Section 11: Classification and Clustering

CS 182 - Artificial Intelligence

Algorithms in machine learning fall under **supervised** and **unsupervised** approaches. In the former, we are given a set of inputs and labels/outputs, and would like to learn the relationship between them to guess label/output values for new data. In the latter, we are only given input values, and aim to find structure within them.

Supervised Learning

Supervised learning can also be broken down into **parametric** and **non-parametric** approaches. Parametric algorithms, such as Naive Bayes or regression, rely on learning a set of parameters that define the classifier. Non-parametric algorithms, such as k-NN, rely directly on data rather than on a parameterized model.

Naive Bayes

Naive Bayes can be thought of as a **probabilistic inference algorithm** over a special case of a Bayes Net that is applied to classification. The model is made of an unknown variable Y representing the possible classes, which causes a collection of emission variables E_1, E_2, ..., E_n. We make a ("naive") assumption that each of the emissions is conditionally independent of one another given Y. While this may not be true for most applications, it still results in a simple and useful algorithm. Finally, given the emissions, we can calculate classification probabilities by inference over the Bayes Net:

$$P(Y|x_1, \ldots, x_n) \propto P(Y, x_1, \ldots, x_n) = P(Y) \prod_{i=1}^{n} P(x_i|Y)$$

To build up this network from labeled data, we can count the frequencies of occurrence of observing (x_i, Y) for each x_i and normalize to construct conditional probabilities. Improvements on this approach involve **smoothing**, which is discussed further down.

Regression

When the output variable y is continuous rather than discrete, we can perform regression. First, a **model** h_θ is chosen to describe the input-output relationship, where the **model parameters** θ are calculated by optimizing over a **loss function** L on the training data. For instance, if we assume that the underlying model is **linear**, we choose $h_\theta(f^{(i)}) = \theta^T f^{(i)}$ for every ith data vector. Often, a quadratic loss function is chosen. This is both intuitive, but also theoretically supported – a quadratic is the **maximum likelihood**

estimate of the data given the model parameters, under the assumption of i.i.d. Gaussian noise in the data. Overall, the optimization problem to solve **linear regression** would be:

$$\min_{\theta} \sum_{i=1}^{m} (\theta^T f^{(i)} - y^{(i)})^2$$

There are other possible loss functions, such as:

- **absolute (L_1) loss:** $L = |h_\theta(f) - y|$
- **deadband loss:** $L = \max(0, |h_\theta(f) - y| - \epsilon)$

Logistic regression is a variant of regression designed for binary classification (where the output variable does assume discrete values 0 and 1). Instead of using the linear function $h_\theta(f) = \theta^T f$, we apply a sigmoid:

$$h_\theta = \frac{1}{1 + e^{-\theta^T f}}$$

Similarly to linear regression, we choose a loss function, pose an optimization problem, and solve for the optimal parameters θ.

While linear regression with a quadratic loss is **convex** and has a closed-form solution, most of these optimization problems are non-convex and are solved with **stochastic gradient descent**. Finally, after an optimization approach is applied and the model parameters θ are determined, we can guess the output of new data f by calculating $h_\theta(f)$ (for the binary logistic case, rounding it to 0 or 1).

k-Nearest Neighbors

This is a simple **non-parametric** classification approach which assigns a label to a new point by choosing the most common class of the k closest neighboring points. Variations on this algorithm might give a weighting scheme to the k labels rather than choosing the most common value.

When designing an ML algorithm, the available data is usually broken up into the **training data**, which is used to train the classifier, and the **test data**, which is used to evaluate the classifier and can in no way be viewed or used beforehand. This division of datasets allows to deal with two issues:

1. **Overfitting** occurs when a classifier is trained for very high (near 100%) training accuracy, but is not able to **generalize** to the testing data, and has low test accuracy.

   (a) **Smoothing** is used in discrete problems to cause more gradual updates to probabilities. For instance, one example of overfitting would be giving zero probabilities to all words not in the training set, which can be mitigated with **Laplace smoothing**:

   $$\text{standard probability: } P_{ML}(x) = \frac{count(x)}{N}$$
   $$\text{with smoothing: } P_{LAP,k}(x) = \frac{count(x) + k}{N + k|X|}$$

   (b) **Regularization** is used in continuous problems such as regression. For instance, an example of overfitting would be using a high-order polynomial that goes through all training data points.

The addition of a regularization term to the loss function places an analytic penalty on problem parameters to encourage smaller values or sparsity:

$$\text{standard loss function: } loss(f^{(i)}, y^{(i)}, \theta) = (\theta^T f^{(i)} - y^{(i)})^2$$
$$\text{with } L_2\text{-regularization: } loss(f^{(i)}, y^{(i)}, \theta) = (\theta^T f^{(i)} - y^{(i)})^2 + \lambda ||\theta||_2^2$$
$$\text{with } L_1\text{-regularization: } loss(f^{(i)}, y^{(i)}, \theta) = (\theta^T f^{(i)} - y^{(i)})^2 + \lambda ||\theta||_1$$

2. Sometimes algorithms depend on a choice of **hyperparameters**, which must be decided outside of the training process. There are a couple common approaches to tuning:
   (a) Part of the training data is taken to be the **held-out dataset**. For each hyperparameter value, the classifier is trained with the rest of the training data and tested on the held-out data, and the one with the highest accuracy is chosen.
   (b) If there are m possible values for a hyperparameter, **cross-validation** divides the training data into m sections. For the ith value, the ith section is chosen to be the tuning test set and the others are chosen to be the tuning training sets. After evaluating all hyperparameter values, the one with the highest tuning test accuracy is chosen.

