Interaction of Game Theory and Machine Learning in a Connected World.

The number of people interacting through internet has been increasing exponentially over the past 10 years and is expected to increase in a similar proportion in the coming years. This has made the world strongly connected with people networking through the internet. This has therefore also seen a rise in the way markets function in the traditional sense with markets, commerce and auction mechanisms coming online, which has further lead to the development of various fields of research which lie at the interface of computer science, game theory and economics theory. An exciting area of research, Algorithmic Game Theory has emerged and is being widely studied for optimizing the solution concepts in game theory and further analyzing the scope of computer science in economics and vice versa along with the applications of game theory in Artificial Intelligence, networking and operations research.
On a parallel front, the advancement of the field of machine learning has made it now possible for us learn from the data and make powerful predictions and analysis based on the inferences drawn from the data. Applications of machine learning in algorithmic game theory have been less studied and with the growing capabilities of machine learning in large decentralized systems it is drawing widespread attention and exciting contributions from computer scientists.
The purpose of this project is three fold:

1. Description and analysis of basic as well as advanced solution concepts of game theory.
2. Theoretical explanation and practical implementation of machine learning algorithms on open source datasets.
3. Interaction of the two fields and their application in state of the art research.

Game Theory

Game theory is defined as the mathematical study of interaction of self interested rational agents. This branch of study models every practical situation in the form of a game with several players described as agents playing the game. The inherent assumption is that the players interact in a manner by playing every move which is in their self interest. The agents have their own description of the world and act based on that description.
The dominant approach to modeling an agent's interest is *utility theory*. This theory quantifies the preferred action by the agent in situations involving a set of alternatives with the agent choosing the alternative with the most favorable outcome state or the state in which it has the maximum utility. This mapping from an existing state to the

final state with the agent's decision taken as an input is done through the *utility function.* When the agent is uncertain about which state of world he faces, his utility is defined as the expected value of his utility function with respect to the appropriate probability distribution over states.

Defining Games

Every game is defined by three parameters namely:

- **Players**: The agents playing the game responsible for making the decisions.
- **Actions**: The set of possible moves the player can play.
- **Payoffs**: The quantified value which will be offered to the player after taking the decided action.

These three parameters can also be thought of answering the basic questions of *Who*, *What* and *Why* with the Player information describing the Who, Action information describing the What and the Payoffs describing Why the Player is motivated to take that specific action.

Standard Representation of Games

Games are described in two forms, Normal Form (also know as Matrix form and Strategic Form) and Extensive Form.
Normal Form game definition lists what players get as a function of their action whilst incorporating the assumptions that the players move simultaneously and their strategies encode various things.
The Extensive form games on the other hand incorporate a temporal structure of the game and are more closer to the practical implementation of the games. The players in this definition move sequentially and are represented as a tree. Every player in this form of representation keeps track of what each player knows while making her decision.

Definition 1 (Normal-form game)[1]. A (finite, n-person) normal-form game is a tuple (N,A,u), where:

- $N = \{1,...,n\}$ is a finite set of n player, indexed by i;
- $A = A_i$ x … x A_n where A_i is a *finite set* of actions *available to player i.* Each vector $a = (a_i, \ldots, a_n) \in A$ is *called* an action profile;
- $u = (u_1, \ldots, u_n)$ where $u_i : A \rightarrow R$ is a real-valued utility (or payoff) function for player *i*.

A natural way to represent games is via an n-dimensional matrix. The normal form representation represents each 2-player game in the form of a matrix with the row player describing player 1 and the column player describing player 2. The rows correspond to the action profile of player 1 and the columns correspond to the action profile of player 2. The cells list the utility or payoff values for each player. The illustration of the normal form games will be specified through the fundamental example of Prisoner's Dilemma.

Prisoner's Dilemma
This is perhaps the most well-known and well-studied game.

Example 1 (Prisoner's Dilemma)[2] Two prisoner's are on a trial for a crime and each one faces a choice of confessing to the crime or remaining silent. If they both remain silent, the authorities will not be able to prove charges against them and they will both serve a short prison term, say 2 years, for minor offenses. If only one of them confesses, his term will be reduced to 1 year and he will be used as a witness against the other, who in turn will get a sentence of 5 years. Finally if they both confess, they will both get a small break for cooperating with the authorities and will have to serve prison sentences of 4 years each.
Hence there are four total outcomes depending on the choices made by each of the two prisoner's. The costs incurred in the four outcomes can be succinctly summarized in the following 2 x 2 matrix.

Player2 / Player1	Confess (Cooperate)	Silent (Defect)
Confess (Cooperate)	(4,4)	(1,5)
Silent (Defect)	(5,1)	(2,2)

In it's most general form, the Prisoner's Dilemma is any normal-form game with payoffs a, b, c and d corresponding 4, 1 , 5 and 2 respectively in a manner such that c > a > d > b. This famous game is often thought paradoxical because it has counter intuitive paradoxical properties in terms of the actions taken by the players and the solution set of the game which will be described in the next section.

Strategies in Normal-Form Games.

Every player has a set of actions available and he could choose a single action and play it. This is called a *pure strategy*. However he could also randomize over his set of available strategies and choose a certain strategy according to a probability distribution. This is known as *mixed strategy*.
The mixed strategy for a normal form game is defined as follows:

Definition 2 (Mixed Strategy). Let *(N, A, u)* be a normal-form game and for any set X let $\prod(X)$ be a set of probability distributions over X. Then the set of mixed strategies for player *i* is $S_i = \prod(A_i)$.

Definition 3 (Mixed-strategy profile). The set of mixed strategy profiles is the cartesian product of individual mixed-strategy sets, $S_1 \times \ldots \times S_n$.
Why play a mixed strategy:
- To randomize and confuse the opponent.
- Randomize when an agent is uncertain about other's actions.
- Mixed strategies are a concise description of repeated plays.
- Mixed strategies define population dynamics.

Definition 4 (Support). The support of a mixed strategy s_i for a player *i* is the set of pure strategies $\{a_i \,|\, s_i(a_i) > 0\}$.

Definition 5 (Expected utility of a mixed strategy). Given a normal-form game *(N, A, u),* the expected utility u_i for a player *i* of the mixed-strategy profile $s = (s_1, \ldots, s_n)$ is defined as

$$u_i(s) = \sum_{a \in A} u_i(a) \prod_{j=1}^{n} s_j(a_j)\,.$$

This is intuitively determined as the probability of reaching each outcome and then we calculate the average of the payoffs of each outcome.

Solution Concepts in Game Theory.

The environment with the interaction of several agents becomes very complex to analyze without a specific analysis technique. Therefore the tow most popular solution concepts namely Nash Equilibrium and Pareto Optimality help game theorists analyze the game and the outcome of the game based on the specific moves by the involved players.

Best Response and Nash Equilibrium.

Nash Equilibrium is one of the most standard solution concept in game theory. It requires the complete information of the game along with the action set profile of all the participating agents. The complete information of the game with the single player allows the player to choose the action with the maximum utility given the actions of the other agents. This allows the self interested agent to respond in a manner which is in his best interest. Therefore if every agent were to act in a manner such that her action were the best response to the actions of all other participating agents, we would arrive that solution which would be optimal for every participating agent.
The mathematical description defines $s_{-i} = (s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n)$, a strategy profile without agent *i's* strategy. Hence the complete strategy profile is described as

$s = (s_{i-1}, s_i)$. The decision problem with player *i* would be to decide his action from the strategy profile in order to maximize her utility given s_{-i}.

Definition 6 (Best Response). Player *i's* best response to strategy profile s_{-i} is a mixed strategy $s_i* \in S_i$ such that $u_i(s_i*, s_{-i}) \geq u_i(s_i, s_{-i})$ for all strategies $s_i \in S_i$.

It can also be thought as the action awe would take to maximize our utility in the light of the actions of others. The best response for an agent is not necessarily unique. However, any particular will not know what other strategies the other player will adopt and it is therefore not a solution concept since it does not help us identify an interesting set of outcomes in a general case. The solution concept however based on this strategy is the Nash Equilibrium.

Definition 7 (Nash Equilibrium). A strategy profile $s = (s_1, \ldots, s_n)$ is a Nash Equilibrium if, for all agents, *i*, s_i is the best response to s_{-i}.
Nash equilibrium is a self-consistent or stable set of profile because no agent has any incentive to deviate from her current strategy. If this were not true, the agents would not be in equilibrium since at least one player would have the incentive to deviate from her current strategy.

Domination
Let s_i and s'_i be two strategies for player *i* and *l*et S_{-i} be the set all possible strategy profiles for the other players.
Definition 8 (Strict Dominance). s_i strictly dominates s'_i if

$$\forall s_{-i} \in S_{-i}, \mathrm{u}_i(s_i, s_{-i}) > u_i(s'_i, s_{-i}).$$

Definition 9 (Weak Dominance). s_i weakly dominates s'_i if

$$\forall s_{-i} \in S_{-i}, \mathrm{u}_i(s_i, s_{-i}) \geq u_i(s'_i, s_{-i}).$$

Equilibria and Dominance
- If one strategy dominates every other strategy, it is said to be dominant.
- A strategy profile consisting of dominant strategy for every player is a Nash equilibrium.
- An equilibrium with strictly dominant strategies is unique.
- In Prisoner's Dilemma the dominant strategy is to defect.
- Intuitively, weak Nash equilibria are less stable than strict Nash equilibria because at least one player has a best response to other player's strategies that is not his equilibrium strategy.
- Mixed-strategy Nash equilibria are necessarily weak, while pure-strategy Nash equilibria may strict or weak, depending on the game.

