598 CHAPTERS 3&4 BINARY & MULTICLASS CLASSIFICATION

CONFUSION MATRIX:

- for a binary classifier w/ MNST for example if you classify a num. as not 5 then you have 90% accuracy. This is skewed.
- Better to evaluate classifier through a confusion matrix.
  - count # of times class A was classified as class B.
  - To know the number of times a classifier confused 5's with 3's, look at the 5th row & 3rd column.
  - rows = actual class, column = predicted class

PRECISION & RECALL

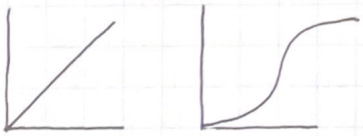
- fraction of relevant instances among the retrieved instances (precision) and recall is the fraction of relevant instances that were retrieved.
- EX. U have a program to identify dogs. 10 cats & 12 dogs. Program identifies eight dogs. Of the 8 identified only 5 are actually dogs (TP). Programs precision is 5/8 and recall is 5/12.
- precision tells us how ~~relevant~~ valid results are while recall tells us how complete.
- increasing precision reduces recall. (precision / recall tradeoff)

THE ROC CURVE

- receiver operating characteristic (ROC) curve is similar to precision / recall curve.
- ROC plots true positive rate against the false positive rate (FPR)
- FPR: ratio of negative instances that are incorrectly classified as positive.
- TNR: true negative rate is negative instances correctly classified as negative.
- one way to compare classifiers is to measure the area under the curve (AUC).
  - perfect classifier will have a ROC AUC = 1 whereas a purely random classifier will have an ROC AUC = 0.5

MULTICLASS CLASSIFICATION

- SGD, RF handle multiclass natively
- Logistic regression and SVM are strictly binary
- one-versus-the-rest (OvR): get the decision score from each classifier for the image and you select classifier that puts out highest score
- one-versus-one (OvO): a binary classifier for every pair of digits.



linear logistic

LOGISTIC REGRESSION

- commonly use cross entropy loss.
- gradient descent is an optimization algorithm for iteratively updating the weights to minimize the loss function.

REGULARIZATION

- to avoid overfitting, we use a regularization term added to the objective function
- $R(\theta)$ penalizes large weights. A setting of the weights that matches the training data perfectly but uses many weights with high values will be penalized more than a setting that matches the data a little less well, but w/ smaller weights.
- L2 regularization is a quadratic function of the weight values. It uses the square of the L2 norm of the weight values (Euclidean Distance)
- L1 regularization is a linear function of the weight values. It takes the sum of the absolute values of the weights, or Manhattan Distance.

* L1 regularization is called Lasso Regression and L2 regularization is called Ridge Regression.

ENSEMBLE LEARNING & RANDOM FORESTS (CHAPTER 7)

- a group of predictors is called an ensemble, and thus we have ensemble learning. An ensemble learning algorithm is called an Ensemble method.

VOTING CLASSIFIERS

- a simple way to create a good classifier is to aggregate the predictions of each classifier and predict the the class that gets the most votes. This majority-vote classifier is called a hard-voting classifier.
- the voting classifier often achieves a higher accuracy than the best classifier in the ensemble.
- even if each classifier is a weak learner (does slightly better than random guessing), the ensemble can be a strong learner given that there are enough weak learners and they are diverse enough.
- ensemble methods work best when the predictors are as independent from one another as possible.
- soft voting - if all classifiers are able to estimate class probabilities and you predict the class with the highest class probability, averaged over all individual classifiers.

BAGGING & PASTING

- using the same training algorithm for every predictor and training them on different random subsets of the training set.
- when sampling is performed with replacement, this method is bagging (bootstrap aggregating).
- when sampling is performed without replacement, this method is called pasting
- Both bagging and pasting allow training instances to be sampled several times across multiple predictors, but only bagging allows training instances to be sampled several times for the same predictor.
- once all of the predictors are trained, the ensemble can make a prediction for a new instance by simply aggregating the predictions of all predictors.

RANDOM FORESTS

- a random forest is an ensemble of Decision trees, generally trained through the bagging method.
- random forest algorithm introduces extra randomness when growing trees.
  - instead of searching for the best feature when splitting a node, it searches for the best feature among a random subset of features.

BOOSTING

- boosting refers to any ensemble method that can combine several weak learners into a strong learner.
- general idea is to train predictors sequentially, each trying to correct its predecessor.

