Kobee Raveendran

**CAP5610 Midterm Study Guide**

Example questions:

*True or false: Assuming that the training set is the same, the cluster means found by k-means clustering will be the same across multiple repetitions of the algorithm.*

False; k-means clustering is initialized randomly, so although the means found in each run can be close or similar, they will most likely not be the exactly the same.

*Given the conditional probabilities of a set of features for a Naïve Bayes classifier, decide on a class label.*

|  |  |  |
| --- | --- | --- |
| Probability | Positive | Negative |
| P(Y) | 0.5 | 0.5 |
| P(small | Y) | 0.4 | 0.4 |
| P(medium | Y) | 0.1 | 0.2 |
| P(large | Y) | 0.5 | 0.4 |
| P(red | Y) | 0.9 | 0.3 |
| P(blue | Y) | 0.05 | 0.3 |
| P(green | Y) | 0.05 | 0.4 |
| P(square | Y) | 0.05 | 0.4 |
| P(triangle | Y) | 0.05 | 0.3 |
| P(circle | Y) | 0.9 | 0.3 |

Assign a class to a test instance with the following features: <medium, red, circle>

Note: P(X) calculation is ignored in the denominator since both terms will have it, so comparison is the same (only the values will be scaled by a factor of 1/P(X))

Positive class: P(Y = +) \* P(medium | Y = +) \* P(red | Y = +) \* P(circle | Y = +)

Positive probability: (0.5) \* (0.1) \* (0.9) \* (0.9) = 0.0405

Negative class: P(Y = -) \* P(medium | Y = -) \* P(red | Y = -) \* P(circle | Y = -)

Negative probability: (0.5) \* (0.2) \* (0.3) \* (0.3) = 0.009

Positive class probability is higher, so Y is likely to be positive.

**Lecture 1: Introduction**

* Feature vector: vector *X* (*X1, …, Xn*), where each element is an attribute that can (desirably) provide meaningful information toward a prediction (classification or regression)
* K-nearest neighbor algorithm:
  + For a test instance, find the distance to all other points (training instances)
  + Pick the k-nearest training instances
  + For the test instance, assign to it the class that the majority of the k-nearest neighbors have
* Distance functions:
  + Euclidean (L2 norm): straight-line distance, given by the square of the sum of component-wise squared differences of the pair of vectors
    - i.e. sqrt((q1 – p1)2 + (q2 – p2)2), for a feature vector with only 2 attributes (elements)
  + Manhattan (L1 norm): sum of absolute value of component-wise differences of a pair of feature vectors
    - i.e. |q1 – p1| + |q2 + p2|
* train/test split: split of dataset into two components; training set will be used to tune parameters or fit a model to a probability distribution, while test set will be used to evaluate a model performance-wise on data that it has not “seen” before (not used to tune parameters or fit during training)
* K-fold cross validation and Leave-One-Out cross validation:
  + K-fold CV: split dataset into K equal or nearly equally-sized partitions. Use each partition as a test set, leaving the other K – 1 partitions as the training sets, repeating K times
  + Leave-One-Out CV: extreme version of K-fold CV where K equals the number of elements in the dataset; that is, one element of the dataset is the training ‘set’, while all other elements serve as the training set
* Problems of overfitting: model becomes too accustomed to the training set and performs poorly on unseen data; it fails to generalize and provide accurate predictions for data it has not seen before -> poor classification accuracy or large regression error
* Curse of dimensionality: as the dimensionality of problem increases, exponentially more data is required; given some dataset, as dimensionality goes up, the data becomes increasingly sparse and it becomes difficult, if not impossible, to group data or draw any meaningful relationships between them. This is because the volume of a hypersphere (which could be used for distance related models like KNN) increases much more slowly as dimensionality *d* increases, compared to the volume of a hypercube (which represents the space that the data is in). As *d* approaches infinity, the volume of the hypersphere approaches 0, while the hypercube’s volume approaches infinity, meaning that 0 neighboring data will be available for a test sample, making it impossible to make predictions on it