Theorem 1 (John F. Nash, 1951)[7]. Every game with a finite number of players and finite number of action profiles has at least one Nash equilibrium.

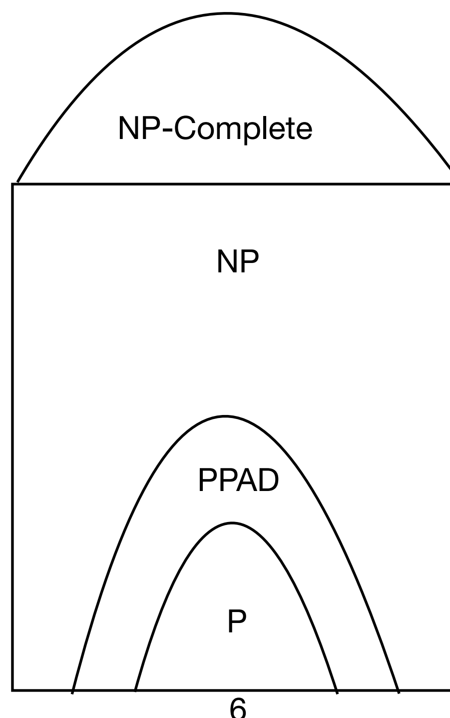
This theorem is motivating for the existence of either a pure strategy Nash equilibrium or a mixed strategy Nash equilibrium for every game. The theorem is proved using the Kakutani fixed-point theorem and is alternatively proved using the Brouwer fixed-point theorem[3].

Complexity of Computing a Nash Equilibrium.

Nash's theorem, proved in 1951, that every game has a Nash equilibrium. This is a reassuring fact which states that every game, in principle, can reach a quiescent state in which no player has an incentive to deviate from her behavior. However there has been a lot research in the field of finding the efficient algorithm to find the equilibrium which is guaranteed to exist.
The two algorithms for finding the Nash equilibrium are Linear Complementarity formulation or the Lemke-Howson algorithm[5] and Support Enumeration method[6]. However these algorithms are exponential in the worst case and finding a single Nash equilibrium seems hard. The problem of computing the Nash equilibrium is a fundamental problem of algorithmic game theory and it has been proved to be PPAD complete. In 1991, the complexity class PPAD, for which the Brouwer's problem is complete, was introduced[8].

The Class PPAD

In any directed graph with one unbalanced node (node with out degree different from its in degree), there must be another unbalanced node. The corresponding class is called *PPAD* for *"polynomial parity arguments for directed graphs",* and it contains Nash and Brouwer. Brouwer was proved to be PPAD-Complete in [8]. Unfortunately, Nash - the problem which had motivated this line of research was not shown PPAD-Complete, though it was conjectured to be.



It is established that computing an approximate Nash equilibrium in an r-player game is in PPAD. The r = 2 case was shown in [8].

Theorem 2 [4]**.** r-Nash is in PPAD, for r $\geq$ 2.

Alternate Solution Concepts in Normal-Form Games.

Beyond Nash Equilibrium
Nash equilibrium is just a way to analyze a game and to reason which strategy a player would choose given the strategy of other players.

Strictly Dominated Strategies and Iterated Removal.

- A strictly dominated strategy is never the best reply.
- It is removed since it will not be played.
- All players know this and hence it is iterated.
- Running the process to termination is called iterated removal of strictly dominated strategies.

A strategy $a_i \in A_i$ is strictly dominated by $a_i' \in A_i$ if $u_i(a_i, a_{-i}) < u_i(a_i', a_{-i}) \forall a_{-i} \in A_i$
The iterated removal of strictly dominant strategies gives us a lot of power by eliminating many of the payoffs and hence making our normal form game representation more succinct.

- Process preserves Nash equilibrium.
- It can be used a preprocessing step before computing an equilibrium.
- Some games are solvable using this technique. These games are dominance solvable.
- The order of removal does not matter with strictly dominated strategies.

Weakly Dominated Strategies
A strategy $a_i \in A_i$ is weakly dominated by $a_i' \in A_i$ if $u_i(a_i, a_{-i}) \leq u_i(a_i', a_{-i}) \forall a_{-i} \in A_i$ and $u_i(a_i, a_{-i}) < u_i(a_i', a_{-i})$ for some $a_{-i} \in A_i$.
a_i' does always at least as well and sometimes strictly better than a_i.

- They may or may not be the best replies.
- Order of removal can matter.
- At least one equilibrium is preserved.
- Nash equilibria is a subset of what preserves.

MAXMIN and MINMAX Strategies.

MAXMIN Strategy.
Player *i's* maxmin strategy is a strategy that maximizes *i's* worst-case payoff, in the situation where all other players *(-i)* happen to play the strategies which cause the

greatest harm to *i.* The maxmin value (or safety level) of the game for player *i* is that minimum payoff guaranteed by a maxmin strategy.

Definition 10 (Maxmin). The maxmin strategy for player *i* is:

$$\arg\max_{s_i} \min_{s_{-i}} u_i(s_i, s_{-i}),$$

and the maxmin value for player *i* is:

$$\max_{s_i} \min_{s_{-i}} u_i(s_i, s_{-i}),$$

MINMAX Strategy

Definition 11 (Minmax). Player i's minmax strategy against player *-i* in a 2-player game is a strategy that minimizes *-i's* best-case payoff and the minmax value for player *i* against player *-i* is payoff $\arg\min_{s_i} \max_{s_{-i}} u_{-i}(s_i, s_{-i})$,

Theorem 3 (Minimax theorem (John von Neumann, 1928))[9]. In any finite, two-player, zero-sum game, in any Nash equilibrium, each player receives a payoff that is equal to both his maxmin value and his minmax value.
This theorem concludes the following for two-player zero-sum games:

1. Each player's maxmin value is equal to his minmax value. By convention, the maxmin value of player 1 is called the value of the game.
2. For both players, the set of maxmin strategies coincides with set of minmax strategies.
3. Any maxmin strategy profile is a Nash equilibrium. Hence all Nash equilibria have the same payoff vector.

Perfect Information Extensive Form Games

Normal form game does not incorporate any notion of sequence or time in the actions of players. The extensive form is an alternate representation of a game that makes the temporal structure explicit. The variants of of extensive form games are:

- Perfect Information extensive form.
- Imperfect information extensive form.

Definition 12 (Perfect-information game). A (finite) perfect-information game (in extensive form) is defined by the tuple *(N, A, H, Z,* χ, ρ, σ, μ*) where:*

- *N* is a set of n players;
- A is a (single) set of actions;
- H is a set of non-terminal choice nodes;
- Z is a set of terminal choice nodes;
- $\chi : H \rightarrow 2^A$ is the action function, which assigns to each choice node a set of possible actions;
- $\rho : H \rightarrow N$ is the player function, which assigns to each nonterminal node a player $i \in N$ who chooses an action at that node;

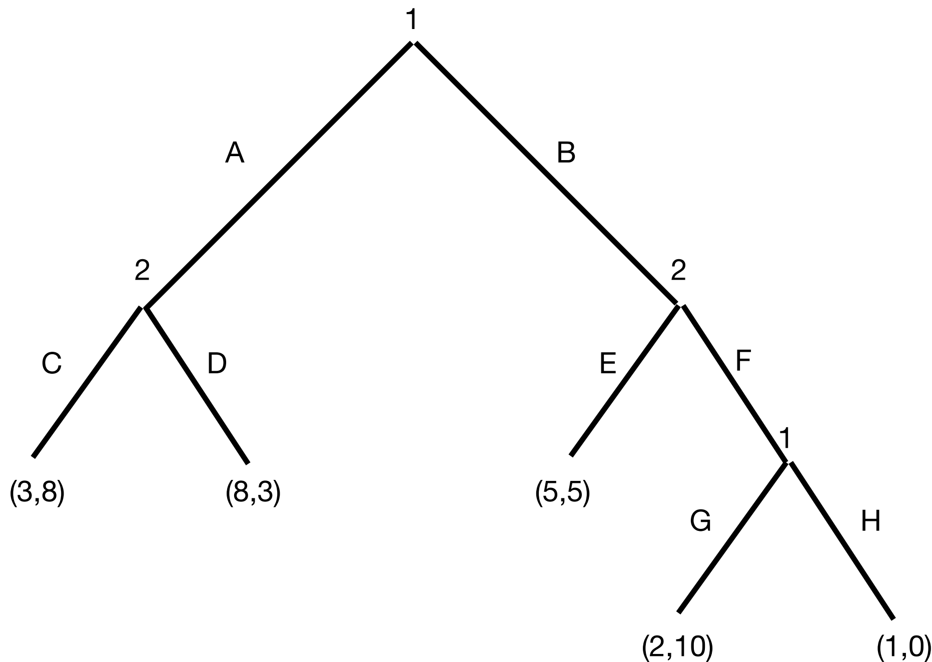
- $\sigma : H \times A \rightarrow H \cup Z$ is the successor function, which maps a choice node and an action to a new choice node or terminal node such that for all $h_1, h_2 \in H$ and $a_1, a_2 \in A$, if $\sigma(h_1, a_1) = \sigma(h_2, a_2)$ then $h_1 = h_2$ and $a_1 = a_2$;
- $u = (u_1, \ldots, u_n)$, where $u_i : Z \rightarrow \mathfrak{R}$ is a real-valued utility function for a player i on the terminal nodes Z.