ADABOOST

- adaboost (adaptive boosting) corrects its predecessor by paying a bit more attention to the training instances that the predecessor underfitted.
- this results in new predictors focusing more and more on the hard cases.

GRADIENT BOOSTING

- gradient boosting also works by sequentially adding predictors to an ensemble.
- However, instead of tweaking the instance weights at every iteration, GB tries to fit the new predictor to the residual errors made by the previous predictor.

STACKING (STACKED GENERALIZATION)

- instead of using trivial functions like hard voting, to aggregate the predictions of all predictors in an ensemble, train a model to perform this aggregation.

16 SUPPORT VECTOR MACHINE (SVM)

- think of an SVM classifier as fitting the widest possible street between the classes. This is called large margin classification.

SOFT MARGIN CLASSIFICATION

- if we strictly impose that all instances must be off the street and on the right side, this is called hard margin classification.
- hard margin classification only works if the data is linearly seperable and is sensitive to outliers.
- to avoid these issues, we use a more flexible model. We want to limit margin violations (instances that end up in the middle of the street or on the wrong side) this is soft margin classification.

POLYNOMIAL KERNEL

- not all datasets are linearly seperable, so SVM handling non linear datasets can be made possible by adding more features like polynomial features.
- use the kernel trick to get the same result as if you added many polynomial features, without actually having to add them, by training the SVM for higher degrees.

SIMILARITY FEATURES:

- add features computed using a similarity function which measures how much each instance resembles a particular landmark.

SVM REGRESSION:

- SVM supports classification and regression. For regression instead of fitting the largest street possible between two classes, SVM regression tries to fit as many instances as possible on the street while limiting margin violations.

- use lagrangian multipliers to find minima and maxima
- Mercers theorem: there exists a function ϕ that maps a and b into another space (possibly with higher dimensions). Function must be continuous and symmetric. this is why we can use the kernel trick.

- we use kernel functions to seperate non-linear regions. Quadratic programming to avoid local minima.
- SVM algo: maximize the street width by reducing # of nonzero weights to just a few, that correspond to important features that matter to the separating hyperplane. These non zero weights are the support vectors.

- moving a support vector moves the decision boundary, but moving the other vectors has no effect.
- form of equation defining hyperplane:

$$w^T x + b = 0 \quad \text{where } w \text{ is a weight vector, } x \text{ is an input vector, and } b \text{ is the bias.}$$

$$w^T x + b \geq 0 \quad \text{for } d_i = +1$$

$$w^T x + b < 0 \quad \text{for } d_i = -1$$

- the optimal hyperplane has the greatest margin of separation.
- distance between the hyperplanes (2) is $\frac{2}{\|w\|}$, $\|w\| = \sqrt{w_1^2 + w_2^2 + \dots + w_n^2}$
- $\max \frac{2|k|}{\|w\|}$ s.t. $(w \cdot x + b) \geq k$ for class 1, $(w \cdot x + b) \leq -k$ for class 2 ($w \cdot x + b = -k$ and $w \cdot x + b = k$)
- THE HYPERPLANE IS BETWEEN THE SUPPORT VECTORS!

★ i think SVM uses OVO approach to multiclass classification but prof says OVA.

CHAPTER 8: NEURAL NETWORK

(MITCHELL)(CH.4)

- delta rule is a G.D. learning rule for updating the weights in an ~~ANN~~ single layer neural network

- $\nabla F(w)$ = partial derivatives of cost function (pg. 89)
- training rule for gradient descent: $\vec{w} = \vec{w} + \Delta\vec{w}$ where $\Delta\vec{w} = -\eta \nabla E(\vec{w})$
- the learning rate determines the step size in the GD search.
- $E(\vec{w}) = \frac{1}{2}\sum_i (t_d - o_d)^2$ where t_d is the target and o_d is output
- $\frac{\partial E}{\partial w_i} = \sum_i (t_d - o_d)(-x_{id}) \Longrightarrow \Delta w_i = \eta \sum_i (t_d - o_d) x_{id}$
- This is all for the delta rule, which considers the task of training an unthresholded perceptron (a linear unit) for which the output o is given by: $o(\vec{x}) = \vec{w} \cdot \vec{x}$
- perceptron training rule: $o(x_1, \dots x_n) = \begin{cases} 1 \text{ if } w_0 + w_1 x_1 + \dots + w_n x_n > 0 \\ -1 \text{ if otherwise} \end{cases}$
  - a single perceptron can be used to represent many boolean functions
  - all primitive Boolean functions AND, OR, NAND, ¬AND, NOR, ¬OR can be represented by perceptrons
    - ★ any boolean function can be implemented by using a 2-level combination of these primitives (XOR cannot, we need a network of them)
  - FOR GD: the error surface contains a single global minimum, so the algo will converge to a weight vector with minimum error, given a sufficiently small η.
    - converging to a local min can be slow & if there are multiple local minima in the error surface, there is no guarentee G.D will reach global minimum.
  - for Perceptron: examples are linearly separable, and small learning rate: will succeed.

