayton Building
- Exam/coursework:
  - Written Examination (2.5 hours, 50%): The examination rubric is: There will be two sections: section A and B, each with two questions. You should answer just one question from each section.
  - Coursework Section (2 pieces, 50%)
- To pass this course, students must obtain an average of at least 50% when the coursework and exam components of the course are weighted together

General Course Information

- Two main parts
  - Part I: JS-T and SG on delayed/partial feedback active learning
  - Part II: Massi Pontil on multi-task and multi-kernel learning
- Lectures:
  - Wednesdays 9-11 in Room G01 of the Charles Bell Building
  - 13-14 in Room B06 of Drayton Building
- Exam/coursework:
  - Written Examination (2.5 hours, 50%): The examination rubric is: There will be two sections: section A and B, each with two questions. You should answer just one question from each section.
  - Coursework Section (2 pieces, 50%)
- To pass this course, students must obtain an average of at least 50% when the coursework and exam components of the course are weighted together

General Course Information

- Two main parts
  - Part I: JS-T and SG on delayed/partial feedback active learning
  - Part II: Massi Pontil on multi-task and multi-kernel learning
- Lectures:
  - Wednesdays 9-11 in Room G01 of the Charles Bell Building
  - 13-14 in Room B06 of Drayton Building
- Exam/coursework:
  - Written Examination (2.5 hours, 50%): The examination rubric is: There will be two sections: section A and B, each with two questions. You should answer just one question from each section.
  - Coursework Section (2 pieces, 50%)
- To pass this course, students must obtain an average of at least 50% when the coursework and exam components of the course are weighted together

General Course Information

- Two main parts
  - Part I: JS-T and SG on delayed/partial feedback active learning
  - Part II: Massi Pontil on multi-task and multi-kernel learning
- Lectures:
  - Wednesdays 9-11 in Room G01 of the Charles Bell Building
  - 13-14 in Room B06 of Drayton Building
- Exam/coursework:
  - Written Examination (2.5 hours, 50%): The examination rubric is: There will be two sections: section A and B, each with two questions. You should answer just one question from each section.
  - Coursework Section (2 pieces, 50%)
- To pass this course, students must obtain an average of at least 50% when the coursework and exam components of the course are weighted together

General Course Information

- Two main parts
  - Part I: JS-T and SG on delayed/partial feedback active learning
  - Part II: Massi Pontil on multi-task and multi-kernel learning
- Lectures:
  - Wednesdays 9-11 in Room G01 of the Charles Bell Building
  - 13-14 in Room B06 of Drayton Building
- Exam/coursework:
  - Written Examination (2.5 hours, 50%): The examination rubric is: There will be two sections: section A and B, each with two questions. You should answer just one question from each section.
  - Coursework Section (2 pieces, 50%)
- To pass this course, students must obtain an average of at least 50% when the coursework and exam components of the course are weighted together

General Course Information

- Two main parts
  - Part I: JS-T and SG on delayed/partial feedback active learning
  - Part II: Massi Pontil on multi-task and multi-kernel learning
- Lectures:
  - Wednesdays 9-11 in Room G01 of the Charles Bell Building
  - 13-14 in Room B06 of Drayton Building
- Exam/coursework:
  - Written Examination (2.5 hours, 50%): The examination rubric is: There will be two sections: section A and B, each with two questions. You should answer just one question from each section.
  - Coursework Section (2 pieces, 50%)
- To pass this course, students must obtain an average of at least 50% when the coursework and exam components of the course are weighted together

Course Information for Part I

- Lecturers: John Shawe-Taylor (<jst@cs.ucl.ac.uk>) and Steffen Grünewalder (<S.Grunewalder@cs.ucl.ac.uk>)
- One piece of coursework
- Delayed/partial feedback active learning:
  - the Bandit problem: simple and multivariate bandits (JS-T)
  - Gaussian processes (Steffen)
  - Kalman filters (Simon Julier)
  - Gaussian process bandits (JS-T)
  - Function approximation and reinforcement learning (David Silver)

Course Information for Part I

- Lecturers: John Shawe-Taylor (<jst@cs.ucl.ac.uk>) and Steffen Grünewalder (<S.Grunewalder@cs.ucl.ac.uk>)
- One piece of coursework
- Delayed/partial feedback active learning:
  - the Bandit problem: simple and multivariate bandits (JS-T)
  - Gaussian processes (Steffen)
  - Kalman filters (Simon Julier)
  - Gaussian process bandits (JS-T)
  - Function approximation and reinforcement learning (David Silver)

Course Information for Part I

- Lecturers: John Shawe-Taylor (<jst@cs.ucl.ac.uk>) and Steffen Grünewalder (<S.Grunewalder@cs.ucl.ac.uk>)
- One piece of coursework
- Delayed/partial feedback active learning:
  - the Bandit problem: simple and multivariate bandits (JS-T)
  - Gaussian processes (Steffen)
  - Kalman filters (Simon Julier)
  - Gaussian process bandits (JS-T)
  - Function approximation and reinforcement learning (David Silver)

Course Information for Part I

- Lecturers: John Shawe-Taylor (<jst@cs.ucl.ac.uk>) and Steffen Grünewalder (<S.Grunewalder@cs.ucl.ac.uk>)
- One piece of coursework
- Delayed/partial feedback active learning:
  - the Bandit problem: simple and multivariate bandits (JS-T)
  - Gaussian processes (Steffen)
  - Kalman filters (Simon Julier)
  - Gaussian process bandits (JS-T)
  - Function approximation and reinforcement learning (David Silver)

Course Information for Part I

- Lecturers: John Shawe-Taylor (<jst@cs.ucl.ac.uk>) and Steffen Grünewalder (<S.Grunewalder@cs.ucl.ac.uk>)
- One piece of coursework
- Delayed/partial feedback active learning:
  - the Bandit problem: simple and multivariate bandits (JS-T)
  - Gaussian processes (Steffen)
  - Kalman filters (Simon Julier)
  - Gaussian process bandits (JS-T)
  - Function approximation and reinforcement learning (David Silver)

Course Information for Part I

- Lecturers: John Shawe-Taylor (<jst@cs.ucl.ac.uk>) and Steffen Grünewalder (<S.Grunewalder@cs.ucl.ac.uk>)
- One piece of coursework
- Delayed/partial feedback active learning:
  - the Bandit problem: simple and multivariate bandits (JS-T)
  - Gaussian processes (Steffen)
  - Kalman filters (Simon Julier)
  - Gaussian process bandits (JS-T)
  - Function approximation and reinforcement learning (David Silver)

Course Information for Part I

- Lecturers: John Shawe-Taylor (<jst@cs.ucl.ac.uk>) and Steffen Grünewalder (<S.Grunewalder@cs.ucl.ac.uk>)
- One piece of coursework
- Delayed/partial feedback active learning:
  - the Bandit problem: simple and multivariate bandits (JS-T)
  - Gaussian processes (Steffen)
  - Kalman filters (Simon Julier)
  - Gaussian process bandits (JS-T)
  - Function approximation and reinforcement learning (David Silver)

Course Information for Part I

- Lecturers: John Shawe-Taylor (<jst@cs.ucl.ac.uk>) and Steffen Grünewalder (<S.Grunewalder@cs.ucl.ac.uk>)
- One piece of coursework
- Delayed/partial feedback active learning:
  - the Bandit problem: simple and multivariate bandits (JS-T)
  - Gaussian processes (Steffen)
  - Kalman filters (Simon Julier)
  - Gaussian process bandits (JS-T)
  - Function approximation and reinforcement learning (David Silver)

1. The Bandit Problem
   - Definition of the Bandit Problem
   - Key Issues for the Bandit problem
2. Methods of Solution
   - Bayesian analysis
   - Gittins Indices
   - Upper confidence bounds
   - More complex decisions
3. Multivariate bandits
   - Definitions
   - Linrel
   - Kernel LinRel
   - Implementation issues

Multi-armed Bandit (MAB) problem

- Modelling of a casino with a finite collection of K slot machines (aka one-armed bandits):
- Learning proceeds in iterations: discrete time slots, $t = 1, 2, \ldots,$
- At each time the player must decide which machine to play
- After playing machine i the player either receives a (randomised) reward R_i, e.g. unit of reward or nothing
- Each machine i has a fixed (but not known to the player) reward distribution with mean μ_i, e.g. probability p_i of giving a reward
- The goal of the player is to maximise his reward:
  - Could be over a fixed (known) number T of plays or horizon
  - Alternatively maximising the accumulated reward at any time: any time performance