Strategies and Equilibria
A pure strategy for a player in a perfect information game is a complete specification of which action to take at each node belonging to that player.

Definition 13 (Pure Strategies). Let *G = (N, A, H Z, χ, ρ, σ, μ)* be a perfect-information extensive-form game. Then the pure strategies for player *i* consist of the cartesian product $\prod_{h \in H, \rho(h)=i} \chi(h)$.

Nash Equilibrium
Given the new definition of pure strategy, the previous definitions of mixed strategies, best responses and Nash equilibrium can be reused.

Induced Normal-Form
Every extensive form game can be converted into normal form however the converse is not true.



(FIGURE): *Representation of a perfect information extensive form game.*

Theorem 4. Every (finite) perfect-information game in extensive form has a pure-strategy Nash equilibrium.
This result is due to Zermelo, 1913 with the intuition that since players take turns and everyone gets to see everything that happened thus far before making a move, it is never necessary to introduce randomness into action selection in order to find an equilibrium.

Subgame-Perfect Equilibrium

In order to make the Nash equilibrium more intuitively correct in the extensive form games, the concept of subgame perfection has been introduced.

Definition 14 (Subgame). Given a perfect-information extensive-form game G, the subgame of G rooted at node h is the restriction of G to the descendants of h. The set of subgames of G consists of all the subgames of G rooted at some node in G.

Definition 15 (Subgame-perfect equilibrium). The subgame-perfect equilibrium (SPE) of a game G are all strategy profiles s such that for any subgame G' of G, the restriction of s to G' is a Nash equilibrium of G'.

The SPE is refinement of the Nash equilibrium for the perfect-information extensive for games and helps us not to reach implausible Nash equilibria. The algorithm of finding a subgame-perfect Nash equilibrium is known as *BackwardInduction* and is a recursive procedure which ensures each subtree of the perfect-information extensive for to be in Nash equilibrium.

Pareto Optimality

Definition 16 (Pareto Domination). Strategy profile s Pareto dominates strategy profile s' if for all $i \in N, u_i(s) \geq u_i(s'),$ there exists some $j \in N$ for which $u_j(s) > u_j(s')$.

Definition 17 (Pareto optimality). Strategy profile s is Pareto optimal, or strictly Pareto efficient, if there does not exist another strategy profile $s' \in S$ that Pareto dominates s.

The concept of Pareto optimally is seen as the social good. It is the most optimal outcome of a game as seen from an outside observer's perspective who wants the socially optimal result which is in everyone's interest. Therefore the socially good outcome may or may not be same to the Nash equilibria.

Repeated Games, Stochastic Games and Bayesian Games

Repeated Games

In real-world many interactions among the agents takes place over a repeated number of times and the game being repeated is called the stage game. The games in theory may be repeated over a finite number of time or infinite number of times. These games respectively known as *finitely repeated games* and *infinitely repeated games*. The payoffs for players in repeated games are defined in terms of average rewards over the number of games played and the payoffs received.

Definition 18 (Average reward). Given an infinite sequence of payoffs $r_i^{(1)}, r_i^{(2)}, \ldots$ for player *i,* the average reward of *i* is

$$\lim_{k \to \infty} \frac{\sum_{j=1}^{k} r_i^{(j)}}{k}.$$

Discounted Rewards

The problem in defining the utility of a player in the manner of average rewards is that even if the initial payoffs for the person are highly negative and they are compensated after a lot of moves, then also the utility comes out to be 1. However it is often thought that the initial payoffs are more important than the later payoffs.

Definition 19 (Discounted reward). Given an infinite sequence of payoffs $r_i^{(1)}, r_i^{(2)}, \ldots$ for player *i,* and a discount factor β with $0 \leq \beta \leq 1$, the future discounted reward of *i* is $\sum_{j=1}^{\infty} \beta^j r_i^{(j)}$.

Learning in Repeated Games

As the stage game is related, the agents learn from their previous experiences and therefore play in a manner which they believe shall be most suitable for the situation. The difference between other learning techniques is that the major objective of other learning techniques is to learn the environment and find an optimal strategy to act in the environment by the single agent. Whereas in this learning technique, the agent is simultaneously teaching other agents as well. The learning is therefore collaborated with teaching. A bad learner can be a very good teacher too.

The two types of learning in repeated games:

1. Fictitious Play

Each player maintains explicit beliefs about other player by counting the opponents actions and asses the opponents strategy using these counts.

2. No-regret learning.

The agent does not start with a learning method but with a criteria. The regret an experiences at time *t* for not not having played strategy s is $R^t(s) = \alpha^t - \alpha^t(s)$.

Definition 20 (No-regret learning). A learning rule exhibits no-regret learning if for any pure strategy of the agent s, it holds that $\Pr(|\lim_{games \to \infty} R^t(s)| \leq 0) = 1$.

Equilibria of infinitely repeated games

The famous strategies for infinitely repeated games involve a choice at every decision node, taking into account the history of actions and taking infinite actions. The two famous strategies are:

1. Tit-for-Tat

In this strategy the agent keeps into account the previous action by the opponent and is good to the opponent if the previous move played by the opponent was taken by taking into consideration her move and the agent is bad otherwise. The first move is taken randomly.

2. Trigger

The agent moves in socially optimal manner till the opponent moves in that manner. However if the opponent deviates from her strategy and becomes selfish, the agent shall play her self interested strategy till the end.

Nash Equilibria

For repeated games, the Nash equilibria is defined by the Folk's theorem.

Theorem 5 (Folk's Theorem). It states that for ant strategy to be in Nash equilibrium, the payoff received for that strategy should be *feasible* and *enforceable.*

Definition 21 (Feasible). A payoff profile $r = (r_1, r_2, \ldots, r_n)$ is feasible if there exists rational non-negative α_a such that for all *i,* we can express r_i as $\sum_{a \in A} \alpha_a u_i(a)$ with $\sum_{a \in A} \alpha_a = 1$.

Definition 22 (Enforceable). A payoff profile $r = (r_1, r_2, \ldots, r_n)$ is enforceable if $\forall i \in N,\ r_i \geq v_i$.

Stochastic Games

A stochastic game is a generalization of repeated games where the same stage game is not repeated instead the agents play games from a set of normal-form games. Moreover, the game played at any iteration depends on the previous game played and on the actions taken by all agents in that game. Hence the transition from one game to the next is probabilistically defined. The stochastic games also generalize the concepts of Markov Decision Process (MDPs) where the MDP may be defined as a single agent stochastic game.

Bayesian Games

Bayesian games define the uncertainty about the utility function. They are games of incomplete information and represent the players' uncertainty about the game being played.

Definition 22 (Bayesian game). A Bayesian game is a tuple *(N, G, P, I)* where:

- *N* is a set of agents;
- *G* is a set of games with N agents each such that if *g*, $g' \in G$ then for each agent $i \in N$ the strategy space in *g* is identical to the strategy space in *g'*.
- $P \in \prod(G)$ is a common prior over games, where $\prod(G)$ is the set of all probabilities over G.
- $I = (I_1, I_2, \ldots, I_N)$ is a set of partitions of G, one for each agent.

Bayesian Games are of immense importance in the field of artificial intelligence with each agent computing the opponent in the form of a bayesian agent and agent calculates its expected utility over the actions of other players and over the types of other players.

Machine Learning

Machine learning is the branch of computer science that uses statistical techniques to give computer systems the ability to "learn" with data, without being explicitly programmed. The agents can improve their behavior through diligent study of their own experiences. The machine trains itself using the training data, makes a hypothesis function and then tests on the test data to make the prediction accuracy. The field of machine learning is closely related to the field pf pattern recognition which is concerned with the automatic discovery of regularities in data through the use of computer algorithms and with the use of these regularities to take actions such as classifying the data into different categories. The field of machine learning is broadly divided into two main categories, Supervised and Unsupervised learning of which algorithms of supervised learning shall be discussed in the report.

Supervised Learning

Applications in which the training data comprises examples of the input vectors along with their corresponding target vectors are known as supervised learning problems.

Notation

1. $x^{(i)}$ is used to denote the input vector or the input features.
2. $y^{(i)}$ is used to denote the output vector or the target variables which are being predicted.
3. $(x^{(i)}, y^{(i)})$ is called a training example and for the discussion, there would be *m* number of training examples $\{(x^{(i)}, y^{(i)}); i = 1,\ldots,m\}$ is the training set.
4. χ denotes the space of input values and γ denotes the space of output values.