convergence properties: perceptron vs Δ rule

- Gradient vs Stochastic: G.D. computes weight updates after summing over all training examples. S.G.D approximates G.D. by updating weights incrementally, following the calculation of the error for each individual example. ✓ delta rule
  - update according to: $\Delta w = \eta (t - o) x_i$ where t is target value and o is unit output. (t is predicted value w/ current weights).

★ In cases where there are multiple local minima w.r.t. $E(\vec{w})$, S.G.D can sometimes avoid falling into these local minima because it uses variur $\nabla E_d(\vec{w})$ rather than $\nabla E(\vec{w})$ to guide its search (pg. 94)

- $o(\vec{x}) = \vec{w} \cdot \vec{x}$ delta rule; o refers to linear unit output
- $o(\vec{x}) = \text{sgn}(\vec{w} \cdot \vec{x})$ perceptron; o refers to threshold output

SUMMARY: perceptron training rule updates weights based on the error in the thresholded perceptron output, whereas delta rule updates weights based on the error in the unthresholded linear combination of inputs.

SIGMOID: also called logistic function. output is a nonlinear function of its inputs and differentiable (unlike perceptron) so it is suitable for gradient descent. threshold is a continuous function of input.

$o = \sigma(\vec{w} \cdot \vec{x})$, $\sigma(y) = \frac{1}{1 + e^{-y}}$, $\frac{d\sigma(y)}{dy} = \sigma(y) \cdot (1 - \sigma(y))$

- Backpropogation algorithm learns the weights for a multilayer network, given a network of a fixed set of units and interconnections.
  - employs gradient descent to attempt to minimize squared error between network output values and target values for these outputs.
  - basically just iteratively doing gradient descent backwards through the network and updating all of the weights based on a data point ∴

... derivation of backpropogation rules ...

CONVERGENCE & LOCAL MINIMA: backprop uses gradient descent so BP over a multilayer network is only guaranteed to converge toward some local minimum in E and not necessarily to the global minimum error.
- common heuristics to alleviate local minima: stochastic gradient descent, training multiple versions of network with different weights.

Representational Power of Feedforward Networks: boolean functions, continuous functions, arbitrary functions.

OVERFITTING: overfitting and fluctuations in testing error occur because the weights are being tuned to fit idiosyncrasies of the training examples that are not representative of the general distribution of examples.
- large number of weights provide many degrees of freedom that can fit noise.
- solutions: decrease each weight by some small factor during each iteration OR/AND have a validation set.

Quiz 8:

① given $g(x)$ is a sigmoid activation function, these are true:
- derivative of $g(x)$ is $g(x)(1-g(x))$
- sigmoid can be replaced by other activation functions such as ReLu and tanh. Different activation functions have diff. impacts on performance
- w/ sigmoid function, initial parameters must be small during training

② - ANN can be used for either regression or classification ✓
- in classification ANN, output layer may use sigmoid (single class) or softmax (multiclass) as activation function ✓
- stochastic G.D. is helpful in reducing local minima ✓
- same cost function for classification & regression ANN X

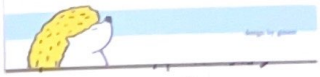
③ if target function is boolean or continuous, we need at most two hidden layers to represent it.
- TRUE

CHAPTER 7: BAYESIAN LEARNING

(MITCHELL CH. 6)

- $P(h|D) = \frac{P(D|h)P(h)}{P(D)}$ • $p(h)$ = prior probability of hypothesis h. may reflect any background knowledge we have about the chance of h being a correct hypothesis.
  - $P(D)$ = prior probability of training data D i.e the probability of D given no knowledge about which hypothesis holds. $P(D|h)$ = probability of D given h.
  - $P(h|D)$ = probability of h given D. This is posterior probability because it reflects our confidence that h holds after seeing training data D.