Multi-armed Bandit (MAB) problem

- Modelling of a casino with a finite collection of K slot machines (aka one-armed bandits):
- Learning proceeds in iterations: discrete time slots, $t = 1, 2, \ldots,$
- At each time the player must decide which machine to play
- After playing machine i the player either receives a (randomised) reward R_i, e.g. unit of reward or nothing
- Each machine i has a fixed (but not known to the player) reward distribution with mean μ_i, e.g. probability p_i of giving a reward
- The goal of the player is to maximise his reward:
  - Could be over a fixed (known) number T of plays or horizon
  - Alternatively maximising the accumulated reward at any time: any time performance

Multi-armed Bandit (MAB) problem

- Modelling of a casino with a finite collection of K slot machines (aka one-armed bandits):
- Learning proceeds in iterations: discrete time slots, $t = 1, 2, \ldots,$
- At each time the player must decide which machine to play
- After playing machine i the player either receives a (randomised) reward R_i, e.g. unit of reward or nothing
- Each machine i has a fixed (but not known to the player) reward distribution with mean μ_i, e.g. probability p_i of giving a reward
- The goal of the player is to maximise his reward:
  - Could be over a fixed (known) number T of plays or horizon
  - Alternatively maximising the accumulated reward at any time: any time performance

Multi-armed Bandit (MAB) problem

- Modelling of a casino with a finite collection of K slot machines (aka one-armed bandits):
- Learning proceeds in iterations: discrete time slots, $t = 1, 2, \ldots,$
- At each time the player must decide which machine to play
- After playing machine i the player either receives a (randomised) reward R_i, e.g. unit of reward or nothing
- Each machine i has a fixed (but not known to the player) reward distribution with mean μ_i, e.g. probability p_i of giving a reward
- The goal of the player is to maximise his reward:
  - Could be over a fixed (known) number T of plays or horizon
  - Alternatively maximising the accumulated reward at any time: any time performance

Multi-armed Bandit (MAB) problem

- Modelling of a casino with a finite collection of K slot machines (aka one-armed bandits):
- Learning proceeds in iterations: discrete time slots, $t = 1, 2, \ldots,$
- At each time the player must decide which machine to play
- After playing machine i the player either receives a (randomised) reward R_i, e.g. unit of reward or nothing
- Each machine i has a fixed (but not known to the player) reward distribution with mean μ_i, e.g. probability p_i of giving a reward
- The goal of the player is to maximise his reward:
  - Could be over a fixed (known) number T of plays or horizon
  - Alternatively maximising the accumulated reward at any time: any time performance

Multi-armed Bandit (MAB) problem

- Modelling of a casino with a finite collection of K slot machines (aka one-armed bandits):
- Learning proceeds in iterations: discrete time slots, $t = 1, 2, \ldots,$
- At each time the player must decide which machine to play
- After playing machine i the player either receives a (randomised) reward R_i, e.g. unit of reward or nothing
- Each machine i has a fixed (but not known to the player) reward distribution with mean μ_i, e.g. probability p_i of giving a reward
- The goal of the player is to maximise his reward:
  - Could be over a fixed (known) number T of plays or horizon
  - Alternatively maximising the accumulated reward at any time: any time performance

Multi-armed Bandit (MAB) problem

- Modelling of a casino with a finite collection of K slot machines (aka one-armed bandits):
- Learning proceeds in iterations: discrete time slots, $t = 1, 2, \ldots,$
- At each time the player must decide which machine to play
- After playing machine i the player either receives a (randomised) reward R_i, e.g. unit of reward or nothing
- Each machine i has a fixed (but not known to the player) reward distribution with mean μ_i, e.g. probability p_i of giving a reward
- The goal of the player is to maximise his reward:
    - Could be over a fixed (known) number T of plays or horizon
    - Alternatively maximising the accumulated reward at any time: any time performance

Multi-armed Bandit (MAB) problem

- Modelling of a casino with a finite collection of K slot machines (aka one-armed bandits):
- Learning proceeds in iterations: discrete time slots, $t = 1, 2, \ldots,$
- At each time the player must decide which machine to play
- After playing machine i the player either receives a (randomised) reward R_i, e.g. unit of reward or nothing
- Each machine i has a fixed (but not known to the player) reward distribution with mean μ_i, e.g. probability p_i of giving a reward
- The goal of the player is to maximise his reward:
  - Could be over a fixed (known) number T of plays or horizon
  - Alternatively maximising the accumulated reward at any time: any time performance

Key issues posed by the BP

- Exploration versus exploitation
  - We are having to explore the different machines to estimate their returns (exploration)
  - We want to play the machine that we think is best (exploitation)
  - For example at the beginning of the session we have no reason to believe that any arm is better than another, so we must choose randomly - i.e. perform exploration
  - For the final play of a fixed length session, we cannot make use of any information learned and so must exploit
- Any algorithm must somehow trade between the two

Key issues posed by the BP

- Exploration versus exploitation
  - We are having to explore the different machines to estimate their returns (exploration)
  - We want to play the machine that we think is best (exploitation)
  - For example at the beginning of the session we have no reason to believe that any arm is better than another, so we must choose randomly - i.e. perform exploration
  - For the final play of a fixed length session, we cannot make use of any information learned and so must exploit
- Any algorithm must somehow trade between the two

Key issues posed by the BP

- Exploration versus exploitation
  - We are having to explore the different machines to estimate their returns (exploration)
  - We want to play the machine that we think is best (exploitation)
  - For example at the beginning of the session we have no reason to believe that any arm is better than another, so we must choose randomly - i.e. perform exploration
  - For the final play of a fixed length session, we cannot make use of any information learned and so must exploit
- Any algorithm must somehow trade between the two

Key issues posed by the BP

- Exploration versus exploitation
  - We are having to explore the different machines to estimate their returns (exploration)
  - We want to play the machine that we think is best (exploitation)
  - For example at the beginning of the session we have no reason to believe that any arm is better than another, so we must choose randomly - i.e. perform exploration
  - For the final play of a fixed length session, we cannot make use of any information learned and so must exploit
- Any algorithm must somehow trade between the two

Key issues posed by the BP

- Exploration versus exploitation
  - We are having to explore the different machines to estimate their returns (exploration)
  - We want to play the machine that we think is best (exploitation)
  - For example at the beginning of the session we have no reason to believe that any arm is better than another, so we must choose randomly - i.e. perform exploration
  - For the final play of a fixed length session, we cannot make use of any information learned and so must exploit
- Any algorithm must somehow trade between the two

Key issues posed by the BP

- Exploration versus exploitation
  - We are having to explore the different machines to estimate their returns (exploration)
  - We want to play the machine that we think is best (exploitation)
  - For example at the beginning of the session we have no reason to believe that any arm is better than another, so we must choose randomly - i.e. perform exploration
  - For the final play of a fixed length session, we cannot make use of any information learned and so must exploit
- Any algorithm must somehow trade between the two

Measures of performance

- Performance is usually measured by the regret: after T rounds, the regret is

$$\rho_T = T\mu^\star - \sum_{t=1}^{T} \hat{r}_t,$$

where $\mu^\star = \max_{1 \leq i \leq K}\{\mu_i\}$ is the maximal expected reward that can be achieved in each round and $\hat{r}_T$ is the actual reward received in round t.

- Note that it is possible for the regret to be negative in a single run, but averaged over a number of runs it will always be positive
- We also use $r_{i,\tau}$ for the reward received when the ith arm is pulled for the τth time

Measures of performance

- Performance is usually measured by the regret: after T rounds, the regret is

$$\rho_T = T\mu^\star - \sum_{t=1}^{T} \hat{r}_t,$$

where $\mu^\star = \max_{1 \le i \le K}\{\mu_i\}$ is the maximal expected reward that can be achieved in each round and $\hat{r}_T$ is the actual reward received in round t.

