CptS 570 Machine Learning, Fall 2018
Homework #2
Due Date: Tue, Oct 23 (9:10am)

NOTE 1: Please use a word processing software (e.g., Microsoft word or Latex) to write your answers and submit a printed copy to me at the beginning of class on Oct 23. The rationale is that it is sometimes hard to read and understand the hand-written answers.

NOTE 2: Please ensure that all the graphs are appropriately labeled (x-axis, y-axis, and each curve). The caption or heading of each graph should be informative and self-contained.

1. (**10 points**) Suppose $x = (x_1, x_2, \cdots, x_d)$ and $z = (z_1, z_2, \cdots, z_d)$ be any two points in a high-dimensional space (i.e., d is very large).

   a. (**5 points**) Try to prove the following, where the right-hand side quantity represent the standard Euclidean distance.

   $$\left(\frac{1}{\sqrt{d}}\sum_{i=1}^{d} x_i - \frac{1}{\sqrt{d}}\sum_{i=1}^{d} z_i\right)^2 \leq \sum_{i=1}^{d}(x_i - z_i)^2 \tag{1}$$

   Hint: Use Jensen's inequality – If X is a random variable and f is a convex function, then $f(\mathbb{E}[X]) \leq \mathbb{E}[f(X)]$.

   b. (**5 points**) We know that the computation of nearest neighbors is very expensive in the high-dimensional space. Discuss how we can make use of the above property to make the nearest neighbors computation efficient?

2. (**10 points**) We briefly discussed in the class about Locality Sensitive Hashing (LSH) algorithm to make the nearest neighbor classifier efficient. Please read the following paper and briefly summarize the key ideas as you understood:

   Alexandr Andoni, Piotr Indyk: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. Communications of ACM 51(1): 117-122 (2008) `http://people.csail.mit.edu/indyk/p117-andoni.pdf`

3. (**15 points**) We know that we can convert any decision tree into a set of if-then rules, where there is one rule per leaf node. Suppose you are given a set of rules $R = \{r_1, r_2, \cdots, r_k\}$, where r_i corresponds to the i^{th} rule. Is it possible to convert the rule set R into an equivalent decision tree? Explain your construction or give a counterexample.

4. (**10 points**) You are provided with a training set of examples (see Figure 1). Which feature will you pick first to split the data as per the ID3 decision tree learning algorithm? Show all your work: compute the information gain for all the four attributes and pick the best one.

5. (**10 points**) Please read the following paper and briefly summarize the key ideas as you understood (You can skip the proofs, but it is important to understand the main results):

   Andrew Y. Ng, Michael I. Jordan: On Discriminative vs. Generative Classifiers: A comparison of logistic regression and naive Bayes. NIPS 2001: 841-848 `http://ai.stanford.edu/~ang/papers/nips01-discriminativegenerative.pdf`

6. (**10 points**)

   a. Let us assume that the training data satisfies the Naive Bayes assumption (i.e., features are independent given the class label). As the training data approaches infinity, which classifier will produce better results, Naive Bayes or Logistic Regression? Please explain your reasoning.

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Figure 1: Table with training examples. Each row corresponds to a single training example. There are four features, namely, outlook, temperature, humidity, and wind. "PlayTennis" is the class label.

b. Let us assume that the training data does **NOT** satisfy the Naive Bayes assumption. As the training data approaches infinity, which classifier will produce better results, Naive Bayes or Logistic Regression? Please explain your reasoning.

7. (**10 points**)

   a. Can we compute $P(X)$ from the learned parameters of a Naive Bayes classifier? Please explain your reasoning.

   b. Can we compute $P(X)$ from the learned parameters of a Logistic Regression classifier? Please explain your reasoning.

8. (**15 points**) In the class, we looked at the log-likelihood derivation and the corresponding gradient ascent algorithm to find the parameters of a binary logistic regression classifier (see slide 12 and slide 13). We want to extend the log-likelihood derivation and parameter learning algorithm to the multi-class case. Suppose we have K different classes, and the posterior probability can be represented using the so-called soft-max function (see slide 18):

$$P(y = k|x) = \frac{exp(w_k \cdot x)}{\sum_{i=1}^{K} exp(w_i \cdot x)} \tag{2}$$

   a. Derive the log-likelihood and the corresponding gradient ascent algorithm to find the parameters.

   b. Add a regularization term to the log-likelihood objective (see slide 16), and derive the gradient ascent update rule with the additional change.