Hypothesis

In formal description, the goal of supervised learning is to learn a function $h : \chi \mapsto \gamma$ so that *h(x)* is a good predictor of the corresponding value of *y.* This function h is called a hypothesis.

When the output values range is continuous, it is known as a *regression* problem, whereas when the target vector can take up only a small number of values, it known as a *classification* problem.

Linear Regression

The motivation of this learning technique is to predict accurately the values of the output vectors which vary linearly with respect to the input vectors. The hypothesis function is therefore a linear mapping from the input space, χ to the output space γ and is defined as:

$$h(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T x,$$

where θ_i's are the parameters (also called weights) parametrizing the hypothesis function. The task at hand is to learn these parameters to output a good prediction. To simplify the notation, $x_0 = 1$ has been introduced as the intercept term. Both θ and *x* are n dimensional vectors.

Cost Function[10]
The aim of machine learning is to predict outputs as close to the actual outputs as possible and they are therefore trained using a maximum likelihood estimate and the objective is to minimize the cost function which evaluates how close the values of $h(x^{(i)})$'s are to $y^{(i)}$'s. The cost function is defined as follows:

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{m} (h_\theta x^{(i))} - y^{(i)})^2 .$$

This algorithm is a part of the family of algorithms known as the generalized linear models.

Batch Gradient Descent
In order to arrive at the value of θ which minimizes the cost function, $J(\theta)$, the value of θ is repeatedly changed till $J(\theta)$ keeps decreasing and the update is stopped when any further change leads to an increase in the value of $J(\theta)$. The update of θ is done according to the gradient descent algorithm, which starts with an initial θ and repeatedly performs the update:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

(This update is simultaneously performed for all j = (0,1,…,n).). α is the *learning rate*. This is a very natural algorithm which takes a step in the direction of the steepest decrease in the value of *J.*

It can be derived that for one training example, $\frac{\partial}{\partial \theta_j} J(\theta) = (h_\theta(x) - y)x_j$.

For a single training example it gives the update rule:

$$\theta_j = \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}.$$

This rule is called the LMS ("least mean squares").
For a training set with more than one example, the following algorithm is used:

Repeat until convergence
{

$$\theta_j = \theta_j + \alpha \sum_{i=1}^{m} (y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)} \qquad \textit{(for every j).}$$

}

While gradient descent may be susceptible to local minima in general, the optimization problem posed here has only one global and no other local optima; thus gradient descent always converges to a global minimum.

Implementation

Problem: Predict the density of wine based on the wine's acidity.
Data Set: linearX.csv and linearY.csv are the files containing the training data with linearX containing the amount of acidity of the wine, *($x^{(i)}$'s, $x^{(i)} \in \mathfrak{R}$)* and linearY containing its density *($y^{(i)}$'s, $y^{(i)} \in \mathfrak{R}$).*
Learning Rate: 0.0001
Stopping Criteria: $J(\theta') - J(\theta) < \epsilon$ *such that* $\epsilon = 0.000000001$.

Code

```
import csv
import numpy as np
from numpy import genfromtxt
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

x=genfromtxt('linearX.csv',delimiter=',')
y=genfromtxt('linearY.csv',delimiter=',')
X=np.c_[np.ones((100,1)),x]
Y=np.c_[y]
epsilon=0.000000001
learning_rate=0.0001
theta=np.zeros((2,1))
theta0=0
theta1=0
theta_x=np.array([0])
theta_y=np.array([0])
Z=np.array([])
```

```
def cost(theta0,theta1,X,Y):
	theta=np.array(([theta0],[theta1]))
	hypothesis=np.matmul(X,theta)
	result=Y-hypothesis
	result=np.square(result)
	final_cost=np.sum(result)
	return 0.5*final_cost

hypothesis=np.matmul(X,theta)
temp=Y-hypothesis
temp1=np.matmul(np.transpose(X),temp)
new_theta=theta+learning_rate*temp1
new_theta0=new_theta[0][0]
new_theta1=new_theta[1][0]

while((cost(theta0,theta1,X,Y)-cost(new_theta0,new_theta1,X,Y))>epsilon):
	theta0=new_theta0
	theta1=new_theta1
	theta_x=np.append(theta_x,theta0)
	theta_y=np.append(theta_y,theta1)
	theta=np.array(([theta0],[theta1]))
	hypothesis=np.matmul(X,theta)
	temp=Y-hypothesis
	temp1=np.matmul(np.transpose(X),temp)
	new_theta=theta+learning_rate*temp1
	new_theta0=new_theta[0][0]
	new_theta1=new_theta[1][0]

print(theta)
fig=plt.figure()
ax=fig.add_subplot(111,projection='3d')
z=np.array([])
for i in range(len(theta_x)):
	z=np.append(z,cost(theta_x[i],theta_y[i],X,Y))
ax.scatter(theta_x,theta_y,z)
plt.show()
```

Result

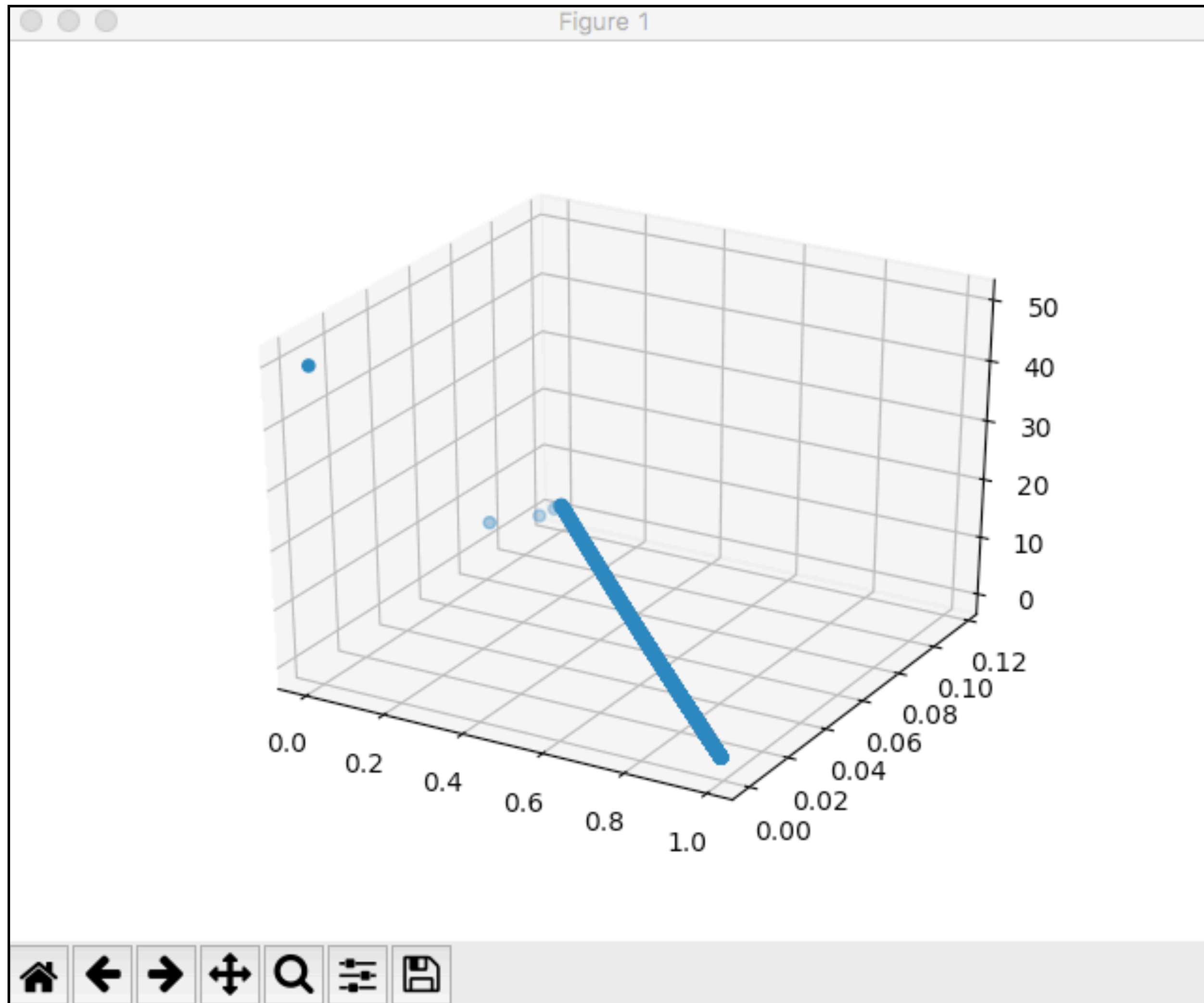
Learned parameters:

$\theta_0 = 9.89620827e-01$

$\theta_1 = 8.64285842e-04$

```
Amans-MacBook-Air:machine_learning amankhullar$ python gradient_descent.py
[[  9.89620827e-01]
 [  8.64285842e-04]]
```

Hypothesis Function



Locally Weighted Linear Regression

Locally Weighted Linear Regression (LWR) generalizes the the ideas of linear regression to model a non-parametric learning algorithm where the different training examples are weighed differently. The cost function is hence altered to be written as:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m} w^{(i)}(h_\theta x^{(i)}) - y^{(i)})^2.$$

where $w^{(i)} = \exp(-\frac{(x^{(i)} - x)^2}{2\tau^2})$, τ is the bandwidth parameter, this controls how fast the weights will off with distance.