- Note that it is possible for the regret to be negative in a single run, but averaged over a number of runs it will always be positive
- We also use $r_{i,\tau}$ for the reward received when the ith arm is pulled for the τth time

Measures of performance

- Performance is usually measured by the regret: after T rounds, the regret is

$$\rho_T = T\mu^\star - \sum_{t=1}^{T} \hat{r}_t,$$

where $\mu^\star = \max_{1 \le i \le K}\{\mu_i\}$ is the maximal expected reward that can be achieved in each round and $\hat{r}_T$ is the actual reward received in round t.
- Note that it is possible for the regret to be negative in a single run, but averaged over a number of runs it will always be positive
- We also use $r_{i,\tau}$ for the reward received when the ith arm is pulled for the τth time

Relation to Markov Decision Processes (MDP)

- Markov decision process is a 4-tuple (S, A, P, R) where
  - S is a finite set of states
  - A is finite set of actions
  - $P(\cdot|s, a)$ is the probability distribution over states reached after taking action a in state s
  - $R(s, a)$ is immediate reward on taking action a in state s: note this could be a random variable and is sometimes assumed to be independent of a
- The problem is to find an 'optimal' policy (that is a mapping π from states to actions) that maximises the expected reward computed with a discount factor or finite horizon:

$$\sum_{t=0}^{T} \gamma^t R(s_0, a_t), \quad \text{with } T < \infty \text{ if } \gamma = 1.$$

Relation to Markov Decision Processes (MDP)

- Markov decision process is a 4-tuple (S, A, P, R) where
  - S is a finite set of states
  - A is finite set of actions
  - $P(\cdot|s, a)$ is the probability distribution over states reached after taking action a in state s
  - $R(s, a)$ is immediate reward on taking action a in state s: note this could be a random variable and is sometimes assumed to be independent of a
- The problem is to find an 'optimal' policy (that is a mapping π from states to actions) that maximises the expected reward computed with a discount factor or finite horizon:

$$\sum_{t=0}^{T} \gamma^t R(s_0, a_t), \quad \text{with } T < \infty \text{ if } \gamma = 1.$$

Relation to Markov Decision Processes (MDP)

- Markov decision process is a 4-tuple (S, A, P, R) where
  - S is a finite set of states
  - A is finite set of actions
  - $P(\cdot|s, a)$ is the probability distribution over states reached after taking action a in state s
  - $R(s, a)$ is immediate reward on taking action a in state s: note this could be a random variable and is sometimes assumed to be independent of a
- The problem is to find an 'optimal' policy (that is a mapping π from states to actions) that maximises the expected reward computed with a discount factor or finite horizon:

$$\sum_{t=0}^{T} \gamma^t R(s_0, a_t), \quad \text{with } T < \infty \text{ if } \gamma = 1.$$

Relation to Markov Decision Processes (MDP)

- Markov decision process is a 4-tuple (S, A, P, R) where
  - S is a finite set of states
  - A is finite set of actions
  - $P(\cdot|s, a)$ is the probability distribution over states reached after taking action a in state s
  - $R(s, a)$ is immediate reward on taking action a in state s: note this could be a random variable and is sometimes assumed to be independent of a
- The problem is to find an ‘optimal’ policy (that is a mapping π from states to actions) that maximises the expected reward computed with a discount factor or finite horizon:

$$\sum_{t=0}^{T} \gamma^t R(s_0, a_t), \quad \text{with } T < \infty \text{ if } \gamma = 1.$$

Relation to Markov Decision Processes (MDP)

- Markov decision process is a 4-tuple (S, A, P, R) where
  - S is a finite set of states
  - A is finite set of actions
  - $P(\cdot|s, a)$ is the probability distribution over states reached after taking action a in state s
  - $R(s, a)$ is immediate reward on taking action a in state s: note this could be a random variable and is sometimes assumed to be independent of a
- The problem is to find an 'optimal' policy (that is a mapping π from states to actions) that maximises the expected reward computed with a discount factor or finite horizon:

$$\sum_{t=0}^{T} \gamma^t R(s_0, a_t), \quad \text{with } T < \infty \text{ if } \gamma = 1.$$

Relation to Markov Decision Processes (MDP)

- Markov decision process is a 4-tuple (S, A, P, R) where
  - S is a finite set of states
  - A is finite set of actions
  - $P(\cdot|s, a)$ is the probability distribution over states reached after taking action a in state s
  - $R(s, a)$ is immediate reward on taking action a in state s: note this could be a random variable and is sometimes assumed to be independent of a
- The problem is to find an 'optimal' policy (that is a mapping π from states to actions) that maximises the expected reward computed with a discount factor or finite horizon:

$$\sum_{t=0}^{T} \gamma^t R(s_0, a_t), \quad \text{with } T < \infty \text{ if } \gamma = 1.$$

MAB as MDPs

- Multi-armed bandit is MDP with $|S| = 1$:
  - $S = \{s_0\}$
  - $A = \{1, \ldots, k\}$ is finite set of arms
  - $P(s_0|s_0, i) = 1$ for all i, i.e. always stays in the single state
  - $R(s_0, i) = R_i$ is the reward on pulling arm i
- The problem is to learn a policy (that is a mapping π_t to actions) that maximises the expected reward computed with a discount factor or finite horizon:

$$\sum_{t=0}^{T} \gamma^t R_{\pi_t}, \quad \text{with } T < \infty \text{ if } \gamma = 1.$$

MAB as MDPs

- Multi-armed bandit is MDP with $|S| = 1$:
  - $S = \{s_0\}$
  - $A = \{1, \ldots, k\}$ is finite set of arms
  - $P(s_0|s_0, i) = 1$ for all i, i.e. always stays in the single state
  - $R(s_0, i) = R_i$ is the reward on pulling arm i
- The problem is to learn a policy (that is a mapping π_t to actions) that maximises the expected reward computed with a discount factor or finite horizon:

$$\sum_{t=0}^{T} \gamma^t R_{\pi_t}, \quad \text{with } T < \infty \text{ if } \gamma = 1.$$

MAB as MDPs

- Multi-armed bandit is MDP with $|S| = 1$:
  - $S = \{s_0\}$
  - $A = \{1, \ldots, k\}$ is finite set of arms
  - $P(s_0|s_0, i) = 1$ for all i, i.e. always stays in the single state
  - $R(s_0, i) = R_i$ is the reward on pulling arm i
- The problem is to learn a policy (that is a mapping π_t to actions) that maximises the expected reward computed with a discount factor or finite horizon:

$$\sum_{t=0}^{T} \gamma^t R_{\pi_t}, \quad \text{with } T < \infty \text{ if } \gamma = 1.$$

MAB as MDPs

- Multi-armed bandit is MDP with $|S| = 1$:
  - $S = \{s_0\}$
  - $A = \{1, \ldots, k\}$ is finite set of arms
  - $P(s_0|s_0, i) = 1$ for all i, i.e. always stays in the single state
  - $R(s_0, i) = R_i$ is the reward on pulling arm i
- The problem is to learn a policy (that is a mapping π_t to actions) that maximises the expected reward computed with a discount factor or finite horizon:

$$\sum_{t=0}^{T} \gamma^t R_{\pi_t}, \quad \text{with } T < \infty \text{ if } \gamma = 1.$$

MAB as MDPs

- Multi-armed bandit is MDP with $|S| = 1$:
  - $S = \{s_0\}$
  - $A = \{1, \ldots, k\}$ is finite set of arms
  - $P(s_0|s_0, i) = 1$ for all i, i.e. always stays in the single state
  - $R(s_0, i) = R_i$ is the reward on pulling arm i
- The problem is to learn a policy (that is a mapping π_t to actions) that maximises the expected reward computed with a discount factor or finite horizon:

$$\sum_{t=0}^{T} \gamma^t R_{\pi_t}, \quad \text{with } T < \infty \text{ if } \gamma = 1.$$

MAB as MDPs

- Multi-armed bandit is MDP with $|S| = 1$:
  - $S = \{s_0\}$
  - $A = \{1, \ldots, k\}$ is finite set of arms
  - $P(s_0|s_0, i) = 1$ for all i, i.e. always stays in the single state
  - $R(s_0, i) = R_i$ is the reward on pulling arm i
- The problem is to learn a policy (that is a mapping π_t to actions) that maximises the expected reward computed with a discount factor or finite horizon:

$$\sum_{t=0}^{T} \gamma^t R_{\pi_t}, \quad \text{with } T < \infty \text{ if } \gamma = 1.$$

Bayesian analysis

- Part of the problem of deciding which arm to play is keeping an estimate of the performance of each arm
- In the case where the rewards are binary with a fixed probability p_i it is natural to use a Bayesian approach with the conjugate prior the Beta distribution over possible values of p_i with hyperparameters α and β

$$P(p_i; \alpha, \beta) \propto p_i^{\alpha-1}(1-p_i)^{\beta-1}$$

- The normalising constant is

$$\int_0^1 p^{\alpha-1}(1-p)^{\beta-1} dp = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} = \frac{1}{B(\alpha, \beta)}$$

Bayesian analysis

- Part of the problem of deciding which arm to play is keeping an estimate of the performance of each arm
- In the case where the rewards are binary with a fixed probability p_i it is natural to use a Bayesian approach with the conjugate prior the Beta distribution over possible values of p_i with hyperparameters α and β

$$P(p_i; \alpha, \beta) \propto p_i^{\alpha-1}(1-p_i)^{\beta-1}$$

- The normalising constant is

$$\int_0^1 p^{\alpha-1}(1-p)^{\beta-1} dp = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} = \frac{1}{B(\alpha,\beta)}$$