Unsupervised Learning

Unsupervised learning algorithms focus primarily on clustering, dimensionality reduction, anomaly detection, or other ways of inferring data structure without labels. The algorithms we have discussed all fall under clustering.

K-means is a **centroid-based** clustering algorithm. First, initialize k cluster centroids. Then, iteratively:

Assign each point to nearest cluster centroid:

$$a_i = \text{argmin}_k \text{ dist}(x_i, c_k), \quad i = 1, \ldots, N$$

Recompute the kth cluster centroid by taking the mean of the points assigned to cluster k:

$$c_k = \frac{1}{|a_i \ : \ a_i = k|} \sum_{(i \, : \, a_i = k)} x_i, \quad k = 1, \ldots, K$$

Agglomerative clustering and **divisive clustering** are both **connectivity-based** clustering algorithms. The former iteratively constructs larger clusters from smaller clusters. The latter iteratively breaks down large clusters into smaller clusters.

Choosing K: The outcome of all of these algorithms vary on the chosen number of clusters. Some common approaches for choosing k are:

- We can plot the average of the distances from each point to its cluster's centroid. Then, it is reasonable to choose the smallest value of k for which this "error" flattens out.
- **Silhouette diagrams** are plots of the measure

  $$m_i = \frac{(b_i - a_i)}{\max(a_i, b_i)}$$

  where a_i is the mean distance from the point x_i to all points in its cluster, and b_i is the mean distance from x_i to all points in the neighboring ("next best") cluster.

Exercises:

1. **Silhouette Diagrams.**

   Suppose we are trying to cluster an unlabeled dataset. We apply k-means with $k = 3$, and are evaluating the clusters with a with a silhouette diagram.

   (a) Suppose that for a certain point x_i, $a_i << b_i$. What will be the approximate value of m_i, and what does this imply about the clustering of x_i?
   If a_i is small relative to b_i, this is a good clustering of x_i, since it is (on average) much closer to points in its own cluster than the next best cluster. The measure m_i will be close to 1.

   (b) Suppose that for a certain point x_i, $a_i \approx b_i$. What will be the approximate value of m_i, and what does this imply about the clustering of x_i?
   If a_i is similar to b_i, this is not a very good clustering of x_i, since it is about the same distance away from points in its own cluster as those in the next best cluster. The measure m_i will be close to 0.

   (c) Suppose that for a certain point x_i, $a_i >> b_i$. What will be the approximate value of m_i, and what does this imply about the clustering of x_i?
   If a_i is much greater than b_i, this is an incorrect clustering of x_i, since it is much farther from points in its own cluster as those in the next best cluster (this situation should be impossible with k-means). The measure m_i will be close to -1.

2. **Regression.**

   You are working for a hospital, and are given a dataset of patient information with hundreds of continuous features (height, weight, blood pressure, etc) along with whether the patient has diabetes. Your goal is to predict whether incoming patients will have diabetes.

   (a) Formalize the problem as an optimization problem using logistic regression.
   Let $f^{(i)}$ be the vector of features for the ith patient, $y^{(i)}$ be the true outcome (diabetes or no diabetes) for each patient, and θ be the parameters we are learning. Then, if we use a sum-of-squared-error loss function, we are trying to solve the problem:

$$\hat{\theta} = \text{argmin}_\theta \sum_i \left(\frac{1}{1 + e^{-\theta^T f^{(i)}}} - y^{(i)}\right)^2$$

   (b) Write down the formula your model would use to compute the probability that a patient has diabetes.
   Once the parameters θ are learned by solving the problem above, a new patient's probability of having diabetes given a feature vector f will be

$$P(y|f) = \frac{1}{1 + e^{-\theta^T f}}$$

   .

   (c) Suppose you are now trying to estimate blood pressure using linear regression on the 20 features you consider most important. If you calculate a regression model on a training set of 12 patients, what (approximately) will be your training set error?
   In this problem, we have 20 "unknowns" (the parameters θ and only 12 "equations" (the training data). Therefore, unless the training data contains equations which exactly contradict each other, this is an under-determined system, and we can choose θ such that all the total error is 0. This is an example of overfitting.