The LWR therefore helps to fit perfectly non-linear data since every prediction depends on its neighboring data points.

The output is defined as $y = \theta^T x$.

Normal Equations

The normal equations gives a beautiful mathematical derivation for finding the learning parameters in the linear regression model. It takes a more direct, calculus oriented approach to find the value of θ for which the cost function $J(\theta)$ shall be minimized.

The input features are defined by a *design matrix* or the *feature matrix*, X, an m x n matrix where m is the number of training examples and n is the number of features of the input. $\overrightarrow{y}$ is the *output vector* or the *target vector.* The cost function in matrix notation can be written in the form:

$$J(\theta) = \frac{1}{2}(X\theta - \overrightarrow{y})^T(X\theta - \overrightarrow{y}).$$

Hence the derivative of the cost function is given by (using matrix calculus),

$$\nabla_\theta J(\theta) = X^T X\theta - X^T\overrightarrow{y}.$$

To minimize $J(\theta)$, it is equated to *0* and the value of θ that minimizes $J(\theta)$ is:

$$X^T X\theta = X^T\overrightarrow{y}$$

$$\theta = (X^T X)^{-1}X^T\overrightarrow{y}.$$

Normal Equation for Locally Weighted Linear Regression

For locally weighted linear regression, the cost function, $J(\theta)$ can be written as:

$$J(\theta) = \frac{1}{2}(X\theta - \overrightarrow{y})^T W(X\theta - \overrightarrow{y}).$$

Similar to the case of the linear regression, in order to find the normal equation for the parameters in the case of LWR, the matrix derivative of $J(\theta)$ is set to *0* given by*:*

$$\nabla_\theta J(\theta) = X^T W X\theta - X^T W\overrightarrow{y}$$

$$X^T W X\theta = X^T W\overrightarrow{y}$$

$$\theta = (X^T W X)^{-1}X^T W\overrightarrow{y}.$$

Implementation

Problem: Locally Weighted linear Regression in Normal Form.

Data Set: The data set includes two files WeightedX.csv and WeightedY.csv corresponding to the input and the output values respectively.

Bandwidth Parameter: $\tau = 0.8$

Code

```
import numpy as np
from numpy.linalg import inv
import matplotlib.pyplot as plt

x=np.genfromtxt('weightedX.csv',delimiter=',')
y=np.genfromtxt('weightedY.csv',delimiter=',')

X=np.c_[np.ones(len(x)),x]
Y=np.c_[y]
```

```
theta=np.matmul(np.matmul(inv(np.matmul(X.transpose(),X)),X.transpose()),Y)

print(theta)

plt.plot(x,theta[0][0]+theta[1][0]*x,'r')
plt.scatter(x,y)
plt.show()

def calculate_weight(x1,x,tau):                    #to return W for each data point
	return np.diag(np.exp(((x1-x)**2)/(-2*tau*tau)))
#Linearly weighted linear regression

weighted_hypothesis=np.array([])

for i in x:
	W=calculate_weight(i,x,0.8)

theta1=np.matmul(np.matmul(np.matmul(inv(np.matmul(np.matmul(X.transpose(),W),X)),
X.transpose()),W),Y)
	weighted_hypothesis=np.append(weighted_hypothesis,theta1[0][0]+theta1[1]
[0]*i)

#print(weighted_hypothesis)
plt.plot(x,weighted_hypothesis,'r')
plt.scatter(x,y)
plt.show()
```

Result

Learned Parameters for linear regression using Normal Equations.

$\theta_0 = 0.32767322$

$\theta_1 = 0.17531247$

```
Amans-MacBook-Air:machine_learning amankhullar$ python weighted_linear_regressio
n.py
[[ 0.32767322]
 [ 0.17531247]]
```

(The results given below are in the plots of linear regression curve using the Normal Equation and the locally weighted linear regression using Normal Equation respectively.)