Bayesian analysis

- Part of the problem of deciding which arm to play is keeping an estimate of the performance of each arm
- In the case where the rewards are binary with a fixed probability p_i it is natural to use a Bayesian approach with the conjugate prior the Beta distribution over possible values of p_i with hyperparameters α and β

$$P(p_i; \alpha, \beta) \propto p_i^{\alpha-1}(1 - p_i)^{\beta-1}$$

- The normalising constant is

$$\int_0^1 p^{\alpha-1}(1-p)^{\beta-1}dp = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} = \frac{1}{B(\alpha,\beta)}$$

Bayesian analysis

- The expected value of the distribution is

$$\mathbb{E}_{p \sim P(p;\alpha,\beta)}[p] = \frac{\alpha}{\alpha+\beta}$$

- while its variance is

$$\sigma^2_{p \sim P(p;\alpha,\beta)} = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$$

- If we make an observation of a reward the posterior distribution satisfies

$$P(p|\alpha,\beta,r=1) \propto p^{\alpha-1}(1-p)^{\beta-1}p = p^{\alpha}(1-p)^{\beta-1}$$

  so that $P(p|\alpha,\beta,r=1) = P(p;\alpha+1,\beta)$
- Similarly

$$P(p|\alpha,\beta,r=0) = P(p;\alpha,\beta+1)$$

Bayesian analysis

- The expected value of the distribution is

$$\mathbb{E}_{p \sim P(p;\alpha,\beta)}[p] = \frac{\alpha}{\alpha + \beta}$$

- while its variance is

$$\sigma^2_{p \sim P(p;\alpha,\beta)} = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$$

- If we make an observation of a reward the posterior distribution satisfies

$$P(p|\alpha,\beta,r=1) \propto p^{\alpha-1}(1-p)^{\beta-1}p = p^{\alpha}(1-p)^{\beta-1}$$

so that $P(p|\alpha,\beta,r=1) = P(p;\alpha+1,\beta)$

- Similarly

$$P(p|\alpha,\beta,r=0) = P(p;\alpha,\beta+1)$$

Bayesian analysis

- The expected value of the distribution is

$$\mathbb{E}_{p \sim P(p;\alpha,\beta)}[p] = \frac{\alpha}{\alpha + \beta}$$

- while its variance is

$$\sigma^2_{p \sim P(p;\alpha,\beta)} = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$$

- If we make an observation of a reward the posterior distribution satisfies

$$P(p|\alpha,\beta,r=1) \propto p^{\alpha-1}(1-p)^{\beta-1}p = p^{\alpha}(1-p)^{\beta-1}$$

so that $P(p|\alpha,\beta,r=1) = P(p;\alpha+1,\beta)$

- Similarly

$$P(p|\alpha,\beta,r=0) = P(p;\alpha,\beta+1)$$

Bayesian analysis

- The expected value of the distribution is

$$\mathbb{E}_{p\sim P(p;\alpha,\beta)}[p] = \frac{\alpha}{\alpha+\beta}$$

- while its variance is

$$\sigma^2_{p\sim P(p;\alpha,\beta)} = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$$

- If we make an observation of a reward the posterior distribution satisfies

$$P(p|\alpha,\beta,r=1) \propto p^{\alpha-1}(1-p)^{\beta-1}p = p^{\alpha}(1-p)^{\beta-1}$$

so that $P(p|\alpha,\beta,r=1) = P(p;\alpha+1,\beta)$

- Similarly

$$P(p|\alpha,\beta,r=0) = P(p;\alpha,\beta+1)$$

Bayesian analysis

- A possible Bayesian strategy is to sample each arm's response rate according to its posterior distribution

$$p_j \sim P(p; \alpha + n_j \bar{r}_j, \beta + n_j(1 - \bar{r}_j))$$

  and then choose arm $j^\star = \mathrm{argmax}_{1 \leq j \leq K}\{p_j\}$
- this corresponds to selecting arm j with probability proportional to it's being the best arm in the posterior distribution
- arms for which we do not have accurate estimates for p_j will tend to be selected because their variance is high (exploration) – while arms with high p_j are also more likely to be selected (exploitation)

Bayesian analysis

- A possible Bayesian strategy is to sample each arm's response rate according to its posterior distribution

$$p_j \sim P(p; \alpha + n_j \bar{r}_j, \beta + n_j(1 - \bar{r}_j))$$

  and then choose arm $j^\star = \text{argmax}_{1 \leq j \leq K}\{p_j\}$
- this corresponds to selecting arm j with probability proportional to it's being the best arm in the posterior distribution
- arms for which we do not have accurate estimates for p_j will tend to be selected because their variance is high (exploration) – while arms with high p_j are also more likely to be selected (exploitation)

Bayesian analysis

- A possible Bayesian strategy is to sample each arm's response rate according to its posterior distribution

$$p_j \sim P(p; \alpha + n_j \bar{r}_j, \beta + n_j(1 - \bar{r}_j))$$

  and then choose arm $j^\star = \operatorname{argmax}_{1 \leq j \leq K}\{p_j\}$
- this corresponds to selecting arm j with probability proportional to it's being the best arm in the posterior distribution
- arms for which we do not have accurate estimates for p_j will tend to be selected because their variance is high (exploration) – while arms with high p_j are also more likely to be selected (exploitation)

Gittins Indices

- Gittins (1979) [3] proved that an optimal bandit policy for the discounted MAB can be given by finding a mapping γ from state s_j of each arm j to the reals in such a way that an arm that currently has largest $\gamma(s_j)$ is played
- The key to proving the result (see Tsitsiklis (1994) [6] for a short proof) is to show that we can remove an optimal state by adjusting the reward and transition probabilities and then apply induction
- The function γ is known as a *Gittins Index*
- Problem is computing the indices – making the Bayesian model means we have a finite number of states at any stage and so we can define the algorithm implied by the proof of optimality

Gittins Indices

- Gittins (1979) [3] proved that an optimal bandit policy for the discounted MAB can be given by finding a mapping γ from state s_j of each arm j to the reals in such a way that an arm that currently has largest $\gamma(s_j)$ is played
- The key to proving the result (see Tsitsiklis (1994) [6] for a short proof) is to show that we can remove an optimal state by adjusting the reward and transition probabilities and then apply induction
- The function γ is known as a *Gittins Index*
- Problem is computing the indices – making the Bayesian model means we have a finite number of states at any stage and so we can define the algorithm implied by the proof of optimality

Gittins Indices

- Gittins (1979) [3] proved that an optimal bandit policy for the discounted MAB can be given by finding a mapping γ from state s_j of each arm j to the reals in such a way that an arm that currently has largest $\gamma(s_j)$ is played
- The key to proving the result (see Tsitsiklis (1994) [6] for a short proof) is to show that we can remove an optimal state by adjusting the reward and transition probabilities and then apply induction
- The function γ is known as a *Gittins Index*
- Problem is computing the indices – making the Bayesian model means we have a finite number of states at any stage and so we can define the algorithm implied by the proof of optimality

Gittins Indices

- Gittins (1979) [3] proved that an optimal bandit policy for the discounted MAB can be given by finding a mapping γ from state s_j of each arm j to the reals in such a way that an arm that currently has largest $\gamma(s_j)$ is played
- The key to proving the result (see Tsitsiklis (1994) [6] for a short proof) is to show that we can remove an optimal state by adjusting the reward and transition probabilities and then apply induction
- The function γ is known as a *Gittins Index*
- Problem is computing the indices – making the Bayesian model means we have a finite number of states at any stage and so we can define the algorithm implied by the proof of optimality

Calculating Gittins Indices

- Furthermore the indices can be computed independently for each arm and are thus just a function of (α_j, β_j), the current values for arm j
- These are computed up to a notional (or real in the case of finite horizon) stopping point M that automatically bounds the size of the state space since $\alpha_j + \beta_j \leq M + \alpha_0 + \beta_0$
- We compute from the end of the period backwards using a recursive formula for the expected gain from the possible evolutions of the current state to the next time point
- By extending M we rapidly converge to a good approximation unless the discount factor is very close to 1

Calculating Gittins Indices

- Furthermore the indices can be computed independently for each arm and are thus just a function of (α_j, β_j), the current values for arm j
- These are computed up to a notional (or real in the case of finite horizon) stopping point M that automatically bounds the size of the state space since $\alpha_j + \beta_j \leq M + \alpha_0 + \beta_0$
- We compute from the end of the period backwards using a recursive formula for the expected gain from the possible evolutions of the current state to the next time point
- By extending M we rapidly converge to a good approximation unless the discount factor is very close to 1

Calculating Gittins Indices

- Furthermore the indices can be computed independently for each arm and are thus just a function of (α_j, β_j), the current values for arm j
- These are computed up to a notional (or real in the case of finite horizon) stopping point M that automatically bounds the size of the state space since $\alpha_j + \beta_j \leq M + \alpha_0 + \beta_0$
- We compute from the end of the period backwards using a recursive formula for the expected gain from the possible evolutions of the current state to the next time point
- By extending M we rapidly converge to a good approximation unless the discount factor is very close to 1