3. **Naive Bayes**

In this problem, we will be using Naive Bayes to predict the probability that a given email is spam or not spam (ham). We assume a model in which an underlying unobserved state y (spam/ham) creates emissions (words) $w_1, \ldots, w_n$. We use a bag-of-words approach to model text – the ordering of words within a sentence is ignored. Then, $P(Y)$ is the prior probability across states Y, and $p(w_i|Y)$ are the conditional probabilities of seeing a word w_i given Y. For the problems, start by assuming a uniform distribution over Y, and use the following conditional probability table over words:

	$p(w\|ham)$	$p(w\|spam)$
CS182	0.05	0.01
friend	0.02	0.04
need	0.01	0.01
on	0.02	0.02
our	0.02	0.02
project	0.007	0.003
start	0.01	0.03
to	0.01	0.01
you	0.01	0.015
we	0.01	0.015
working	0.01	0.03
...	...	...

(a) What is the probability of seeing the following spam email: "we need you on our project friend"?

$$p(email|spam) = 0.015 * 0.01 * 0.015 * 0.02 * 0.003 * 0.04 = 5.4 * 10^{-12}$$

(b) What is the probability that the following text is spam: "start CS182 project"?

$$p(email|ham) = 0.01 * 0.05 * 0.007 = 0.0000035$$
$$p(email|spam) = 0.03 * 0.01 * 0.003 = 0.0000009$$
$$p(spam|email) \propto 0.5 * 0.0000009$$
$$p(ham|email) \propto 0.5 * 0.0000035$$

We can normalize this and get a probability for $p(spam|email) = 0.205$.

(c) We now consider a fresh inbox for which we have no previous data. We want to build a new classifier from scratch. The inbox right now contains three emails:

spam: "I am only prince"

spam: "I will send money"

ham: "I will send you the review for it"

Determine the prior over Y (spam/ham), fill in the probability table below, and calculate the probabilities that the email "I will send" is spam/ham.

	freq ham	$p(w\|ham)$	freq spam	$p(w\|spam)$
am	0	0	1	1/8
for	1	1/8	0	0
I	1	1/8	2	2/8
it	1	1/8	0	0
money	0	0	1	1/8
only	0	0	1	1/8
prince	0	0	1	1/8
review	1	1/8	0	0
send	1	1/8	1	1/8
the	1	1/8	0	0
you	1	1/8	0	0
will	1	1/8	1	1/8

We can determine the prior by counting the number of emails from each category: $p(spam) = 2/3$ and $p(ham) = 1/3$. To construct the table, we count the word frequencies and normalize by the number of words. Using this table, we can take the same approach as before:

$$\begin{aligned} p(email|ham) &= (1/8)^3 = 0.00195 \\ p(email|spam) &= 2/8 * 1/8 * 1/8 = 0.0039 \\ p(ham|email) &\propto \frac{1}{3} * 0.00195 = 0.00065 \\ p(spam|email) &\propto \frac{2}{3} * 0.0039 = 0.0026 \end{aligned}$$

Normalizing this yields 0.2 and 0.8 probabilities for the two classes respectively.

(d) For long texts, you will need to multiply many numbers smaller than 1. This can cause implementation problems, where all of the predicted probabilities are so small that they are numerically rounded to zero. What can you do to address this issue? Why does it work?
In order to avoid those issues, you can add up the log-probabilities. This works because

$$P(Y)\prod_{i=1}^{n} P(w_i|Y) = e^{\log(P(Y)\prod_{i=1}^{n} P(w_i|Y))} = e^{\log P(Y)\sum_{i=1}^{n}\log P(w_i|Y)}$$

(e) Use your classifier to predict the emails "Will you send the review" and "Please send the review". Those two are special cases. Since $p(review|spam) = 0$), we have a 0 probability for the whole term. The word "please" does not occur in the table and with the current approach, we can't compute the probabilities.

(f) Repeat task (c) with laplace smoothing and $k = 1$.

	freq ham	$p(w\|ham)$	freq spam	$p(w\|spam)$
am	1	1/20	2	2/20
for	2	2/20	1	1/20
I	2	2/20	3	3/20
it	2	2/20	1	1/20
money	1	1/20	2	2/20
only	1	1/20	2	2/20
prince	1	1/20	2	2/20
review	2	2/20	1	1/20
send	2	2/20	2	2/20
the	2	2/20	1	1/20
you	2	2/20	1	1/20
will	2	2/20	2	2/20

$$
\begin{aligned}
p(email|ham) &= (1/10)^3 = 0.001 \\
p(email|spam) &= 3/20 * 2/20 * 2/20 = 0.0015 \\
p(ham|email) &\propto \frac{1}{3} * 0.001 = 0.000333... \\
p(spam|email) &\propto \frac{2}{3} * 0.0015 = 0.001
\end{aligned}
$$

Normalizing this yields 0.25 and 0.75 probabilities for the two classes respectively.

(g) What are other possible features you can consider?
We can incorporate any features that are representative of spam emails in addition to word frequencies. Examples are

- "Online Pharmacy"
- Mentions large quantities of money
- Sent by an unknown address with many numbers
- Subject is all capitals
- Email body has low ratio of text to images
- "One hundred percent guaranteed"
- "Prestigious Non-Accredited Universities"

Another approach is to also count occurrences of longer phrases (called n-grams).