9. (**30 points**) Fortune Cookie Classifier[1]

   You will build a binary fortune cookie classifier. This classifier will be used to classify fortune cookie messages into two classes: messages that predict what will happen in the future (class 1) and messages that just contain a wise saying (class 0). For example,

[1]Thanks to Weng-Keen Wong and his advisor Andrew Moore for sharing the data.

"Never go in against a Sicilian when death is on the line" would be a message in class 0.
"You will get an A in Machine learning class" would be a message in class 1.

Files Provided There are three sets of files. All words in these files are lower case and punctuation has been removed.

1) The training data:

traindata.txt: This is the training data consisting of fortune cookie messages.
trainlabels.txt: This file contains the class labels for the training data.

2) The testing data:

testdata.txt: This is the testing data consisting of fortune cookie messages.
testlabels.txt: This file contains the class labels for the testing data.

3) A list of stopwords: stoplist.txt

There are two steps: the pre-processing step and the classification step. In the pre-processing step, you will convert fortune cookie messages into features to be used by your classifier. You will be using a bag of words representation. The following steps outline the process involved:

Form the vocabulary. The vocabulary consists of the set of all the words that are in the training data with stop words removed (stop words are common, uninformative words such as "a" and "the" that are listed in the file stoplist.txt). The vocabulary will now be the features of your training data. Keep the vocabulary in alphabetical order to help you with debugging.

Now, convert the training data into a set of features. Let M be the size of your vocabulary. For each fortune cookie message, you will convert it into a feature vector of size M. Each slot in that feature vector takes the value of 0 or 1. For these M slots, if the ith slot is 1, it means that the ith word in the vocabulary is present in the fortune cookie message; otherwise, if it is 0, then the ith word is not present in the message. Most of these feature vector slots will be 0. Since you are keeping the vocabulary in alphabetical order, the first feature will be the first word alphabetically in the vocabulary.

a) Implement the Naive Bayes Classifier (with Laplace Smoothing) and run it on the training data. Compute the training and testing accuracy.

To debug and test your implementation, you can employ Weka (weka.classifiers.bayes.NaiveBayes): `http://www.cs.waikato.ac.nz/ml/weka/downloading.html` or scikit-learn (`http://scikit-learn.org/stable/modules/naive_bayes.html`)

b) Run the off-the-shelf Logistic Regression classifier from Weka (weka.classifiers.functions.Logistic) or scikit-learn (`http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html`) on the training data. Compute the training and testing accuracy.

10. (**30 points**) Income Classifier using Decision Trees. You will use the *Adult Income* dataset from HW1 for this question.

    a) Implement the ID3 decision tree learning algorithm that we discussed in the class. The key step in the decision tree learning is choosing the next feature to split on. Implement the information gain heuristic for selecting the next feature. Please see lecture notes or `https://en.wikipedia.org/wiki/ID3_algorithm` for more details. I explained how to select candidate thresholds for continuous features: Sort all candidate values for feature f from training data. Suppose $f_1, f_2, \cdots, f_n$ is the sorted list. The candidate thresholds are chosen as $f_i + (f_{i+1} - f_i)/2$ for i=1 to n.

    b) Run the decision tree construction algorithm on the training examples. Compute the accuracy on validation examples and testing examples.

c) Implement the decision tree pruning algorithm discussed in the class (via validation data).

d) Run the pruning algorithm on the decision tree constructed using training examples. Compute the accuracy on validation examples and testing examples. List your observations by comparing the performance of decision tree with and without pruning.

To debug and test your implementation, you can employ Weka (weka.classifiers.trees.J48): `http://www.cs.waikato.ac.nz/ml/weka/downloading.html` or scikit-learn (`http://scikit-learn.org/stable/modules/tree.html`).

Instructions for Code Submission and Output Format.

Please follow the below instructions. It will help us in grading your programming part of the homework. We will provide a dropbox folder link for code submission.