Calculating Gittins Indices

- Furthermore the indices can be computed independently for each arm and are thus just a function of (α_j, β_j), the current values for arm j
- These are computed up to a notional (or real in the case of finite horizon) stopping point M that automatically bounds the size of the state space since $\alpha_j + \beta_j \leq M + \alpha_0 + \beta_0$
- We compute from the end of the period backwards using a recursive formula for the expected gain from the possible evolutions of the current state to the next time point
- By extending M we rapidly converge to a good approximation unless the discount factor is very close to 1

Calculating Gittins Indices

- For the finite horizon and no discount the computation simplifies to give the initialisation:

$$\gamma_{M-1}(\alpha, \beta) = \frac{\alpha}{\alpha + \beta}$$

- and recursion

$$\gamma_{t-1}(\alpha, \beta) = \frac{\alpha}{\alpha + \beta}(1 + \gamma_t(\alpha + 1, \beta)) + \frac{\beta}{\alpha + \beta}\gamma_t(\alpha, \beta + 1)$$

- With $\gamma_t(\alpha, \beta)$ being computed for all pairs (α, β) such that $\alpha + \beta \leq t + \alpha_0 + \beta_0$

Calculating Gittins Indices

- For the finite horizon and no discount the computation simplifies to give the initialisation:

$$\gamma_{M-1}(\alpha, \beta) = \frac{\alpha}{\alpha + \beta}$$

- and recursion

$$\gamma_{t-1}(\alpha, \beta) = \frac{\alpha}{\alpha + \beta}(1 + \gamma_t(\alpha + 1, \beta)) + \frac{\beta}{\alpha + \beta}\gamma_t(\alpha, \beta + 1)$$

- With $\gamma_t(\alpha, \beta)$ being computed for all pairs (α, β) such that $\alpha + \beta \leq t + \alpha_0 + \beta_0$

Calculating Gittins Indices

- For the finite horizon and no discount the computation simplifies to give the initialisation:

$$\gamma_{M-1}(\alpha, \beta) = \frac{\alpha}{\alpha + \beta}$$

- and recursion

$$\gamma_{t-1}(\alpha, \beta) = \frac{\alpha}{\alpha + \beta}(1 + \gamma_t(\alpha + 1, \beta)) + \frac{\beta}{\alpha + \beta}\gamma_t(\alpha, \beta + 1)$$

- With $\gamma_t(\alpha, \beta)$ being computed for all pairs (α, β) such that $\alpha + \beta \leq t + \alpha_0 + \beta_0$

Example table of Gittins Indices

Table 1. *Values of* $\nu(\alpha, \beta, 0{\cdot}75)$

β	$\alpha = 1$	$\alpha = 2$	$\alpha = 3$	$\alpha = 4$	$\alpha = 5$	$\alpha = 6$	$\alpha = 7$	$\alpha = 8$	$\alpha = 9$	$\alpha = 10$
1	0·6211	0·7465	0·8062	0·8419	0·8659	0·8833	0·8965	0·9069	0·9153	0·9223
2	0·4256	0·5760	0·6607	0·7159	0·7548	0·7841	0·8068	0·8251	0·8401	0·8526
3	0·3182	0·4641	0·5554	0·6191	0·6659	0·7023	0·7312	0·7548	0·7745	0·7912
4	0·2519	0·3871	0·4773	0·5436	0·5946	0·6348	0·6673	0·6946	0·7176	0·7372
5	0·2073	0·3307	0·4182	0·4838	0·5360	0·5784	0·6134	0·6428	0·6678	0·6896
6	0·1755	0·2883	0·3713	0·4359	0·4875	0·5306	0·5669	0·5978	0·6244	0·6476
7	0·1518	0·2550	0·3334	0·3961	0·4473	0·4899	0·5266	0·5584	0·5860	0·6102
8	0·1335	0·2285	0·3025	0·3627	0·4129	0·4553	0·4916	0·5236	0·5518	0·5767
9	0·1190	0·2067	0·2767	0·3343	0·3832	0·4249	0·4611	0·4928	0·5212	0·5465
10	0·1072	0·1886	0·2547	0·3100	0·3573	0·3983	0·4341	0·4657	0·4937	0·5192

- (From Gittins and Jones (1979) [4] Note how for a fixed value of $\alpha/(\alpha+\beta)$ (say $= 0.5$) the index decreases with increasing α indicating how the index favours states with higher uncertainty but equal reward expectation

Upper Confidence Bound (UCB) Strategies

- One more efficient way of devising an index is to keep a current best estimate of the reward probability $\bar{r}_j$ for arm j and our associated uncertainty $\bar{\sigma}_j$ in this estimation
- We now use the 'index'

$$G(j) = \bar{r}_j + B(t)\bar{\sigma}_j,$$

where $B(t)$ is an increasing function of the time t
- The idea behind this policy is that we are likely to:
  - play arms that have received good reward rates as they will have high $\bar{r}_j$ (exploitation)
  - play under-explored arms since they will have high $\bar{\sigma}_j$ (exploration)

Upper Confidence Bound (UCB) Strategies

- One more efficient way of devising an index is to keep a current best estimate of the reward probability $\bar{r}_j$ for arm j and our associated uncertainty $\bar{\sigma}_j$ in this estimation
- We now use the 'index'

$$G(j) = \bar{r}_j + B(t)\bar{\sigma}_j,$$

where $B(t)$ is an increasing function of the time t

- The idea behind this policy is that we are likely to:
  - play arms that have received good reward rates as they will have high $\bar{r}_j$ (exploitation)
  - play under-explored arms since they will have high $\bar{\sigma}_j$ (exploration)

Upper Confidence Bound (UCB) Strategies

- One more efficient way of devising an index is to keep a current best estimate of the reward probability $\bar{r}_j$ for arm j and our associated uncertainty $\bar{\sigma}_j$ in this estimation
- We now use the 'index'

$$G(j) = \bar{r}_j + B(t)\bar{\sigma}_j,$$

  where $B(t)$ is an increasing function of the time t
- The idea behind this policy is that we are likely to:
  - play arms that have received good reward rates as they will have high $\bar{r}_j$ (exploitation)
  - play under-explored arms since they will have high $\bar{\sigma}_j$ (exploration)

Upper Confidence Bound (UCB) Strategies

- One more efficient way of devising an index is to keep a current best estimate of the reward probability $\bar{r}_j$ for arm j and our associated uncertainty $\bar{\sigma}_j$ in this estimation
- We now use the 'index'

$$G(j) = \bar{r}_j + B(t)\bar{\sigma}_j,$$

  where $B(t)$ is an increasing function of the time t
- The idea behind this policy is that we are likely to:
  - play arms that have received good reward rates as they will have high $\bar{r}_j$ (exploitation)
  - play under-explored arms since they will have high $\bar{\sigma}_j$ (exploration)

Upper Confidence Bound (UCB) Strategies

- One more efficient way of devising an index is to keep a current best estimate of the reward probability $\bar{r}_j$ for arm j and our associated uncertainty $\bar{\sigma}_j$ in this estimation
- We now use the 'index'

$$G(j) = \bar{r}_j + B(t)\bar{\sigma}_j,$$

  where $B(t)$ is an increasing function of the time t
- The idea behind this policy is that we are likely to:
  - play arms that have received good reward rates as they will have high $\bar{r}_j$ (exploitation)
  - play under-explored arms since they will have high $\bar{\sigma}_j$ (exploration)

Upper Confidence Strategies: analysis

- Auer, Cesa-Bianchi and Fischer [2] analysed UCB strategies for MABs with arbitrary reward distributions in $[0, 1]$
- Letting $\Delta_i = \mu^\star - \mu_i$ they showed that, if we choose $B(t) = \sqrt{\ln T}$ and $\bar{\sigma}_j = 1/\sqrt{n_j}$ where n_j is number of times arm j has been played, then the expected regret after T plays is at most

$$\ln T \left[\sum_{i:\mu_i<\mu^\star} \left(\frac{1}{\Delta_i} \right) \right] + \left(1 + \frac{\pi^2}{3} \right) \sum_{j=1}^{K} \Delta_j$$

Upper Confidence Strategies: analysis

- Auer, Cesa-Bianchi and Fischer [2] analysed UCB strategies for MABs with arbitrary reward distributions in $[0, 1]$
- Letting $\Delta_i = \mu^\star - \mu_i$ they showed that, if we choose $B(t) = \sqrt{\ln T}$ and $\bar{\sigma}_j = 1/\sqrt{n_j}$ where n_j is number of times arm j has been played, then the expected regret after T plays is at most

$$\ln T \left[\sum_{i:\mu_i < \mu^\star} \left(\frac{1}{\Delta_i} \right) \right] + \left(1 + \frac{\pi^2}{3} \right) \sum_{j=1}^{K} \Delta_j$$