ln(p) is a monotonic function of p. Therefore, maximizing ln(p) also maximizes p ☺



thm of total probability ✓

BASIC PROBABILITY:

product rule: $P(A \wedge B) = P(A|B)P(B) = P(B|A)P(A)$

sum rule: $P(A \vee B) = P(A) + P(B) - P(A \wedge B)$

- given the training data, we want the most probable hypothesis (maximum a posteriori hypothesis)

$$h_{MAP} = \underset{h \in H}{\operatorname{argmax}}\ P(h|D)$$

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \qquad P(B) = P(B|A)P(A) + P(B|\neg A)P(\neg A)$$

$$P(c|+) = \frac{P(+|c)P(c)}{P(+|c)P(c) + P(+|\neg c)P(\neg c)} \Rightarrow P(c|+) = \frac{(.98)(.008)}{(.98)(.008) + (.03)(.992)} = \frac{.00784}{.00784 + .02976}$$

- a hypothesis that minimizes the sum of squared errors is also a maximum likelihood hypothesis:

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}}\ p(D|h) \quad \text{(lowercase } p \text{ is probability density and } D = \langle d_1 \dots d_m \rangle$$

$d_i = f(x_i) + e_i$ where e_i is noise. $\Rightarrow h_{ML} = \underset{h \in H}{\operatorname{argmax}} \prod_{i=1}^{m} p(d_i|h) = \underset{h \in H}{\operatorname{argmax}} \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-1}{2\sigma^2}(d_i - h(x_i))^2}$

(page 166):

*we are writing the expression for the probability of d_i given that h is the correct description of the target function $f \Rightarrow \mu = f(x_i) = h(x_i)$ ... take ln transformation ... maximizing negative quantity is equivalent to minimizing positive quantity $\rightarrow$

$$h_{ML} = \underset{h \in H}{\operatorname{argmin}} \sum_{i=1}^{m} (d_i - h(x_i))^2 \quad (6.6)$$

- equation 6.6 shows that the maximum likelihood hypothesis h_{ML} is one that minimizes the sum of the squared errors between observed training values d_i and hypothesis predictions $h(x_i)$

BAYES OPTIMAL CLASSIFICATION:

$$\underset{v_j}{\operatorname{argmax}} \sum_{h_i} P(v_j|h_i)P(h_i|D) \Rightarrow$$

$V = \{\oplus, \ominus\}$

$P(h_1|D) = .4$, $P(\ominus|h_1) = 0$, $P(\oplus|h_1) = 1$

$P(h_2|D) = .3$, $P(\ominus|h_2) = 1$, $P(\oplus|h_2) = 0$

$P(h_3|D) = .3$, $P(\ominus|h_3) = 1$, $P(\oplus|h_3) = 0$

$\sum_{h_i} P(\oplus|h_i)P(h_i|D) = .4$ and $\sum_{h_i} P(\ominus|h_i)P(h_i|D) = .6$ and $\underset{v_j \in \{\oplus,\ominus\}}{\operatorname{argmax}} \sum_{h_i} P(v_j|h_i)P(h_i|D) = \ominus$

- no other classification method using the same hypothesis space and same prior knowledge can outperform this method on average.

GIBBS ALGORITHM: bayes can be costly, so less optimal is Gibbs.

1) choose a hypothesis h from H at random, according to posterior probability distribution over H.
2) use h to predict the classification of the next instance x.

NAIVE BAYES CLASSIFIER: $v_{NB} = \text{argmax } P(v_j) \prod P(a_i | v_j)$

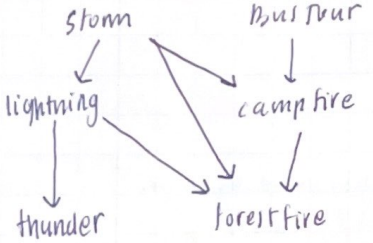
- method involves a learning step in which various $P(v_j)$ and $P(a_i|v_j)$ terms are estimated, based on their frequencies over the training data.

using tennis example: $v_{NB} = \text{argmax } P(v_j)\ P(\text{outlook} = \text{sunny} | v_j)\ P(\text{Temp} = \text{cool} | v_j)$
$P(\text{humidity} = \text{high} | v_j)\ P(\text{wind} = \text{strong} | v_j)$