- Supported programming languages: Python, Java, C++
- Store all the relevant files in a folder and submit the corresponding zipfile named after your student-id, e.g., 114513209.zip
- This folder should have a script file named

  `run_code.sh`

  Executing this script should do all the necessary steps required for executing the code including compiling, linking, and execution
- Assume relative file paths in your code. Some examples:

  ‘‘./filename.txt’’ or ‘‘../hw2/filename.txt’’
- The output of your program should be dumped in a file named “output.txt”
- Make sure the output.txt file is dumped when you execute the script

  `run_code.sh`
- Zip the entire folder and submit it as

  `<student_id>.zip`

Grading Rubric

Each question in the students work will be assigned a letter grade of either A,B,C,D, or F by the Instructor and TAs. This five-point (discrete) scale is described as follows:

- **A) Exemplary (=100%).**
  Solution presented solves the problem stated correctly and meets all requirements of the problem.
  Solution is clearly presented.
  Assumptions made are reasonable and are explicitly stated in the solution.
  Solution represents an elegant and effective way to solve the problem and is not overly complicated than is necessary.

- **B) Capable (=75%).**
  Solution is mostly correct, satisfying most of the above criteria under the exemplary category, but contains some minor pitfalls, errors/flaws or limitations.

- **C) Needs Improvement (=50%).**
  Solution demonstrates a viable approach toward solving the problem but contains some major pitfalls, errors/flaws or limitations.

- **D) Unsatisfactory (=25%)**
  Critical elements of the solution are missing or significantly flawed.
  Solution does not demonstrate sufficient understanding of the problem and/or any reasonable directions to solve the problem.

- **F) Not attempted (=0%)**
  No solution provided.

The points on a given homework question will be equal to the percentage assigned (given by the letter grades shown above) multiplied by the maximum number of possible points worth for that question. For example, if a question is worth 6 points and the answer is awarded a B grade, then that implies 4.5 points out of 6.

Assignment 2

Sheryl Mathew (11627236)

October 28, 2018

1. Jensen's Inequality and Nearest Neighbors Computation Efficiency

a. Jensen's Inequality

Solution 1:

Standard Euclidean distance,

$$d(x,z) = \sqrt{(x_1 - z_1)^2 + (x_2 - z_2)^2 + \cdots (x_d - z_d)^2}$$

Squared Euclidean distance,

$$d^2(x,z) = (x_1 - z_1)^2 + (x_2 - z_2)^2 + \cdots (x_d - z_d)^2$$

By Jensen's inequality,

$$f(E[X]) \leq E[f(X)]$$

$$f\left[\sum_{i=1}^{d}(x_i - z_i)^2\right] \leq \sum_{i=1}^{d} f(x_i - z_i)^2$$

$$f\left[\sum_{i=1}^{d}(x_i - z_i)^2\right] \leq (x_1 - z_1)^2 + (x_2 - z_2)^2 + \cdots (x_d - z_d)^2$$

$$f\left[\sum_{i=1}^{d}(x_i - z_i)^2\right] \leq d^2(x,z)$$

$$f\sqrt{\left[\sum_{i=1}^{d}(x_i - z_i)^2\right]} \leq d(x,z)$$

Hence this proves that $\sum_{i=1}^{d}(x_i - z_i)^2$ is equal to the standard Euclidean distance.

Solution 2:

$$\left(\frac{1}{\sqrt{d}}\sum_{i=1}^{d} x_i - \frac{1}{\sqrt{d}}\sum_{i=1}^{d} z_i\right)^{\mathbf{2}} \leq \sum_{i=0}^{d}(x_i - z_i)^2$$

$$\left(\frac{1}{\sqrt{d}}\sum_{i=1}^{d}(x_i - z_i)\right)^2 \leq \sum_{i=1}^{d}(x_i - z_i)^2$$

Let $(x_i - z_i) = s_i$

$$\left(\frac{1}{\sqrt{d}}\sum_{i=1}^{d} s_i\right)^2 \leq \sum_{i=1}^{d}(s_i)^2$$

$$\frac{1}{d}\left(\sum_{i=1}^{d} s_i\right)^2 \leq \sum_{i=1}^{d}(s_i)^2$$

By Jensen's inequality,

$$f(E[X]) \leq E[f(X)]$$

$$f\left[\frac{1}{d}\left(\sum_{i=1}^{d} s_i\right)^2\right] \leq \frac{1}{d}\sum_{i=1}^{d}(s_i)^2$$

$$\frac{1}{d^2}\left(\sum_{i=1}^{d} s_i\right)^2 \leq \frac{1}{d}\sum_{i=1}^{d}(s_i)^2$$

$$\frac{1}{d}\left(\sum_{i=1}^{d} s_i\right)^2 \leq \sum_{i=1}^{d}(s_i)^2$$

If d is greater than 1 then the left side of the quantity is less than the Euclidean distance.