Upper Confidence Strategies: practice

- In practical experiments it is usually better to use an empirical estimate of the variance
- This is known as UCB-tuned where we compute the variance as

$$\bar{\sigma}_j = \sqrt{\frac{\min\{0.25, V_j(n_j)\}}{n_j}} \quad \text{where}$$

$$V_j(s) = \frac{1}{s}\sum_{\tau=1}^{s} r_{j,\tau}^2 - \bar{r}_{j,s}^2 + \sqrt{\frac{2\ln t}{s}}$$

(t is the total number of plays when j is played for the sth time)

- Could also compute mean and variance from the Bayesian estimates

Upper Confidence Strategies: practice

- In practical experiments it is usually better to use an empirical estimate of the variance
- This is known as UCB-tuned where we compute the variance as

$$\bar{\sigma}_j = \sqrt{\frac{\min\{0.25, V_j(n_j)\}}{n_j}} \quad \text{where}$$

$$V_j(s) = \frac{1}{s}\sum_{\tau=1}^{s} r_{j,\tau}^2 - \bar{r}_{j,s}^2 + \sqrt{\frac{2\ln t}{s}}$$

(t is the total number of plays when j is played for the sth time)

- Could also compute mean and variance from the Bayesian estimates

Upper Confidence Strategies: practice

- In practical experiments it is usually better to use an empirical estimate of the variance
- This is known as UCB-tuned where we compute the variance as

$$\bar{\sigma}_j = \sqrt{\frac{\min\{0.25, V_j(n_j)\}}{n_j}} \quad \text{where}$$

$$V_j(s) = \frac{1}{s}\sum_{\tau=1}^{s} r_{j,\tau}^2 - \bar{r}_{j,s}^2 + \sqrt{\frac{2\ln t}{s}}$$

(t is the total number of plays when j is played for the sth time)

- Could also compute mean and variance from the Bayesian estimates

ϵ_t-Greedy

- As a comparison a relatively naive approach is the ϵ_t-Greedy algorithm
- We define the sequence

$$\epsilon_t = \min\left\{1, \frac{cK}{d^2 t}\right\}$$

where $c > 0$ and $0 < d < 1$ are parameters

- Now at iteration t with probability $1 - \epsilon_t$ play the machine with best current average reward, and otherwise play a random arm
- Note that the ϵ_t-greedy strategy does not do selective exploration and so will typically over-explore very weak arms

ϵ_t-Greedy

- As a comparison a relatively naive approach is the ϵ_t-Greedy algorithm
- We define the sequence

$$\epsilon_t = \min\left\{1, \frac{cK}{d^2 t}\right\}$$

  where $c > 0$ and $0 < d < 1$ are parameters
- Now at iteration t with probability $1 - \epsilon_t$ play the machine with best current average reward, and otherwise play a random arm
- Note that the ϵ_t-greedy strategy does not do selective exploration and so will typically over-explore very weak arms

ϵ_t-Greedy

- As a comparison a relatively naive approach is the ϵ_t-Greedy algorithm
- We define the sequence

$$\epsilon_t = \min\left\{1, \frac{cK}{d^2 t}\right\}$$

  where $c > 0$ and $0 < d < 1$ are parameters
- Now at iteration t with probability $1 - \epsilon_t$ play the machine with best current average reward, and otherwise play a random arm
- Note that the ϵ_t-greedy strategy does not do selective exploration and so will typically over-explore very weak arms

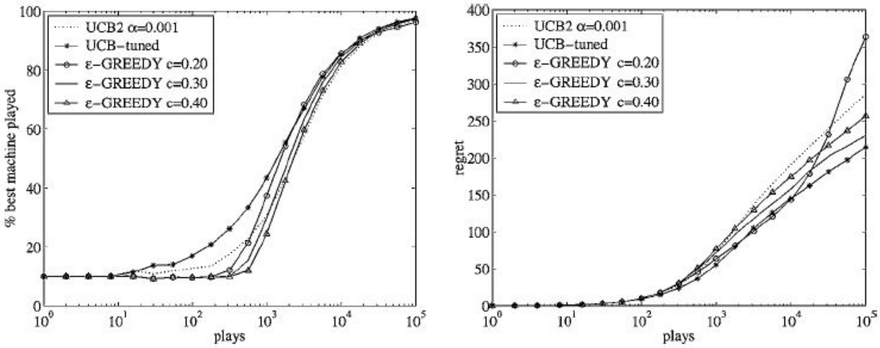
ϵ_t-Greedy

- As a comparison a relatively naive approach is the ϵ_t-Greedy algorithm
- We define the sequence

$$\epsilon_t = \min\left\{1, \frac{cK}{d^2 t}\right\}$$

where $c > 0$ and $0 < d < 1$ are parameters
- Now at iteration t with probability $1 - \epsilon_t$ play the machine with best current average reward, and otherwise play a random arm
- Note that the ϵ_t-greedy strategy does not do selective exploration and so will typically over-explore very weak arms

Some empirical comparisons



- % best arm played and cumulative regret with 10 arms, $p_1 = 0.55, p_i = 0.45, 2 \leq i \leq 10$

Sequences of actions

- The bandit approach can be extended to sequences of decisions
- One possibility is to form a tree where each sequence of actions traces a path from the root
- At each node we consider a standard bandit to decide which child to pick
- Gives rise to the UCT algorithm [5]

Sequences of actions

- The bandit approach can be extended to sequences of decisions
- One possibility is to form a tree where each sequence of actions traces a path from the root
- At each node we consider a standard bandit to decide which child to pick
- Gives rise to the UCT algorithm [5]

Sequences of actions

- The bandit approach can be extended to sequences of decisions
- One possibility is to form a tree where each sequence of actions traces a path from the root
- At each node we consider a standard bandit to decide which child to pick
- Gives rise to the UCT algorithm [5]

Sequences of actions

- The bandit approach can be extended to sequences of decisions
- One possibility is to form a tree where each sequence of actions traces a path from the root
- At each node we consider a standard bandit to decide which child to pick
- Gives rise to the UCT algorithm [5]

UCT

- Uses simple UCB style formula:

$$I_t = \operatorname{argmax}_{i \in \{1,\ldots,K\}} \left\{ \bar{X}_{i,T_i(t-1)} + c_{t-1,T_i(t-1)} \right\}$$

where

- $$c_{t,s} = \sqrt{\frac{2 \ln t}{s}}$$
- t is the number of the visit to the node,
- $T_i(t-1)$ is the number of times the ith child (action) has been selected and
- $\bar{X}_{i,T_i(t-1)}$ is the average (discounted) reward arising from this action

UCT

- Uses simple UCB style formula:

$$I_t = \operatorname{argmax}_{i \in \{1,\dots,K\}} \left\{ \bar{X}_{i,T_i(t-1)} + c_{t-1,T_i(t-1)} \right\}$$

where

- $$c_{t,s} = \sqrt{\frac{2 \ln t}{s}}$$
- t is the number of the visit to the node,
- $T_i(t-1)$ is the number of times the ith child (action) has been selected and
- $\bar{X}_{i,T_i(t-1)}$ is the average (discounted) reward arising from this action

UCT

- Uses simple UCB style formula:

$$I_t = \operatorname{argmax}_{i \in \{1,\ldots,K\}} \left\{ \bar{X}_{i,T_i(t-1)} + c_{t-1,T_i(t-1)} \right\}$$

where

- $$c_{t,s} = \sqrt{\frac{2 \ln t}{s}}$$
- t is the number of the visit to the node,
- $T_i(t-1)$ is the number of times the ith child (action) has been selected and
- $\bar{X}_{i,T_i(t-1)}$ is the average (discounted) reward arising from this action

UCT

- Uses simple UCB style formula:

$$I_t = \text{argmax}_{i \in \{1,\dots,K\}} \left\{ \bar{X}_{i,T_i(t-1)} + c_{t-1,T_i(t-1)} \right\}$$

where

- $$c_{t,s} = \sqrt{\frac{2 \ln t}{s}}$$
- t is the number of the visit to the node,
- $T_i(t-1)$ is the number of times the ith child (action) has been selected and
- $\bar{X}_{i,T_i(t-1)}$ is the average (discounted) reward arising from this action

UCT

- Uses simple UCB style formula:

$$I_t = \operatorname{argmax}_{i\in\{1,\ldots,K\}} \left\{ \bar{X}_{i,T_i(t-1)} + c_{t-1,T_i(t-1)} \right\}$$

where

- $$c_{t,s} = \sqrt{\frac{2\ln t}{s}}$$
- t is the number of the visit to the node,
- $T_i(t-1)$ is the number of times the ith child (action) has been selected and
- $\bar{X}_{i,T_i(t-1)}$ is the average (discounted) reward arising from this action