$P(\text{play} = \text{yes}) = 9/14$
$P(\text{play} = \text{no}) = 5/14$
$P(\text{wind} = \text{strong} | \text{play} = \text{yes}) = 3/9$
$P(\text{wind} = \text{strong} | \text{play} = \text{no}) = 3/5$

$\Rightarrow P(\text{yes})\ P(\text{sunny}|\text{yes})\ P(\text{cool}|\text{yes})\ P(\text{high}|\text{yes})\ P(\text{strong}|\text{yes}) = .0053$
$P(\text{no})\ P(\text{sunny}|\text{no})\ P(\text{cool}|\text{no})\ P(\text{high}|\text{no})\ P(\text{strong}|\text{no}) = .0206$

Bayesian belief networks specify a set of conditional independence assumptions



* once we know values for storm and bus tour, we don't need any more info for campfire. lightning and thunder are independent of campfire.

QUIZ 7:

① $P(AB) = P(A)\,P(B)$

② use EM algo to train BBN. Em is dependent on unobserved variables
- NBC makes a simplifying assumption that the attribute values are conditionally independent given the target value. (?)

③ $P(h1) = 0.45$, $P(h2) = 0.3$, $P(h3) = 0.25$
$h1(x) = \text{yes}$ $h2(x) = \text{no}$ $h3(x) = \text{no}$
What is predicted result of Bayes optimal classifier? NO

1:30
2:30

CHAPTER 5: BV & ENSEMBLE LEARNING ... CONTINUED

(GERON CH. 7)

- **Bias problem**: wrong assumptions. A high-bias model is going to underfit training data
- **Variance prob**: models excessive sensitivity to small variations in the training data. A model w/ many degrees of freedom is going to have high variance and overfit the training data.

Soft output: classifier outputs a value representing the confidence of this sample belonging to each class (probability)

hard output: a classifier outputs only one class label.

SOFT:
- simple summary function calculates the average, minimum, maximum, and product for each class
- weighted average is when you multiply the probabilities by their weights (of each classifier) to get weighted average of each class.

HARD:
- majority vote is when you count the frequency that classifiers selected a class.
- weighted majority vote is the same, but multiply each probability with respective classifier weights.
- Naive Bayes: outputs of classifiers are 1, 2, 1

EXAMPLE ① OF NB:

$\hat{P}(w_1 | d_{1,1}(x)=1) = 40/70$ $\quad \hat{P}(w_2 | d_{1,1}(x)=1) = 30/70$

$\hat{P}(w_1 | d_{2,2}(x)=1) = 30/60$ $\quad \hat{P}(w_2 | d_{2,2}(x)=1) = 30/60$

$\hat{P}(w_1 | d_{3,1}(x)=1) = 50/90$ $\quad \hat{P}(w_2 | d_{3,1}(x)=1) = 40/90$

CL(1) decision

	C1	C2
C1	40	10
C2	30	20

$\hat{P}(w_1 | d_{1,1}(x)=1)$: probability of seeing class 1 given that classifier 1 chose class 1

$\hat{P}(w_2 | d_{1,1}(x)=1)$: probability of seeing class 2 given that classifier 1 chose 1

CL(2) Decision

	C1	C2
C1	20	30
C2	20	30

EXAMPLE MV:

② classifier 1 outputs class 1
classifier 2 outputs class 2
classifier 3 outputs class 1

weights:
classifier 1: .3
classifier 2: .4
classifier 3: .3

CL(3) Decision

	C1	C2
C1	50	0
C2	40	10

Weighted majority: class 1 = .3 + .3 = .6
class 2 = .4 = .4
} PICK CLASS 1

- Bagging uses same training algo to produce diverse set of weak learners for ensemble learning
- stacking aggregates predictions of weak learners by training a model for aggregation, instead of using trivial functions (i.e. hard voting)
- most boosting methods produce weak learners by training them sequentially i.e by correcting the predecessor of each weak learner at each iteration of training

CHAPTER 4: DECISION TREE LEARNING

(GERON CH. 6)

- Gini impurity measures the frequency at which any element of the dataset will be mislabelled when it is randomly labeled.

  Gini Index = $1 - \sum_j p_j^2$, where p_j is the probability of class j.