b. Nearest Neighbors Computation Efficiency

Computing nearest neighbors is expensive while using Euclidean distance.
From Solution 1, the computational complexity can be reduced by using Squared Euclidean distance. This is because Squared Euclidean distance gives the same answer as Euclidean distance without the square root computation. This improves the computation efficiency of nearest neighbors.

From Solution 2, In the LHS we find all the difference between x and z points, find the sum and then take the square of the values to find nearest neighbors which is has less computation complexity when compared to the RHS where we find the difference between the x and z points, square each pair and then take the sum of all the squared terms.

2. Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions

Nearest neighbor algorithm is to find the data point that is closest to the given query point. The main concern is that when the dimension of the feature vector increases, the computation time becomes same as linear time algorithm where the query point will be compared against all the point in the database. This is referred to as the "curse of dimensionality" of Nearest neighbor algorithm.

In "Randomized c-approximate R-near neighbor or (c, R) – NN", a data structure is constructed where when a query point is given we report some cR-near neighbor of the query point with probability $1 - \delta$ where R > 0 and $\delta > 0$. The probability of success is amplified by building and querying several instances of the data structure. Either assume R = 1 or divide all coordinates by R. Thus, we can eliminate R and refer to this as "c-approximate near neighbor problem or c-NN"

In "Randomized R-near neighbor reporting", a data structure is constructed where when a query point is given we report all R-near neighbor of the query point with probability $1 - \delta$

In Locality-Sensitive Hashing, all the input points are hashed using several hash functions such that for every function the probability of collision is much higher for points that are close to each other than for those that are far apart ie similar points map to the same "buckets" with high probability. This helps to identify the query point by hashing it and finding the elements stored in the buckets containing that query point.

Consider data points which are binary (each coordinate is either 0 or 1). Compute between points p and q as Hamming distance. H is a family of functions which contains all projections of the input point on one of the coordinate i.e. H contains all functions h_i from $\{0, 1\}^d$ to $\{0, 1\}$ such that $h_i(p) = p_i$. When a hash function h is chosen randomly h(p) returns a random coordinate of p. Therefore, $P_1 = 1 - \frac{R}{d}$, $P_2 = 1 - \frac{cR}{d}$. If approximation factor, c > 1, then $P_1 > P_2$.

H cannot be used as shown above since the gap between the probabilities P_1 and P_2 will be quite small. An amplification process required to achieve the desired probabilities of collision. To amplify the gap between high probability P_1 and low probability P_2 several functions are concatenated. L functions are chosen independently and uniformly at random from H such that $g_j(q) = (h_{1,j}(q), \ldots\ldots, h_{k,j}(q))$, where $h_{t,j}$ $(1 \leq t \leq k,\ 1 \leq j \leq L)$. The data structure is constructed by placing each point p from the input set into a bucket $g_j(p)$, for j = 1,…,L. Only the nonempty buckets are retained by resorting to (standard) hashing3 of the values $g_j(p)$. To process a query point q, we scan through all the buckets $g_1(q)$,…, $g_L(q)$, and retrieve the points stored in them.

After retrieving the points, we compute their distances to the query point, and report any point that is a valid answer to the query.

Scanning Strategy 1: Interrupt the search after finding the first L points (including duplicates) for some parameter L. This solves (c, R)-near neighbor problem. Scanning Strategy 2: Continue the search until all points from all buckets are retrieved; no additional parameter is required. This solves R-near neighbor reporting problem.

Large values of k lead to a larger gap between the probabilities of collision for close points and far points are P_1^k and P_2^k respectively. This makes the hash functions more selective. So, if k is large then P_1^k is small therefore L must be large enough to ensure that an R-near neighbor collides with the query point q at least once. By calculating the expected number of collisions, we can compute the time for distance computations and the time to hash the query into all L hash tables given the data set, query point q and a set of sample queries. The sum of the estimates of these two terms is the estimate of the total query time for q. Select k value such that it minimizes this sum over a small set of sample queries. This is done in the E^2LSH package.