UCT

- For a domain where we can generate samples – i.e. have a generative model for the domain
- Example might be a game as we know the rules and can simulate games
- Classical game playing algorithms perform α-β search to prune the search tree
- Can be manageable for games with a reasonable branching factor (i.e. number of move choices) such as chess (with powerful computers)
- Becomes impractical if the branching factor is too large, e.g. Go

UCT

- For a domain where we can generate samples – i.e. have a generative model for the domain
- Example might be a game as we know the rules and can simulate games
- Classical game playing algorithms perform α-β search to prune the search tree
- Can be manageable for games with a reasonable branching factor (i.e. number of move choices) such as chess (with powerful computers)
- Becomes impractical if the branching factor is too large, e.g. Go

UCT

- For a domain where we can generate samples – i.e. have a generative model for the domain
- Example might be a game as we know the rules and can simulate games
- Classical game playing algorithms perform α-β search to prune the search tree
- Can be manageable for games with a reasonable branching factor (i.e. number of move choices) such as chess (with powerful computers)
- Becomes impractical if the branching factor is too large, e.g. Go

UCT

- For a domain where we can generate samples – i.e. have a generative model for the domain
- Example might be a game as we know the rules and can simulate games
- Classical game playing algorithms perform α-β search to prune the search tree
- Can be manageable for games with a reasonable branching factor (i.e. number of move choices) such as chess (with powerful computers)
- Becomes impractical if the branching factor is too large, e.g. Go

UCT

- For a domain where we can generate samples – i.e. have a generative model for the domain
- Example might be a game as we know the rules and can simulate games
- Classical game playing algorithms perform α-β search to prune the search tree
- Can be manageable for games with a reasonable branching factor (i.e. number of move choices) such as chess (with powerful computers)
- Becomes impractical if the branching factor is too large, e.g. Go

UCT

- UCT replaces the exhaustive α-β search with an exploration exploitation strategy
- By keeping track of the quality of different branches of the tree those that are promising are explored more than others
- This will only be fruitful if strategy revisits nodes – so that statistics are accumulated
- Otherwise reduces to monte-carlo exploration – a popular strategy for exploring MDPs
- Will be more on this later in the course

UCT

- UCT replaces the exhaustive α-β search with an exploration exploitation strategy
- By keeping track of the quality of different branches of the tree those that are promising are explored more than others
- This will only be fruitful if strategy revisits nodes – so that statistics are accumulated
- Otherwise reduces to monte-carlo exploration – a popular strategy for exploring MDPs
- Will be more on this later in the course

UCT

- UCT replaces the exhaustive α-β search with an exploration exploitation strategy
- By keeping track of the quality of different branches of the tree those that are promising are explored more than others
- This will only be fruitful if strategy revisits nodes – so that statistics are accumulated
- Otherwise reduces to monte-carlo exploration – a popular strategy for exploring MDPs
- Will be more on this later in the course

UCT

- UCT replaces the exhaustive α-β search with an exploration exploitation strategy
- By keeping track of the quality of different branches of the tree those that are promising are explored more than others
- This will only be fruitful if strategy revisits nodes – so that statistics are accumulated
- Otherwise reduces to monte-carlo exploration – a popular strategy for exploring MDPs
- Will be more on this later in the course

UCT

- UCT replaces the exhaustive α-β search with an exploration exploitation strategy
- By keeping track of the quality of different branches of the tree those that are promising are explored more than others
- This will only be fruitful if strategy revisits nodes – so that statistics are accumulated
- Otherwise reduces to monte-carlo exploration – a popular strategy for exploring MDPs
- Will be more on this later in the course

Multivariate or Associative Bandits

- More realistic scenario is when the response rate of the different options is a function of some side information:

$$p_i = f(\mathbf{x}_i)$$

  where we assume that at each play we are given a vector $\mathbf{x}_i$ of features for each arm i
- For example, this could correspond to a user visiting a website when a decision must be taken about which banner content to display
- Each choice of content corresponds to an arm
- Note that the feature vector could include information about the user and the arm

Multivariate or Associative Bandits

- More realistic scenario is when the response rate of the different options is a function of some side information:

$$p_i = f(\mathbf{x}_i)$$

  where we assume that at each play we are given a vector $\mathbf{x}_i$ of features for each arm i
- For example, this could correspond to a user visiting a website when a decision must be taken about which banner content to display
- Each choice of content corresponds to an arm
- Note that the feature vector could include information about the user and the arm

Multivariate or Associative Bandits

- More realistic scenario is when the response rate of the different options is a function of some side information:

$$p_i = f(\mathbf{x}_i)$$

  where we assume that at each play we are given a vector $\mathbf{x}_i$ of features for each arm i
- For example, this could correspond to a user visiting a website when a decision must be taken about which banner content to display
- Each choice of content corresponds to an arm
- Note that the feature vector could include information about the user and the arm

Multivariate or Associative Bandits

- More realistic scenario is when the response rate of the different options is a function of some side information:

$$p_i = f(\mathbf{x}_i)$$

  where we assume that at each play we are given a vector $\mathbf{x}_i$ of features for each arm i
- For example, this could correspond to a user visiting a website when a decision must be taken about which banner content to display
- Each choice of content corresponds to an arm
- Note that the feature vector could include information about the user and the arm

Linear Associative Bandits

- If there is no special information for the arm we can simply form a vector by taking

$$\mathbf{x}_i = \mathbf{e}_i \otimes \mathbf{x}$$

  where $\mathbf{e}_i$ is the ith unit vector and $\mathbf{x}$ contains the features of the user
- A natural simple model is to assume a linear model

$$p_i = \langle \mathbf{w}, \mathbf{x}_i \rangle$$

- Gives rise to the LINREL algorithm of Auer [1]

Linear Associative Bandits

- If there is no special information for the arm we can simply form a vector by taking

$$\mathbf{x}_i = \mathbf{e}_i \otimes \mathbf{x}$$

where $\mathbf{e}_i$ is the ith unit vector and $\mathbf{x}$ contains the features of the user
- A natural simple model is to assume a linear model

$$p_i = \langle \mathbf{w}, \mathbf{x}_i \rangle$$

- Gives rise to the LINREL algorithm of Auer [1]

Linear Associative Bandits

- If there is no special information for the arm we can simply form a vector by taking

$$\mathbf{x}_i = \mathbf{e}_i \otimes \mathbf{x}$$

  where $\mathbf{e}_i$ is the ith unit vector and $\mathbf{x}$ contains the features of the user
- A natural simple model is to assume a linear model

$$p_i = \langle \mathbf{w}, \mathbf{x}_i \rangle$$

- Gives rise to the LINREL algorithm of Auer [1]

LINREL

- The strategy employed is similar to UCB – i.e. estimate expected return and variance
- Key to analysis is expressing each new feature vector $\mathbf{x}_i^T$ at stage T in terms of those already chosen in earlier stages:

$$\mathbf{x}_i^T \approx \sum_{t=1}^{T-1} \alpha_t^i \mathbf{x}_{j_t}^t$$

where j_t is the arm chosen at time t with feature vector $\mathbf{x}_{j_t}^t$

LINREL

- The strategy employed is similar to UCB – i.e. estimate expected return and variance
- Key to analysis is expressing each new feature vector $\mathbf{x}_i^T$ at stage T in terms of those already chosen in earlier stages:

$$\mathbf{x}_i^T \approx \sum_{t=1}^{T-1} \alpha_t^i \mathbf{x}_{j_t}^t$$

where j_t is the arm chosen at time t with feature vector $\mathbf{x}_{j_t}^t$

LINREL

- We can now estimate the expected return as

$$
\begin{aligned}
p_i &= \left\langle \mathbf{w}, \mathbf{x}_i^T \right\rangle \\
&= \sum_{t=1}^{T-1} \alpha_t^i \left\langle \mathbf{w}, \mathbf{x}_{j_t}^t \right\rangle \\
&= \sum_{t=1}^{T-1} \alpha_t^i r_t
\end{aligned}
$$

where r_t is the reward received on step t

- Gives rise to the LINREL algorithm

LINREL

- We can now estimate the expected return as

$$p_i = \left\langle \mathbf{w}, \mathbf{x}_i^T \right\rangle = \sum_{t=1}^{T-1} \alpha_t^i \left\langle \mathbf{w}, \mathbf{x}_{j_t}^t \right\rangle = \sum_{t=1}^{T-1} \alpha_t^i r_t$$

  where r_t is the reward received on step t
- Gives rise to the LINREL algorithm

LINREL

- At stage t solve

$$\left(Z(t)'Z(t) + \lambda I\right) \alpha^i = Z(t)'\mathbf{x}_i^t$$

  where $Z(t)$ is a matrix formed with columns the feature vectors of the selected arms