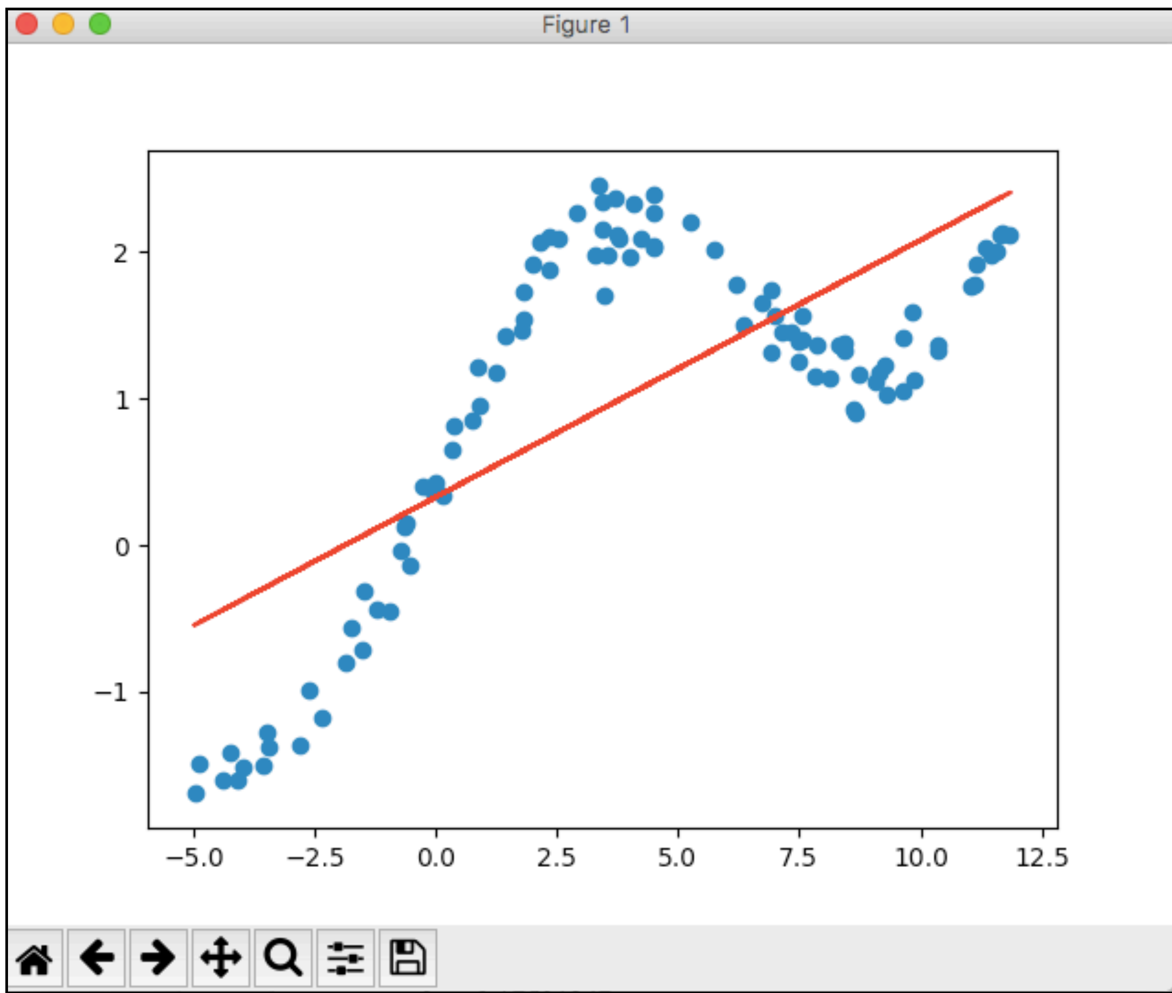


Figure 1
2
1
0
−1
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
12.5

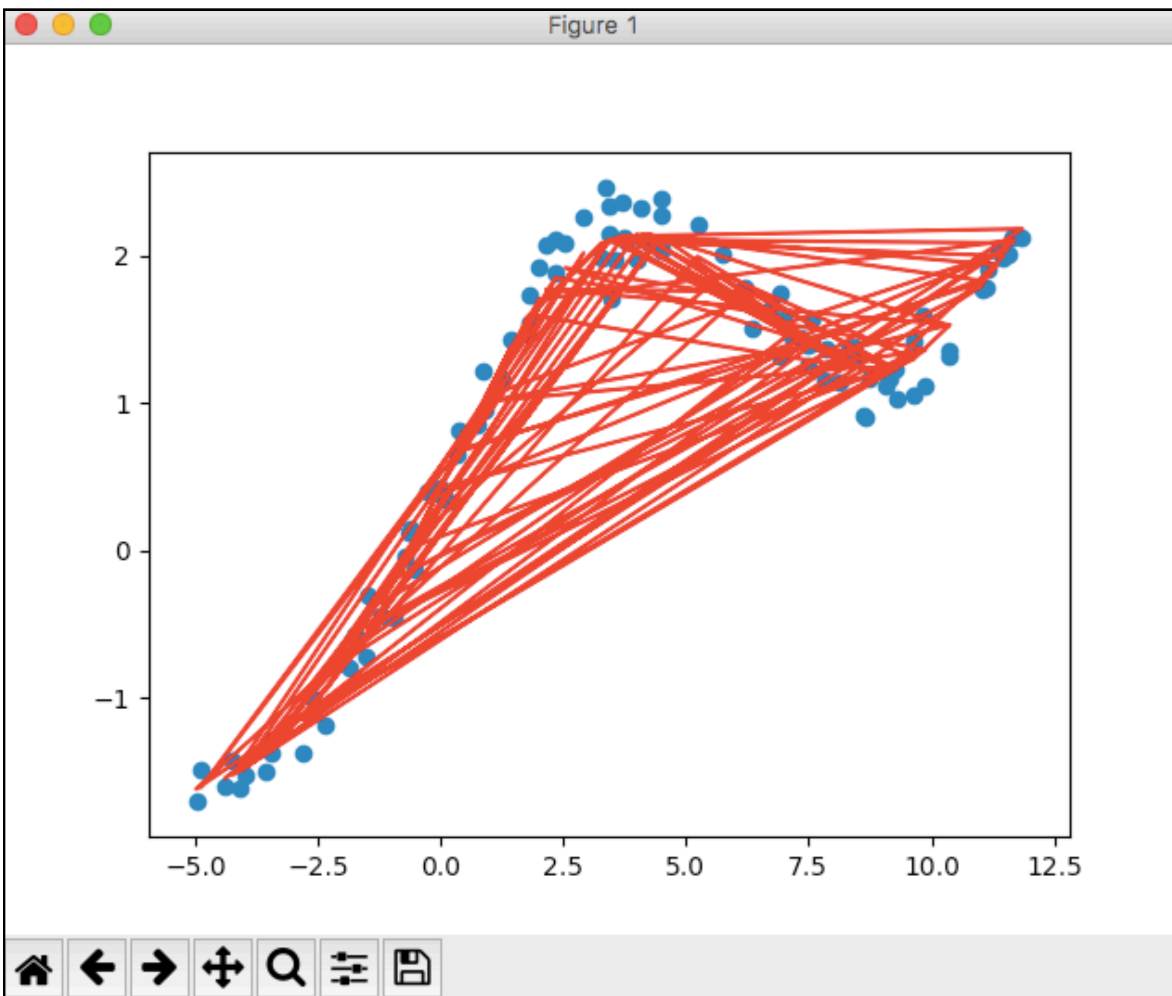


Figure 1
2
1
0
−1
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
12.5

Classification

The classification problem is just like the regression problem with the only exception that the output vector can take values from a specific range of outputs rather than continuous values. For input variables, the corresponding outputs are called labels and the classification algorithms may be multi-label or binary classification which separates the output into two classes *1* and *0*, which are also referred to as positive class or negative class respectively.

Logistic Regression

In logistic regression the The hypothesis function is defined as:

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}},$$

where

$$g(z) = \frac{1}{1 + e^{-z}}$$

This is called the *logistic* or the *sigmoid* function and enforces the hypothesis to remain bounded between 0 and 1.

For logistic regression, the assumptions taken are:

$P(y = 1 | x; \theta) = h_\theta(x)$

$P(y = 0 | x; \theta) = 1 - h_\theta(x)$

where the first equation is read as, probability of y= 1, given x and parametrized by θ.

This can be written more compactly as :

$P(y | x; \theta) = (h_\theta(x))^y (1 - h_\theta(x))^{1-y}$.

Assuming the m training examples to be generated independently, the likelihood of the parameters is written as

$$L(\theta) = p(\overrightarrow{y} | X; \theta)$$

$$L(\theta) = \prod_{i=1}^{m} p(y^{(i)} | x^{(i)}; \theta)$$

$$L(\theta) = \prod_{i=1}^{m} (h_\theta(x^{(i)}))^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1-y^{(i)}}.$$

Since it is easier to maximize the log of the likelihood, the log likelihood function, the optimization problem for logistic regression is considered to be the problem of finding the maximum value of parameter fitting the log likelihood.

$$l(\theta) = \log L(\theta) = \sum_{i=1}^{m} y^{(i)} \log(h_\theta(x^{(i)}))(1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})).$$

The maximization is done through matrix calculus which gives the gradient ascent rule:

$\theta_j = \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}$.

An important observation is that this rule is similar to the parameter upgrade rule of linear regression, however the hypothesis function is entirely different in this case.

Generative Learning Algorithms

The classification algorithms that have been talked about until now model $p(y|x;\theta)$, the conditional distribution of *y* given *x*. The classification algorithm for example, logistic regression mapped the output based on the input features and then accordingly trained a hypothesis function according to which the decision was made based on what side of the decision boundary the unseen training example resides. However the algorithm can also be trained in another manner by making the algorithm first understand the features of a particular kind by exposing the algorithm explicitly and then after the algorithm has learned the features, it is tested on the unseen examples to predict the class of higher probability.

Algorithms that try to learn $p(y|x)$ directly (such as logistic regression) or algorithms that try to learn mapping directly from the space of inputs χ to the labels *{0,1}* are called *discriminative learning algorithms.*

The algorithms that instead model $p(x|y)$ *(and* $p(y)$*)* are called *generative learning algorithms.* After modeling $p(y)$ and $p(x|y)$, the algorithm uses the Bayes Rule to derive the posterior distribution of y given x:

$p(y|x) = \dfrac{p(x|y)p(y)}{p(x)}$, where $p(x) = p(x|y=1)p(y=1) + p(x|y=0)p(y=0)$.

Gaussian Discriminative Analysis

This is among the most widely used generative learning algorithms. In this model, it is assumed that $p(x|y)$ is distributed according to a multivariate normal distribution.

Multivariate Normal Distribution

The multivariate normal distribution in n-dimensions, is parametrized by a *mean vector* $\mu \in \Re^n$ and a *covariance matrix* $\sum \in \Re^{n\times n}$ where $\sum \geq 0$ is symmetric and positive semi-definite. The distribution is defined as:

$$\mathcal{N}(\mu, \sum) = p(x;\mu,\sum) = \frac{1}{(2\pi)^{n/2}|\sum|^{1/2}}\exp(-\frac{1}{2}(x-\mu)^T(\sum)^{-1}(x-\mu)).$$

$|\sum|$ is the determinant of the matrix $\sum$.

The Gaussian Discriminative Analysis Model

In this classification problem, *x* are the input features which are continuous-valued random variables. The Gaussian Discriminative Model (GDA) models p(x|y) using multivariate normal distribution. The model is:

$$y \sim Bernoulli(\phi);$$
$$x|y=0 \sim \mathcal{N}(\mu_0, \sum);$$
$$x|y=1 \sim \mathcal{N}(\mu_1, \sum).$$

The distributions are given by:

$p(y) = \phi^y(1-\phi)^{1-y}$

$$p(x|y=0) = \frac{1}{(2\pi)^{n/2}|\sum|^{1/2}} \exp(-\frac{1}{2}(x-\mu_0)^T(\sum)^{-1}(x-\mu_0))$$
$$p(x|y=1) = \frac{1}{(2\pi)^{n/2}|\sum|^{1/2}} \exp(-\frac{1}{2}(x-\mu_1)^T(\sum)^{-1}(x-\mu_1))$$

The parameters of the model are $\phi, \sum, u_0,$ and μ_1.
The log-likelihood of the data is given by

$$l(\phi, \mu_0, \mu_1, \sum) = \log \prod_{i=1}^{m} p(x^{(i)}, y^{(i)}; \phi, \mu_0, \mu_1, \sum)$$
$$= \log \prod_{i=1}^{m} p(x^{(i)}|y^{(i)}; \mu_0, \mu_1, \sum) p(y^{(i)}; \phi)$$

By maximizing *l* with respect to the parameters, the maximum likelihood estimate of the parameters is found to be:

$$\phi = \frac{1}{m}\sum_{i=1}^{m} 1\{y^{(i)} = 1\}$$
$$\mu_0 = \frac{\sum_{i=1}^{m} 1\{y^{(i)} = 0\} x^{(i)}}{\sum_{i=1}^{m} 1\{y^{(i)} = 0\}}$$
$$\mu_1 = \frac{\sum_{i=1}^{m} 1\{y^{(i)} = 1\} x^{(i)}}{\sum_{i=1}^{m} 1\{y^{(i)} = 1\}}$$
$$\sum = \frac{1}{m}\sum_{i=1}^{m} (x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T.$$

where $1\{x\} = 1$ when x is true and 0 otherwise. This is the indicator vector notation.