The existing LSH families are Hamming Distance, Jaccard and Arccos. Near-Optimal LSH Functions for Euclidean Distance is a new LSH family introduced. This is developed using LSH family for the Euclidean distance, which achieves a near-optimal separation between the collision probabilities of close and far points. This reduces the approximate nearest neighbor problem for worst-case data to the exact nearest neighbor problem over random (or pseudorandom) point configuration in low-dimensional spaces. This is a theoretical approach since asymptotic improvement in the running time achieved via a better separation of collision probabilities makes a difference only for a large number of input points in a d-dimensional Euclidean space.

In this method random projection into R^t, where t is super-constant is performed. R^t is then partitioned the space R^t into cells. The hash function returns the index of the cell which contains projected point p. Partitioning is done by creating a sequence of balls B_1, B_2…, each of radius w, with centers chosen independently at random. Each ball B_i defines a cell containing points $B_i \cup_{j<i} B_j$. This is done by replacing each ball by a grid of balls in one dimensional case, the minimum is achieved for finite w but for large w the method yields only the exponent of $\frac{1}{c}$. For that value of w, the exponent tends to $\frac{1}{c^2}$ as t tends to infinity. The query time exponent $\rho(c) = \frac{1}{c^2} + O\left(\frac{\log\log n}{\log_3^1 n}\right)$. For large n, the $(c) \sim \frac{1}{c^2}$. This proves that this is near-optimal in the class of the locality-sensitive hashing algorithms.

3. Decision tree as a set of rules

Yes, we can convert the set of rules $R = \{r_1, r_2, \ldots, r_n\}$ where r_i corresponds to the i^{th} rule into its equivalent decision tree.

Source: Converting Declarative Rules into Decision Trees [Amany Abdelhalim, Issa Traore]

A decision tree that is built on rules, assigns attributes to the nodes using criteria based on the properties of the attributes in the decision rules. They handle manipulations in the data through the rules induced from the data and not the tree itself.

To generate a decision tree, RBDT-1 method is used. The best attribute for each node is decided based on attribute effectiveness (AE), attribute autonomy (AA) and minimum value distribution (MVD).
Let $a_1, a_2 \ldots a_n$ be the attributes characterizing the data, $D_1, D_2, \ldots D_n$ be the corresponding domains i.e. D_i represents the set of values for attribute a_i and $c_1, c_2, \ldots . c_m$ be the decision classes.

The decision rules must be prepared before it can be used by RBDT-1 method. A "don't care" value is assigned to all the attributes that are omitted in any of the rules.

Consider 3 attributes a_1, a_2, a_3 of same domain D and containing values v_1, v_2, v_3 as possible values.

Let the following rules correspond to class c:

$$r_1: c \leftarrow a_1 = v_1 \& a_2 = v_2$$
$$r_2: c \leftarrow a_1 = v_3$$

After formatting,

$$r_1: c \leftarrow a_1 = v_1 \,\&\, a_2 = v_2 \,\&\, a_3 = "don't\ care"$$
$$r_2: c \leftarrow a_1 = v_3 \,\&\, a_2 = "don't\ care" \,\&\, a_3 = "don't\ care"$$

While building the tree, the best attribute is selected based on the set of rules CR calculated by AE, AA and MVD. The root node contains the whole set of CR. From each node a number of branches are pulled out. Based on the total number of values available for the corresponding attribute in CR. Each branch is associated with a reduced set of rules RR which is a subset of CR that satisfies the value of the corresponding attribute. If RR is empty, then a single node will be returned with the value of the most frequent class found in CR. If all the rules in RR assigned to the branch belong to the same decision class, a leaf node will be created and assigned value of that decision class. This process continues until each branch from the root node is terminated with a leaf node and no more further branching is required.