  - min value of Gini Index is 0. This happens when the node is pure i.e. all the contained elements in the node are of one unique class
    - Node will not split again when pure, thus, optimum split is chosen by features with smaller Gini index.
  - max value when probability of two classes are the same.
- Entropy is a measure of information that indicates the disorder of the features with the target.
- Gini impurity vs. Entropy: 1) Gini is slightly faster
  2) Entropy tends to produce slightly more balanced trees
  3) most of the time they lead to similar trees.
- For trees based on information gain and Entropy, split on highest information gain.
- CART Training algorithm: (classification and regression tree) used to train decision trees. Algo first splits training set into two subsets using a single feature k and a threshold t_k (petal length and petal length ≤ 2.45 cm)
  - chooses pair (k, t_k) that produces the purest subsets (weighted by size)
  - CART cost function: $J(k, t_k) = \frac{m_{left}}{m} G_{left} + \frac{m_{right}}{m} G_{right}$

    where { $G_{left/right}$ measures the impurity of the left/right subset; $m_{left/right}$ is the number of instances in the left/right subset }
  - stops when reaches maximum depth or cannot reduce impurity.
- ID3 can produce decision trees with nodes that have more than two children.
- CART vs ID3: CART uses gini impurity while ID3 uses entropy.
- OVERFITTING IN DECISION TREE LEARNING:
  - bigger trees are more likely to overfit
  - reduce overfitting: 1) stop growing the tree when data split is not statistically significant
    2) grow a full tree and then prune it.
- CART cost function for regression:

  $J(k, t_k) = \frac{m_{left}}{m} MSE_{left} + \frac{m_{right}}{m} MSE_{right}$ where { $MSE_{node} = \sum (\hat{y} - y^i)^2$; $\hat{y}_{node} = \frac{1}{m_{node}} \sum y^i$ }
- CHI-SQUARE: sum of squared differences between observed and expected frequencies of target variable for each node

  Chi-square = $\sqrt{(actual - exp)^2} / Exp$
- a decision tree represents a disjunction of conjunctions of constraints on the attribute values of instances
- decision trees are used for both classification and regression.
- all decision trees have a complete space of finite discrete-valued functions

CHAPTER 2: TRAINING MODELS

(GERON 4)

- MSE cost function for Linear Regression:
  $MSE(X, h_\theta) = \frac{1}{m} \sum_{i=1}^{m} (\theta^T x^i - y^i)^2$
- to find the value of θ that minimizes the cost function, we have a closed-form solution which is the Normal Equation: $\hat{\theta} = (X^T X)^{-1} X^T y$
- Batch gradient descent uses the whole training set to compute the gradients at every step
- Stochastic gradient descent picks a random instance in the training set at every step and computes the gradients based only on that single instance.
  - when the cost function is very irregular (SGD) the algorithm may jump out of local minima, so there is a better chance of finding the global minimum compared to BGD.
  - randomness is good for escaping local minima, but bad for settling at the minimum.
- Mini Batch Gradient Descent computes the gradients on small random sets of instances called mini-batches.
- Elastic Net is a middle ground between Ridge Regression and lasso regression. The regularization term is a simple mix of both L2 and L1 regularization terms.
  - mix ratio is r. $r = 0$ elastic net is ridge and when $r = 1$ elastic net is lasso
- in order to reduce overfitting, we can regularize the model (i.e. constrain it).
- the fewer degrees of freedom, the harder it is to overfit the data.
- use ridge/lasso regression or elastic net to constrain weights.

LOGISTIC REGRESSION:

- just like linear, but outputs the logistic (sigmoid) of the result: $\hat{p} = h_\theta(x) = \sigma(x^T \theta)$
  where $\sigma(t) = \frac{1}{1 + \exp(-t)}$, $\hat{y} = \begin{cases} 0 \text{ if } \hat{p} < 0.5 \\ 1 \text{ if } \hat{p} \geq 0.5 \end{cases}$
- cost function of a single training instance: $c(\theta) = \begin{cases} -\log(\hat{p}) & \text{if } y = 1 \\ -\log(1 - \hat{p}) & \text{if } y = 0 \end{cases}$ (pg. 144)
- the cost function over the whole training set is the average cost over all training instances, called log-loss:
  $J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} [y^i \log(\hat{p}^i) + (1 - y^i) \log(1 - \hat{p}^i)]$
- There is no closed form solution (Normal equation) to minimize cost function, but cost function is convex, so G.D. guarenteed to find global minimum.
- Basis function converts non linear model to linear model.