Implementation

Problem: Separating out salmons from Alaska and Canada. Each salmon is represented by two attributes x_1 and x_2 depicting growth ring diameters in 1) fresh water, 2) marine water.
Data Set: "q4x.dat" stores the two attribute values with one entry on each row. "q4y.data" stores the target values ($y^{(i)} \in$ {Alaska, Canada}) on respective rows.
Assumption: $(\sum)_0 = (\sum)_1 = \sum$

Code

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

x=np.genfromtxt('q4x.dat',dtype=int)
```

```
y=np.genfromtxt('q4y.dat',dtype=str)
y=np.c_[y]

def model_feature(feature,mean,covariance):
	temp = np.exp( np.matmul( np.matmul( (feature-mean).transpose(), np.linalg.inv(covariance) ), (feature-mean) ) / (-2) )
	return temp / ( ( (2*np.pi)**(feature.shape[0]/2) ) * np.linalg.det(covariance) )


indicator_1=0
for i in y:
	if i=='Canada':
		indicator_1+=1

indicator_0=y.shape[0]-indicator_1

#Calculating Phi
phi=indicator_1/y.shape[0]

#Calculating Mu0
Mean_0=0
for i in range(0,y.shape[0]):
	if y[i]=='Alaska':
		Mean_0+=x[i]

Mean_0=Mean_0/indicator_0
Mean_0=np.c_[Mean_0]

#Calculating Mu1
Mean_1=0
for i in range(0,y.shape[0]):
	if y[i]=='Canada':
		Mean_1+=x[i]

Mean_1=Mean_1/indicator_1
Mean_1=np.c_[Mean_1]

#Calculating Covariance
covariance=0
for i in range(0,y.shape[0]):
	if y[i]=='Alaska':
		covariance+=(np.matmul((x[i].transpose()-Mean_0),(x[i].transpose()-Mean_0).transpose()))
	else:
		covariance+=(np.matmul((x[i].transpose()-Mean_1),(x[i].transpose()-Mean_1).transpose()))
```

```
covariance=covariance/y.shape[0]

#printing mean and variances
print("The Mean for the Alaska training set is\n",Mean_0,"\n")
print("The Mean for the Canada training set is\n",Mean_1,"\n")
print("The Covariance Matrix for the multivariate gaussian distribution is\n",covariance,"\n")

test=np.array([[0],[0]])
test[0][0] = int(input('Enter the features of the fish\n'))
test[1][0] = int(input())

#Calculating the probability of the fish coming from Alaska
probability_Alaska = model_feature(test,Mean_0,covariance)*(1-phi)
#calculating the probability of the fish coming from Canada
probability_Canada = model_feature(test,Mean_1,covariance)*phi

if probability_Alaska > probability_Canada :
	print('Fish is from Alaska\n')
else:
	print('Fish is from Canada\n')


fig=plt.figure()
ax=fig.add_subplot(111,projection='3d')

for i in range(0,y.shape[0]):
	if y[i]=='Alaska':
		ax.scatter(x[i][0],x[i][1],zs=0,c='r',marker='x')
	else:
		ax.scatter(x[i][0],x[i][1],zs=0,c='b',marker='o')

ax.set_xlabel('Fresh Water')
ax.set_ylabel('Marine Water')

c = np.matmul( np.matmul( Mean_0.transpose(),np.linalg.inv(covariance) ), Mean_0 ) - np.matmul( np.matmul( Mean_1.transpose(),np.linalg.inv(covariance) ), Mean_1 ) - 2*np.log((1-phi)/phi)
b = 2 * np.matmul( (Mean_1-Mean_0).transpose(),np.linalg.inv(covariance) )

x_1=np.zeros((1,x.shape[0]))
x_2=np.zeros((1,x.shape[0]))

for i in range(0,x.shape[0]):
	x_1[0][i]=x[i][0]
```

```
        x_2[0][i]=x[i][1]

print(b)
print(c)

plt.plot(x_1,((b[0][0]*x_1+c)/(-1*b[0][1])),'r' )

ax.view_init(elev=90,azim=90)
plt.show()
```

Result

$$\mu_0 = \begin{bmatrix} 98.38 \\ 429.66 \end{bmatrix}$$

$$\mu_1 = \begin{bmatrix} 137.46 \\ 366.62 \end{bmatrix}$$

$$\sum = \begin{bmatrix} 82541.104 & 1410.732 \\ 1410.732 & 82541.104 \end{bmatrix}$$

Input Feature values: 132 and 42.

```
Amans-MacBook-Air:machine_learning amankhullar$ python gda.py
The Mean for the Alaska training set is
 [[  98.38]
 [ 429.66]]

The Mean for the Canada training set is
 [[ 137.46]
 [ 366.62]]

The Covariance Matrix for the multivariate gaussian distribution is
 [[ 82541.104   1410.732]
 [  1410.732  82541.104]]

Enter the features of the fish
132
42
Fish is from Canada

[[ 0.00097331 -0.00154412]]
[[ 0.50000148]]
```

Support Vector Machines

Support Vector Machines (SVMs) are among the best supervised learning algorithms. They take into exhaustive consideration of vector representation of the training examples and divide the linearly separable labels with the help of margin and the greater the margin, the more accurate prediction there can be. Though they are defined for linearly separable classifiers, they are extended to non-linearly separable classifiers with the help of Kernels, which make the SVMs work like a charm for non-linearly separable data.

A single decision rule is defined which decides the class of label based on the decision rule. The decision rule is the median line of the gutter, which is defined as the vectors lying on the margins of the two types of labels. The width is defined as the width of the street.

The basic intuition of SVMs as stated earlier is that the greater the width of the street, the greater the accuracy of prediction. Hence the task is to maximize the width under a given set of constraints. This is beautifully accounted by the Lagrange's multipliers.

Decision Rule: $\vec{w} \, . \, \vec{u} + b \geq 0$ for positive examples

Function: $\frac{1}{2} || \vec{w} ||^2$

Constraint: $y_i(\vec{x_i} \, . \, \vec{w} + b) - 1$

where $y_i = +1$ and -1 for positive and negative examples respectively.

$\vec{x_i}$ is the input data in vector space.

$\vec{w}$ is the vector perpendicular to the median line of the margin.

b is a positive constant.

Using Lagrange's Multipliers,

$$L = \frac{1}{2} || \vec{w} ||^2 - \sum_{i=1}^{m} \alpha_i [y_i(\vec{x_i} \, . \, \vec{w} + b) - 1],$$

Differentiating to find the extremums, it can be proved that the decision rule depends only on the dot product of the unknown $\vec{u}$ and the sample vectors $\vec{x_i}$.

Hence the decision rule becomes,

$\sum_{i=1}^{m} \alpha_i y_i \vec{x_i} \vec{u} + b \geq 0$ then it will belong to positive class else the unknown will belong to the negative class.

SVM Optimization Problem:

$$[\frac{1}{m} \sum_{i=1}^{m} max(0, 1 - y_i(\vec{w} \, . \, \vec{x} + b))] + \lambda || \vec{w} ||^2$$

where, λ is the tradeoff between increasing the margin-size and ensuring that $\vec{x}$ lies on the correct side of the margin.

Implementation

Problem: Build a handwritten digit classifier using the mini-batch Pegasos algorithm and the customized solver LIBSVM.
Data Set: A subset of the MNIST dataset with 2500 training examples and 2500 testing examples. Each row in the data file corresponds to an image of size 28 x 28, represented as a vector of grayscale pixel intensities followed by the label associated with the image. Every column represents a feature where the feature value denotes the grayscale value (0-255) of the corresponding pixel in the image. There is a feature for every pixel in the image. Last column gives the corresponding label.
Algorithm: The "Pegasos: Primal Estimated sub-GrAdient SOlver for SVM" has been used to solve for *w* and *b.* The mini-batch size is taken to be 100. The algorithm is given by:

INPUT: S, λ, T, k
INITIALIZE: Set $\mathbf{w}_1 = 0$
FOR $t = 1, 2, \ldots, T$
  Choose $A_t \subseteq [m]$, where $|A_t| = k$, uniformly at random
  Set $A_t^+ = \{i \in A_t : y_i \langle \mathbf{w}_t, \mathbf{x}_i \rangle < 1\}$
  Set $\eta_t = \frac{1}{\lambda t}$
  Set $\mathbf{w}_{t+1} \leftarrow (1 - \eta_t \lambda)\mathbf{w}_t + \frac{\eta_t}{k} \sum_{i \in A_t^+} y_i \mathbf{x}_i$
  [Optional: $\mathbf{w}_{t+1} \leftarrow \min\left\{1, \frac{1/\sqrt{\lambda}}{\|\mathbf{w}_{t+1}\|}\right\} \mathbf{w}_{t+1}$]
OUTPUT: $\mathbf{w}_{T+1}$

Code
```
import numpy as np
X=np.genfromtxt('mnist/train.csv',delimiter=',',dtype=int)
X=np.c_[np.ones((X.shape[0]),dtype=int),X]
y=np.c_[X[:,X.shape[1]-1]]
X=X[:,:-1]
lambda_1=0.00001
Y=np.zeros((X.shape[0],10),dtype=int)-1
k=100
T=1000
for i in range(0,y.shape[0]):
	Y[i][y[i]-1]=1
```