**Lecture 2: Probability Theory**

* Probability: discrete and continuous
  + Discrete probability: categorical, can still hold an infinite number of values, but there is no “gray” area; must be of one value or the other, no in-betweens
  + Continuous: strictly numerical; can hold an infinite number of values, and can be in-between values
* Distributions: (see cheat sheet)
  + Note: Bernoulli distribution is a special case of binomial distribution in which n = 1; in other words, the binomial distribution is simply the probability of *k* successes on *n* Bernoulli trials
  + Gaussian distribution: the uniform distribution, where probabilities are highest as *x* approaches the mean and increasingly lower as *x* deviates from the mean
* Sample mean and variance:
  + Sample mean: sum of elements divided by number of elements
  + Sample variance: sum of (each element with mean subtracted from it squared), then divided by number of elements; standard deviation is the square root of this
* Conditional probability: probability of some event occurring given the probability of another event
  + Bayes rule: P(X | Y) = P(Y | X) P(X) / P(Y) = P(X, Y) / P(Y)
* Joint probability tables: for probability of event A and event B occurring, simple calculate: P(A, B) = P(A) \* P(B)
* Conditional independence: given A and B, and a third event C; knowledge of whether A occurs has no influence on whether B occurs, given only that event C occurs
  + This is just normal independence applied to an event with a condition
    - See <https://math.stackexchange.com/questions/23093/could-someone-explain-conditional-independence>
* MLE estimate for Bernoulli and univariate Guassian:
  + Gaussian MLEs:
    - Mean: same as sample mean
    - Variance: same as sample variance
  + Bernoulli:
    - p = sample mean
* MAP: psuedocounts and how it relates to MAP
  + Pseudocounts are a constant that can be added to the prior probability to avoid cases where a dataset omits a class entirely; for each class, it adds a constant alpha divided by the sum of the alphas of all classes. This allows a classifier using MAP to still be capable of assigning predictions to a class even if the class just happens to not be in the dataset, because it is expected. Higher pseudocounts make the prior affect the final prediction more.