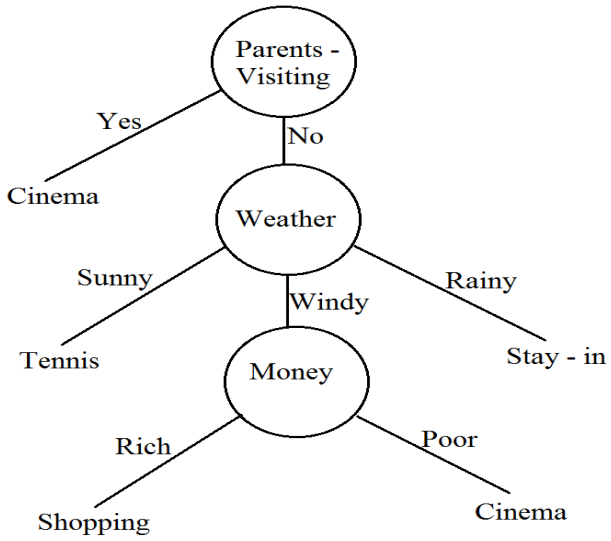
Consider the weekend dataset,

Weekend	**Weather**	**Parents**	**Money**	**Decision**
W1	Sunny	Yes	Rich	Cinema
W2	Sunny	No	Rich	Tennis
W3	Windy	Yes	Rich	Cinema
W4	Rainy	Yes	Poor	Cinema
W5	Rainy	No	Rich	Stay in
W6	Rainy	Yes	Poor	Cinema
W7	Windy	No	Poor	Cinema
W8	Windy	No	Rich	Shopping
W9	Windy	Yes	Rich	Cinema
W10	Sunny	No	Rich	Tennis

The set of rules based on which the decision tree is split,

Rule#	Description
1	Cinema ← Parents-visiting = "yes" & weather = "don't care" & Money = "rich"
2	Tennis ← Parents-visiting = "no" & weather = "sunny" & Money = "don't care"
3	Shopping ← Parents-visiting = "no" & weather = "windy" & Money = "rich"
4	Cinema ← Parents-visiting = "no" & weather = "windy" & Money = "poor"
5	Stay-in ← Parents-visiting = "no" & weather = "rainy" & Money = "poor"

The Weekend problem is a dataset containing 10 records. It consists of 3 nodes and 5 leaves with 100% classification accuracy for the data.



4. Split the data as per ID3 decision tree learning algorithm

Root Node:

$S = [5-,9+]$

$$H(S) = -\frac{5}{14}\log_2\left(\frac{5}{14}\right) - \frac{9}{14}\log_2\left(\frac{9}{14}\right) = 0.940$$

Outlook:

$S_{sunny} = [3-,2+]$

$$H(S_{sunny}) = -\frac{3}{5}\log_2\left(\frac{3}{5}\right) - \frac{2}{5}\log_2\left(\frac{2}{5}\right) = 0.971$$

$S_{overcast} = [0-,4+]$

$$H(S_{overcast}) = -\frac{0}{4}\log_2\left(\frac{0}{4}\right) - \frac{4}{4}\log_2\left(\frac{4}{4}\right) = 0$$

$S_{rain} = [2-,3+]$

$$H(S_{rain}) = -\frac{2}{5}\log_2\left(\frac{2}{5}\right) - \frac{3}{5}\log_2\left(\frac{3}{5}\right) = 0.971$$

$Information\ Gain$

$$IG(S, outlook) = H(S) - \frac{|S_{sunny}|}{|S|} H(S_{sunny}) - \frac{|S_{overcast}|}{|S|} H(S_{overcast}) - \frac{|S_{rain}|}{|S|} H(S_{rain})$$

$$IG(S, outlook) = 0.940 - \frac{5}{14}\ 0.971 - \frac{4}{14}\ 0 - \frac{5}{14}\ 0.971 = 0.247$$

Wind:

$S_{weak} = [2-,6+]$

$$H(S_{weak}) = -\frac{2}{8}\log_2\left(\frac{2}{8}\right) - \frac{6}{8}\log_2\left(\frac{6}{8}\right) = 0.811$$

$S_{strong} = [3-,3+]$

$$H(S_{strong}) = -\frac{3}{6}\log_2\left(\frac{3}{6}\right) - \frac{3}{6}\log_2\left(\frac{3}{6}\right) = 1$$

$Information\ Gain$

$$IG(S, wind) = H(S) - \frac{|S_{weak}|}{|S|} H(S_{weak}) - \frac{|S_{strong}|}{|S|} H(S_{strong})$$

$$IG(S, wind) = 0.940 - \frac{8}{14}\ 0.811 - \frac{6}{14}\ 1 = 0.048$$

Temperature:

$S_{hot} = [2-,2+]$

$$H(S_{hot}) = -\frac{2}{4}\log_2\left(\frac{2}{4}\right) - \frac{2}{4}\log_2\left(\frac{2}{4}\right) = 1$$

$S_{mild} = [2-,4+]$

$$H(S_{mild}) = -\frac{2}{6}\log_2\left(\frac{2}{6}\right) - \frac{4}{6}\log_2\left(\frac{4}{6}\right) = 0.918$$

$S_{cool} = [1-,3+]$

$$H(S_{cool}) = -\frac{1}{4}\log_2\left(\frac{1}{4}\right) - \frac{3}{4}\log_2\left(\frac{3}{4}\right) = 0.811$$

$Information\ Gain$

$$IG(S, temperature) = H(S) - \frac{|S_{hot}|}{|S|} H(S_{hot}) - \frac{|S_{mild}|}{|S|} H(S_{mild}) - \frac{|S_{cool}|}{|S|} H(S_{cool})$$

$$IG(S, temperature) = 0.940 - \frac{4}{14} 1 - \frac{6}{14} 0.918 - \frac{4}{14} 0.811 = 0.029$$

Humidity:

$S_{high} = [4-,3+]$

$$H(S_{high}) = -\frac{4}{7}\log_2\left(\frac{4}{7}\right) - \frac{3}{7}\log_2\left(\frac{3}{7}\right) = 0.985$$

$S_{normal} = [1-,6+]$

$$H(S_{normal}) = -\frac{1}{7}\log_2\left(\frac{1}{7}\right) - \frac{6}{7}\log_2\left(\frac{6}{7}\right) = 0.592$$

$Information\ Gain$

$$IG(S, humidity) = H(S) - \frac{|S_{high}|}{|S|} H(S_{high}) - \frac{|S_{normal}|}{|S|} H(S_{normal})$$

$$IG(S, humidity) = 0.940 - \frac{7}{14} 0.985 - \frac{7}{14} 0.592 = 0.152$$

The Information gains for all the attributes

$IG(S, humidity) = 0.152$ $IG(S, temperature) = 0.029$

$IG(S, outlook) = 0.247$ $IG(S, wind) = 0.048$

Since the Information gain of Outlook is maximum, we split the data on "**Outlook**" attribute.

5. On Discriminative vs Generative classifiers: A comparison of Logistic Regression and Naïve Bayes

Generative classifiers learns a model based on the joint probability p(x, y) where x is the input and y is the class label. Predictions are made on Bayes rule by calculating p(y | x) and selecting the most likely label. The generative classifier has higher asymptotic error when the number of examples is large and it approaches this asymptotic error faster with a small number of examples. An example of generative classifier is Naïve Bayes.

Discriminative classifier learns by directly mapping from the inputs x to the class label y or by modelling the posterior p(y | x) directly. The sample complexity is linear in VC dimension i.e. the VC dimension is roughly linear or at most some low order polynomial in the number of parameters used. An example of discriminative classifier is logistic regression.

The regimes of performance measured includes measuring whether the generative classifier has already approached its asymptotic error, thus doing better and whether the discriminative classifier approaches its lower asymptotic error, thus doing better.

Let h_{gen} and h_{dis} be generative and discriminative classifiers, $h_{gen,\infty}$and $h_{dis,\infty}$ be their asymptotic/population versions.

Discriminative logistic regression classifier has a smaller asymptotic error when compared to generative naïve bayes. This is because $\varepsilon(h_{gen,\infty}) \leq \varepsilon(h_{dis,\infty})$. The number of examples needed for discriminative classifier to approach asymptotic error is O(n) and for generative classifier it is O(log n) training examples. Generative naïve bayes classifier converges more quickly i.e. they reach their higher asymptotic error faster when compared to discriminative logistic regression classifier. When the number of training examples increases, generative classifier initially does better but eventually discriminative classifier catches up and overtakes the performance of generative classifier.

6. Training Data

a. As the training data approaches infinity, which classifier will produce better results, Naive Bayes or Logistic Regression when training data satisfies the Naive Bayes assumption?

If the training data approaches infinity and the data satisfies conditional independence then logistic regression is the discriminative counterpart of Naïve Bayes. This is because the parameters wi in Logistic regression can be expressed in terms of Gaussian Naive Bayes parameter.

b. As the training data approaches infinity, which classifier will produce better results, Naive Bayes or Logistic Regression when training data does not satisfy the Naive Bayes assumption?