```
def main():
	W=np.zeros((X.shape[1],1))
	for label in range(0,10):
		w=np.c_[np.zeros((X.shape[1]),int)]
		for t in range(1,T+1):
			A=np.random.uniform(0,X.shape[0]-1,k)
			for i in range(0,A.size):
				A[i]=int(A[i])
			A_plus=np.array([])
			for i in A:
				if(  (  np.matmul(w.transpose(),np.c_[X[int(i),:]])*Y[int(i)][label]  )
[0][0]<1 ):
					A_plus=np.append(A_plus,int(i))
				#print(int(i))
			eta=1/(lambda_1*t)
			sum_total=0
			for i in A_plus:
				sum_total+=(Y[int(i)][label]*X[int(i),:])
			w=(1-eta*lambda_1)*np.c_[w]+(eta/k)*np.c_[sum_total]
		W=np.append(W,w,axis=1)
	W=W[:,1:]
	return W

def testing(W):
	X=np.genfromtxt('mnist/test.csv',delimiter=',',dtype=int)
	X=np.c_[np.ones((X.shape[0]),dtype=int),X]
	y=np.c_[X[:,X.shape[1]-1]]
	X=X[:,:-1]
	Y=np.zeros((X.shape[0],10),dtype=int)-1
	for i in range(0,y.shape[0]):
		Y[i][y[i]-1]=1
	result=np.matmul(X,W)
	print(result[0:100,:])
	correct=0
	for i in range(0,result.shape[0]):
		max_1=-1000
		for j in range(0,result.shape[1]):
			if result[i][j]>0:
				max_1=(j+1)%10
		if y[i]==max_1:
			correct+=1
		print(max_1)
	print((correct/X.shape[0])*100,"%")

W=main()
testing(W)
```

Result
The prediction accuracy from the algorithm is 77.37%.

```
77.37 %
Amans-MacBook-Air:machine_learning amankhullar$
```

LIBSVM Implementation

$\gamma = 0.05$

C=1.0

Kernel: Gaussian Kernel, ($K(x, z) = \exp^{-\gamma * ||x-z||^2}$).

Code

```
import numpy as np
from svmutil import *

X=np.genfromtxt('../../mnist/train.csv',delimiter=',',dtype=int)
y=np.c_[X[:,X.shape[1]-1]]
X=X[:,:-1]
X=X/255
X=X.astype(str)
y=y.astype(str)

for i in range(0,X.shape[0]):
	for j in range(0,X.shape[1]):
		X[i][j]=str(j)+":"+X[i][j]

np.savetxt('data.txt',np.c_[y,X],fmt='%s')

a,b=svm_read_problem('data.txt')
prob=svm_problem(a,b)
param=svm_parameter('-s 0 -t 0 -c 1')
m=svm_train(prob,param)

X_test=np.genfromtxt('../../mnist/test.csv',delimiter=',',dtype=int)
y_test=np.c_[X_test[:,X_test.shape[1]-1]]
X_test=X_test[:,:-1]
X_test=X_test/255
X_test=X_test.astype(str)
y_test=y_test.astype(str)

for i in range(0,X_test.shape[0]):
	for j in range(0,X_test.shape[1]):
		X_test[i][j]=str(j)+":"+X_test[i][j]
```

```
np.savetxt('data_test.txt',np.c_[y_test,X_test],fmt='%s')
a_test,b_test=svm_read_problem('data_test.txt')
p_labels, p_acc, p_vals = svm_predict(a_test,b_test,m)
print(p_acc)
param=svm_parameter('-s 0 -t 2 -g 0.05 -c 1')
m=svm_train(prob,param)
p_labels, p_acc, p_vals = svm_predict(a_test,b_test,m)
print(p_acc)
```

Result
The prediction accuracy from the algorithm is 97.23%.

```
optimization finished, #iter = 2949
nu = 0.162154
obj = -327.605222, rho = -0.085296
nSV = 1877, nBSV = 42
Total nSV = 10493
Accuracy = 97.23% (9723/10000) (classification)
(97.23, 0.5448, 0.9360124703766993)
Amans-MacBook-Air:python amankhullar$
```

Interaction of Game Theory and Machine Learning

The proposed project has explored the foundational concepts of the fields of game theory and machine learning. Though the two fields are well researched areas in themselves, the research area lying at the interface of both the fields is developing at rapid rate. The advancement of computing power has made the development of applications in the fields of deep learning and reinforcement learning highly optimal. This has influenced the application of these algorithms in the areas where self interested rational agents interact among themselves which heralds the application of game theoretic solution strategies in this area.

As discussed in the theoretical description of repeated games and stochastic games, the stochastic games are a natural extension of Markov Decision Process (MDPs) to include multiple agents. The MDPs are an integral part of reinforcement learning in which the agent learns from about success and failure through reward and punishment.

Similarly, a number of other fields in artificial intelligence and even in areas of economics are taking the advantage of the usage of the both machine learning and game theory as tools to improve upon the existing state of the art techniques. The various applications of machine learning in algorithmic game theory concepts have been proposed as follows:

Machine Learning in Selfish Routing
The inefficiency of equilibria is quantified using the concept of price of anarchy which distinguishes between the selfish outcome of the agents in comparison to the optimal outcome of the game. This is exemplified through the Pigou's example, explained by the economist Pigou in 1920.

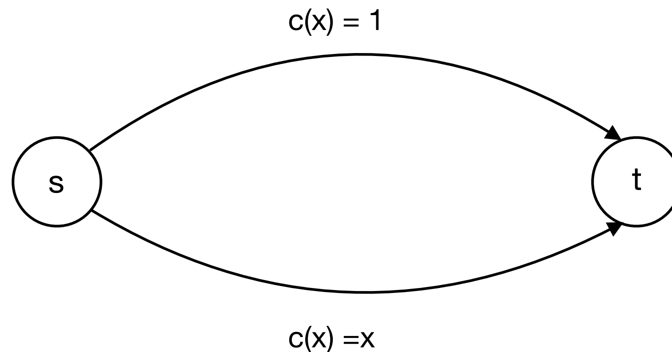


Figure: Pigou's Example

The Pigou's example illustrates a very simple network problem which illustrates the source node, s and the terminal node, t with two routes between them with costs 1 and x, respectively where x depends on the number of users using the specific edge. The upper edge on the other hand represents a constant cost edge. The costs are labelled with the cost function c(x). In an equilibrium approach, all the users prefer the lower route which gets congested as the number of users increase through that route and hence gives a total cost of 1 when all the users use the lower edge. The upper edge on the other hand could have provided a more optimal solution with half users using the upper edge and the other half using the lower edge. The total cost in such case would have been 3/4. The price of anarchy calculated in this case would therefore be 1/(3/4) = 4/3. This suboptimal performance of the network can be enhanced in the case of repeated games when the agents learn about the previous inefficiencies. The machine learning prediction models can therefore be used to improve the routing efficiency in the network with directing the node to route half the traffic through the upper edge and the other half through the lower edge.

Games improving the Web

Construction of Empire State Building: 7 million human-hours. The Panama canal: 20 million human-hours. Estimated number of human hours spent playing solitaire around the world in one year: 9 billion. [11]

The number of users on the internet playing games is extremely large. Therefore there has been a lot of work in mechanism design to develop games which would in fact improve the quality of web services. Some of the most popular game which have been responsible for improving the functionality of the Web are:

- **ESP game** (Google Image labeler): This game was developed to to resolve the problem of data classification, which is a difficult task for the computers even with sophisticated learning algorithm and efficient hardware. This game used the computational ability of humans to classify the images and then train the computer for the same.
- **CAPTCHA:** CAPTCHA is an acronym for **C**ompletely **A**utomated **P**ublic **T**uring test to **C**omputers and **H**umans **A**part. This test is used on various authentication platforms and is mostly used for network security to disallow hackers from running scripts to distort sensitive data. Various models in artificial intelligence are tested on the CAPTCHA and are predicted to be able to solve hard AI problems if they are successful in solving CAPTCHA. There have been increasing use of machine learning and Optical Character Recognition, machine learning attacks on CAPTCHAs to invade the network security.

Electronic Market Design

The number of online markets has grown throughout the world with companies like Google, Amazon and eBay bringing customers and markets online. This has resulted in a large number of mechanism being designed for the revenue generation for the companies and at the same time convenient models of expenditure for the customer. The pay-per click policy by google was a major revenue model change for the online mechanisms. Similarly research for online algorithms based on the changing users are being designed and machine learning is being incorporated to understand particular users and learn their choices which are then used for their future choices. This has therefore brought about exciting algorithmic challenges which are solved by using the concepts of machine leaning, economics theory and game theory.

Conclusion

The major results highlighted by this paper include the various solution models in game theory and the efficient approach of the usage of different models in different scenarios. The second part illustrates the comparison and implementation of the basic as well the sophisticated machine learning regression and classification algorithms which can be used and further improved upon. The last part of the paper is based on the interface of these two exciting fields and how they are coming together to solve exciting challenges and to improve the existing problems by making the solutions more efficient and optimum. The future scope of work lies in implementing the algorithms for implementing these ideas in developing fields of 5G Wireless Networks and electronic market design for online shopping and working on new routing protocols using these analysis methods form the algorithmic game theory.