**Lecture 3 and 4: Bayesian Classifiers**

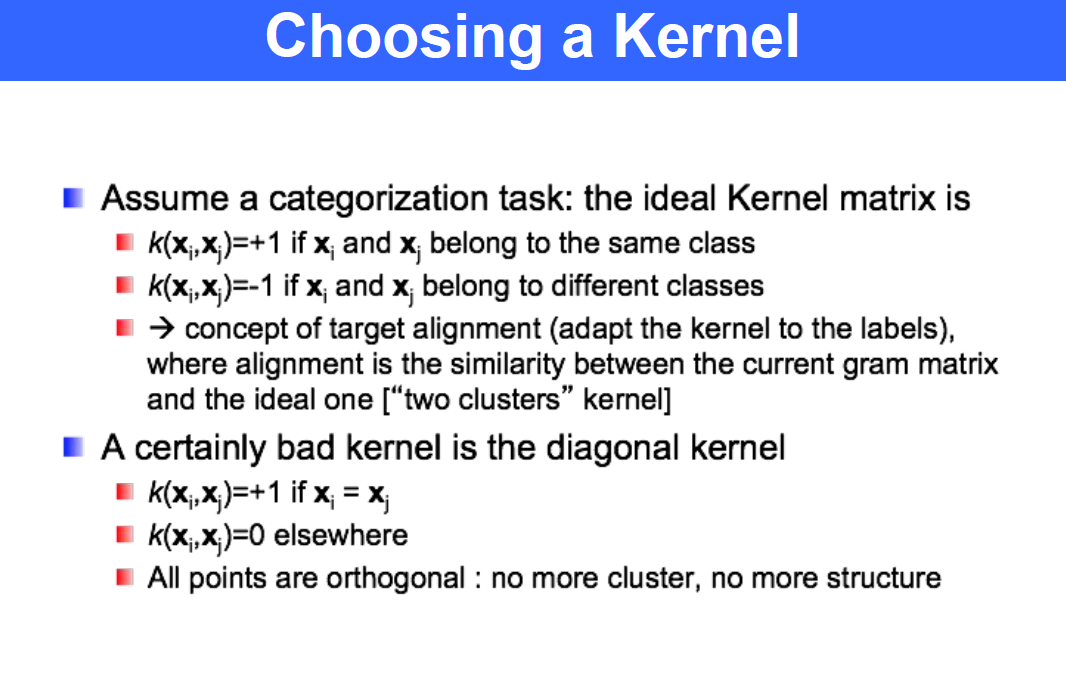
* Prior, class conditional, and posterior probability distributions:
  + Prior: probability distribution using expected probabilities before using information derived from the data empirically
    - *Prior* to evidence being viewed
    - Can be thought of as the fraction of a particular class in a dataset
  + Class conditional: probability distribution of one random variable given the set value of another random variable
    - Can be thought of as the distribution from which we can draw an example for the given class
  + Posterior probability distribution: combination of the prior probability and a likelihood function (which contains the information from evidence); prior probability + likelihood function
    - *Posterior* or *after* evidence has been accounted for
    - Given an input vector X, what’s the probability of a particular class?
* MAP decision rule: choose the class with the maximum probability given input vector X (prob of class conditioned on input vector X; aka class conditional probability times prior probability of getting that class)
  + i.e. argmax( P(Y = F | *X*1, *X*2), P(Y = ¬F | *X*1, *X*2) ) for determining if someone has the flu or not
  + the MAP decision rule gives the lowest possible classification error of any classifier (Bayes error)
* likelihood ratio: uses MAP rule
  + f(*X*) = P (Y = F | *X*1, *X*2) / P (Y = ¬F | *X*1, *X*2)
  + if f(*X*) > 1, belongs to class F, otherwise belongs to class ¬F
* definition of Bayes error: minimal error that is made by the MAP decision rule, and is the lowest bound of error rate that can be achieved by any classifier
  + min {*p* (*Y* = *C*1 | *X*), *p* (*Y* = *C*2 | *X*)}
* Naïve Bayes classifier example: see above
  + Process: calculate conditional probability of obtaining each class based on features: P(+|X) = P(+) \* P(X | +) / P(X) [ though denominator isn’t needed for just predictions]
* Univariate Gaussian class conditional density and special case (equal covariance matrix): see formula sheet
  + Special case: equal covariance matrices 🡪 f(*X*) = *XW* + b
* True/False positives, True/False negatives, Precision, recall:
  + Precision: TP / (TP + FP)
  + Recall: TP / P or equivalently TP / (TP + FN) (aka the true positive rate)
  + True positive: classifier correctly predicts positive class
  + False positive: classifier incorrectly predicts a positive class
  + True negative: classifier classifies a negative class as negative
  + False negative: classifier incorrectly predicts a negative class

**Lecture 5: Logistic/Linear Regression**

* Difference between discriminative and generative models:
  + Discriminative: model can only differentiate classes from each other
  + Generative: model can both differentiate classes from each other as well as use probability distributions to generate new data points (i.e. can generate feature vector belonging to a given class *C* by sampling that class’s probability distribution)
* Logistic function for P (Y = 1 | X): see cheat sheet
* Multiclass model: P (Y = *c* | X): see cheat sheet
* Role of regularizer to avoid overfitting: regularizer keeps weights closer to 0 by imposing prior knowledge on the weights
* Comparison between Naïve Bayes and logistic regression (parameter reduction):
  + Naïve Bayes: 4*n* + 2 params for a binary classifier, where *n* = num classes; 2 for mean and variance per attribute, and 2*n* for class prior distributions
    - Also, assumes conditional independence given a class label
  + Logistic regression: *n* weight coefficients, where *n* = num classes
    - No independence assumptions
* Idea behind bias-variance decomposition: there is an inherent tradeoff between bias and variance
  + Bias: deviation of the average predictor from the optimal one (can think of it as an error)
  + Variance: how stable the learned predictors are over different training sets
  + High bias = low variance
  + Low bias = high variance
  + High bias is associated with underfitting (failure to determine relationships/trends between feature vectors and a class), and high variance is associated with overfitting (failure to generalize to unseen data)
  + Flexible model can have small bias but large variance (can capture and adjust to subtle changes in training set, lead to overfitting)
  + Rigid model has smaller variance but larger bias (stronger prior limits the variance, but makes it less flexible and prone to high bias and error, lead to underfitting)
* Difference between logistic and linear regression:
  + Logistic regression: used for discrete problems (classification); logistic loss function causes errors to be penalized toward an asymptotic constant
  + Linear regression: used for continuous problems (regression); least squares (LSE) error function minimizes least squares error so large errors are penalized quadratically
* Pseudoinverse for finding weights that minimize least squares error between XW and Y: <https://stats.stackexchange.com/a/361614/238279>