If the training data approaches infinity and the data does not satisfy conditional independence then logistic regression is better since it does not assume that the data will satisfy conditional independence.

7. Learned Parameters

a. Can we compute P(X) from the learned parameters of a Naive Bayes classifier?

Yes, we can computer P(X) from the learned parameters of Naïve Bayes classifier since Naïve Bayes is a generative classifier which estimates P(X|Y). To calculate P(X), we marginalize over the class label y.

$$P(X) = \sum_{y} P(X|Y = y)\, \mathrm{P}(\mathrm{Y} = \mathrm{y})$$

b. Can we compute P(X) from the learned parameters of a Logistic Regression classifier?

No, we cannot computer P(X) from the learned parameters of Logistic regression classifier since Logistic regression is a discriminative classifier which estimates P(Y|X) and not P(X|Y). To calculate P(X) we need to know the value of P(X|Y)

8. Multi-class Logistic Regression Classifier

Let F_{lk} be a function where $F_{lk} = \begin{cases} 1 & Y^l = k \\ 0 & otherwise \end{cases}$,

Likelihood,

$$L(w_1, w_2 \ldots . w_k) = \prod_{l=1}^{D} \prod_{k=1}^{K} P(Y^l = k \mid X^l = x; w)^{F_{lk}}$$

$$L(w_1, w_2 \ldots . w_k) = \prod_{l=1}^{D} \prod_{k=1}^{K} \left(\frac{\exp(w_k x^l)}{\sum_{i=1}^{k} \exp(w_i x^l)} \right)^{F_{lk}}$$

Taking log,

$$l(w_1, w_2 \ldots . w_k) = \sum_{l=1}^{D} \sum_{k=1}^{K} F_{lk} \left[w_k\, x^l - \ln \sum_{i} \exp(w_i\, x^l) \right]$$

Add L_2 regularization term,

$$l(w_1, w_2 \ldots . w_k) = \sum_{l=1}^{D} \sum_{k=1}^{K} F_{lk} \left[w_k\, x^l - \ln \sum_{i} \exp(w_i\, x^l) \right] - \frac{\lambda}{2} \|w_k\|^2$$

Take derivative,

$$\partial \frac{l(w_1, w_2 \ldots w_k)}{\partial w_i} = \sum_{l=1}^{D} [\, F_{li}\, x^l - \frac{x^l \exp(w_i\, x^l)}{\sum_i w_i x^l}] - \lambda w_i$$

$$\partial \frac{l(w_1, w_2 \ldots w_k)}{\partial w_i} = \sum_{l=1}^{D} [\, F_{li} - P(Y^l = i|X^l)] x^l - \lambda w_i$$

Update rule with the gradient descent for w,

$$w_i = w_i + v \sum_{l=1}^{D} [\, F_{li} - P(Y^l = i|X^l)] \, x^l - v \, \lambda \, w_i$$

9. Fortune Cookie Dataset

Naive Bayes:

Accuracy of training data using Naive Bayes from Scratch: 95.962733
Accuracy of testing data using Naive Bayes from Scratch: 83.168317

Accuracy of training data using Naive Bayes from Sklearn Multinomial: 95.962733
Accuracy of testing data using Naive Bayes from Sklearn Multinomial: 83.168317

Logistic Regression:

Accuracy of training data using Logistic Regression from Sklearn: 94.409938
Accuracy of testing data using Logistic Regression from Sklearn: 79.207921

10. Income Dataset

Decision tree:

Accuracy of training data using Decision tree from Scratch: 78.980000
Accuracy of training data using Decision tree from Sklearn: 96.716000

Accuracy of development data using Decision tree from Scratch: 78.050398
Accuracy of development data using Decision tree from Sklearn: 78.249337

Accuracy of testing data using Decision tree from Scratch: 76.736597
Accuracy of testing data using Decision tree from Sklearn: 78.648019

The decision tree from Sklearn has become over fitted with the values of training data. Therefore the training accuracy is high but the development and testing accuracy is low when compared to the training accuracy. Pruning will help generalize better.

The decision tree from Scratch has not become over fitted with the values of training data. Therefore the development and testing accuracy is close to the training accuracy.