- Define

$$\begin{aligned} \mathrm{width}_i(t) &= \|\alpha^i\|\sqrt{\ln(2TK/\delta)} \\ \mathrm{ucb}_i(t) &= \left\langle \mathbf{r}^t, \alpha^i \right\rangle + \mathrm{width}_i(t) \end{aligned}$$

- Play the arm with largest $\mathrm{ucb}_i(t)$

LINREL

- At stage t solve

$$\left(Z(t)'Z(t) + \lambda I\right) \alpha^i = Z(t)'\mathbf{x}_i^t$$

where $Z(t)$ is a matrix formed with columns the feature vectors of the selected arms

- Define

$$\begin{aligned} \mathrm{width}_i(t) &= \|\alpha^i\| \sqrt{\ln(2TK/\delta)} \\ \mathrm{ucb}_i(t) &= \left\langle \mathbf{r}^t, \alpha^i \right\rangle + \mathrm{width}_i(t) \end{aligned}$$

- Play the arm with largest $\mathrm{ucb}_i(t)$

LINREL

- At stage t solve

$$\left(Z(t)'Z(t) + \lambda I\right)\alpha^i = Z(t)'\mathbf{x}_i^t$$

  where $Z(t)$ is a matrix formed with columns the feature vectors of the selected arms
- Define

$$\begin{aligned} \mathrm{width}_i(t) &= \|\alpha^i\|\sqrt{\ln(2TK/\delta)} \\ \mathrm{ucb}_i(t) &= \left\langle \mathbf{r}^t, \alpha^i \right\rangle + \mathrm{width}_i(t) \end{aligned}$$

- Play the arm with largest $\mathrm{ucb}_i(t)$

LINREL analysis

- For a variant of the LINREL algorithm there is a regret bound of the form

$$R(T) = O\left(\sqrt{TK \ln(T^3 K)}\right)$$

where K is the number of arms

- This is significantly weaker than the $\ln(T)$ bound for multi-armed bandits

LINREL analysis

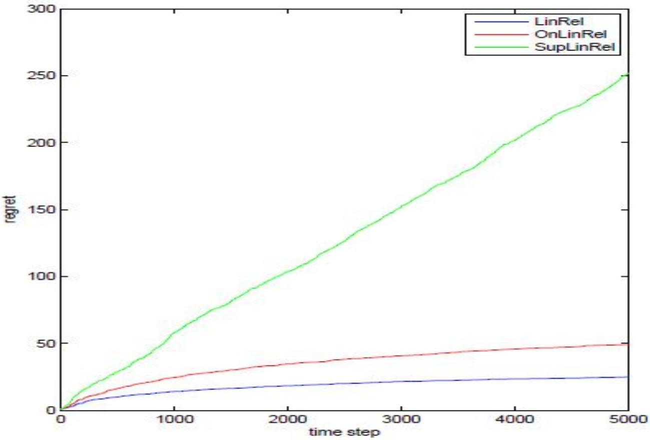
- For a variant of the LINREL algorithm there is a regret bound of the form

$$R(T) = O\left(\sqrt{TK \ln(T^3 K}\right)$$

where K is the number of arms

- This is significantly weaker than the $\ln(T)$ bound for multi-armed bandits

Some empirical comparisons



- Regret as a function of time steps for LinRel, OnLinRel and SupLinRel

Kernel LinRel

- Of course we can use the kernel trick indeed if we take the formula

$$\left(Z(t)'Z(t) + \lambda I\right) \alpha^i = Z(t)'\mathbf{x}_i^t$$

for LinRel this can be given in a kernel defined feature space as

$$\left(K(t) + \lambda I\right) \alpha^i = \mathbf{k}_i^t$$

where $K(t)$ is the kernel matrix of the feature vectors of the selected arms and $\mathbf{k}_i^t$ is the vector of kernel evaluations between the feature vector for the ith arm at time t and each of the selected arms up to time t

Speeding evaluation

- Frequently the response time for taking a decision is critical so that solving

$$\left(Z(t)'Z(t) + \lambda I\right)\alpha^i = Z(t)'\mathbf{x}_i^t$$

may be prohibitive

- We can expedite this by precomputing a cholesky decomposition of

$$R'R = \left(Z(t)'Z(t) + \lambda I\right)$$

so that to make a decision we only need to solve the much simpler upper and lower diagonal equations

$$R'\mathbf{z} = Z(t)'\mathbf{x}_i^t \quad \text{and} \quad R\alpha^i = \mathbf{z}$$

Speeding evaluation

- Frequently the response time for taking a decision is critical so that solving

$$\left(Z(t)'Z(t) + \lambda I\right)\alpha^i = Z(t)'\mathbf{x}_i^t$$

may be prohibitive
- We can expedite this by precomputing a cholesky decomposition of

$$R'R = \left(Z(t)'Z(t) + \lambda I\right)$$

so that to make a decision we only need to solve the much simpler upper and lower diagonal equations

$$R'\mathbf{z} = Z(t)'\mathbf{x}_i^t \quad \text{and} \quad R\alpha^i = \mathbf{z}$$

Controlling the size of $Z(t)$

- The other problem with the equation

$$\left(Z(t)'Z(t) + \lambda I\right)\alpha^i = Z(t)'\mathbf{x}_i^t$$

is that it grows in size with t

- Online algorithms can be used to decide which points to retain hence ensuring a manageable size of equation

Controlling the size of $Z(t)$

- The other problem with the equation

$$\left(Z(t)'Z(t) + \lambda I\right)\alpha^i = Z(t)'\mathbf{x}_i^t$$

  is that it grows in size with t
- Online algorithms can be used to decide which points to retain hence ensuring a manageable size of equation

Conclusions

- Multi-armed bandit perhaps the simplest problem in which learning feedback is only partial
- We must decide which arm to play and only receive outcome for this arm
- Hence, involves both exploration and exploitation
- Surprisingly rich theory has evolved including Gittins indices and upper confidence bound algorithms
- Upper confidence approach has been extended to trees (UCT) and Gaussian process bandits (GPB)

Conclusions

- Multi-armed bandit perhaps the simplest problem in which learning feedback is only partial
- We must decide which arm to play and only receive outcome for this arm
- Hence, involves both exploration and exploitation
- Surprisingly rich theory has evolved including Gittins indices and upper confidence bound algorithms
- Upper confidence approach has been extended to trees (UCT) and Gaussian process bandits (GPB)

Conclusions

- Multi-armed bandit perhaps the simplest problem in which learning feedback is only partial
- We must decide which arm to play and only receive outcome for this arm
- Hence, involves both exploration and exploitation
- Surprisingly rich theory has evolved including Gittins indices and upper confidence bound algorithms
- Upper confidence approach has been extended to trees (UCT) and Gaussian process bandits (GPB)

Conclusions

- Multi-armed bandit perhaps the simplest problem in which learning feedback is only partial
- We must decide which arm to play and only receive outcome for this arm
- Hence, involves both exploration and exploitation
- Surprisingly rich theory has evolved including Gittins indices and upper confidence bound algorithms
- Upper confidence approach has been extended to trees (UCT) and Gaussian process bandits (GPB)

Conclusions

- Multi-armed bandit perhaps the simplest problem in which learning feedback is only partial
- We must decide which arm to play and only receive outcome for this arm
- Hence, involves both exploration and exploitation
- Surprisingly rich theory has evolved including Gittins indices and upper confidence bound algorithms
- Upper confidence approach has been extended to trees (UCT) and Gaussian process bandits (GPB)

References

P. Auer.
Using confidence bounds for exploitation-exploration trade-offs.
Journal of Machine Learning Research, 3:397–422, 2002.

P. Auer, N. Cesa-Bianchi, and P. Fischer.
Finite-time analysis of the multiarmed bandit problem.
Machine Learning, 47:235–256, 2002.

J. C. Gittins.
Bandit processes and dynamic allocation indices.
Journal of the Royal Statistical Society. Series B (Methodological), 41(2):148, 1979.

J.C. Gittins and D.M. Jones.
A dynamic allocation index for the discounted multiarmed bandit problem.
Biometrita, 66(3):561–6, 1979.

L. Kocsis and Cs. Szepesvári.
Bandit based monte-carlo planning.
In *Proceedings of the 17th European Conference on Machine Learning*, volume LNCS/LNAI 4212, pages 282–293. Springer-Verlag, 2006.

J.N. Tsitsiklis.
A short proof of the Gittins index theorem.
The Annals of Applied Probability, 4(1):194–199, 1994.