**Lecture 6 and 7: Support Vector Machines**

* support vectors: the point(s) of each class that are the hardest to classify, lie closest to the decision hyperplane, and for which the Lagrange multiplier is non-zero
* margin: the “strip” surrounding the decision hyperplane for both classes; the margins are optimized so that the number of points needing non-zero weights (the points that matter the most for making a decision) are as low as possible
  + the margin is essentially the width that the boundary could be increased by before hitting a data point
* hard margin: no training error, all training examples classified correctly (no room for mistakes)
* soft margin: allows some misclassification by introducing a slack variable
* kernel trick: rather than mapping the features into a higher dimensional space manually, since dot product is all we need, we instead use a kernel function *K* that directly represents the dot product in this higher dimensional space, eliminating the need for feature mapping (which is time-consuming)
* Types of kernels and choosing kernels:
  + Radial basis function (RBF, aka Gaussian): creates nonlinear combinations of features in such a way that they can then be separated by a hyperplane (linearly); performs well in higher-dimensional, complex data
  + Linear kernel: for data that is already linearly separable, is able to draw a decision line; performs well on low dimensional, linearly separable data
  + Polynomial kernel: middle ground between RBF and linear; attempts to draw a decision boundary using a polynomial function of degree *p* (which is a hyperparameter)
  + Sigmoid: typically used in neural networks and in logistic regression as an activation function, but can also work on some data (like the concentric circles toy dataset); tricky to deal with since it has hyperbolic properties in its decision boundaries (boundaries tend toward corners, which often isn’t desirable)
  + Choosing a kernel:



* SVM multiclass:
  + One-vs-One: train N (N – 1) / 2 SVM classifiers on combinations of pairs of classes from the dataset; at inference time, make each classifier vote and choose the class that has the most votes
  + One-vs-Rest: train N SVM classifiers on class *i* against all other classes; at inference time, pick the class of the classifier that reports the highest confidence (furthest from the margin)
* Weighted SVMs: for imbalanced datasets, scale the less-represented classes up and more-represented classes down to simulate balance; each class has weight n\_samples / (n\_classes \* bincount(class\_i))

**Lecture 8 and 9: Neural Networks and CNNs**

* Neural network structure: series of layers of neurons
  + Neuron: takes an input vector X, activates neuron using WX + b (weighted sum of input features), passes this through activation function to the next layer
  + Consists of an input layer (layer 1), hidden layer(s), and an output layer (layer N)
* Stochastic gradient descent: gradient descent applied to only one training example
  + Only compute the gradient of the squared error for one example at a time
* Backpropagation:
* Momentum: some constant added to weights during update stage to avoid getting stuck at steep but bad local optima (a potentially better optimum may be available)
* Purpose of regularized squared error: reduce generalization error but not its training error
  + add zero mean Gaussian prior on the weights
* Neural networks vs. SVMs:
  + Neural networks use a nonlinear function *f*(*x*) in order to draw complex boundaries, without changing the data
  + SVMs instead use a linear function, but have to change the data in a way that lets the decision boundary be linear
* Advantages of deep networks:
  + Can provide sufficient complexity depending on the complexity of the problem (simpler classifiers may oversimplify it)
    - Shallow structures require exponentially more neurons, but deep structure can represent the same function with many less neurons in a compact way
    - Think of the difference between a linear combination and a composition of functions (i.e. g1( g2( g3( …(x)…)))
* Autoencoder structure:
  + Input layer, hidden (bottleneck) layer, and output layer
  + Outputs of previous layer become inputs to current layer
  + Hidden layers serve as feature representations, and can reconstruct the input in the output layer
    - Allows for unsupervised learning
* Convolution and max-pooling operations:
  + Convolution: sliding-window dot product, with sliding set by stride
    - Results in fewer parameters than an FCN due to shared weights
  + Max pooling: same sliding-window concept with sliding set by stride, but now simply choose the maximum element in each *n* x *n* window as the current cell’s value