Classification - Part 2

Invited Talk - December 6th 2023

Bayesian Classification

- A statistical classifier to predicts class membership probabilities
- Based on Bayes' Theorem.

Bayes' Theorem: Basics

Bayes' Theorem: $$P(H|\mathbf{X}) = \frac{P(\mathbf{X}|H)P(H)}{P(\mathbf{X})} = P(\mathbf{X}|H) \times P(H) / P(\mathbf{X})$$

- Let **X** be a data sample ("*evidence*"): class label is unknown
- Let H be a *hypothesis* that X belongs to class C_i
  - e.g., X will buy computer
- Classification is to determine P(H|**X**), (i.e., *posterior probability):* the probability that the hypothesis holds given the observed data sample **X**
  - e.g., X will buy computer, knowing that the age of X is 35 and the income is 40k, ...
- P(H) (*prior probability*): the initial probability
  - E.g., **X** will buy computer, regardless of age, income, ...
- P(**X**): probability that sample data is observed
- P(**X**|H) (likelihood): the probability of observing the sample **X**, given that the hypothesis holds
  - E.g., Given that **X** will buy computer, the prob. that X is 31..40, medium income

Classification Is to Derive the Maximum Posteriori

- Let D be a training set of tuples and their associated class labels, and each tuple is represented by an n-D attribute vector **X** = (x_1, x_2, …, x_n)
- Suppose there are *m* classes C_1, C_2, …, C_m.
- Classification is to derive the maximum posteriori, i.e., the maximal $P(C_i|\mathbf{X})$
- This can be derived from Bayes' theorem

$$P(C_i|\mathbf{X}) = \frac{P(\mathbf{X}|C_i)P(C_i)}{P(\mathbf{X})}$$

- Since P(X) is constant for all classes, only

$$P(C_i|\mathbf{X}) = P(\mathbf{X}|C_i)P(C_i)$$

needs to be maximized

Naïve Bayes Classifier

- A simplified assumption: attributes are conditionally independent (i.e., no dependence relation between attributes):

$$P(\mathbf{X}|C_i) = \prod_{k=1}^{n} P(x_k|C_i) = P(x_1|C_i) \times P(x_2|C_i) \times ... \times P(x_n|C_i)$$

- This greatly reduces the computation cost: Only counts the class distribution
- If A_k is categorical, $P(x_k|C_i)$ is the # of tuples in C_i having value x_k for A_k divided by $|C_{i,D}|$ (# of tuples of C_i in D)
- If A_k is continous-valued, $P(x_k|C_i)$ is usually computed based on Gaussian distribution with a mean μ and standard deviation σ $g(x,\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$

and $P(x_k|C_i)$ is $P(\mathbf{X}|C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i})$

Naïve Bayes Classifier: Training Dataset

Class:
C1:buys_computer = 'yes'
C2:buys_computer = 'no'

Data to be classified:
X = (age <=30,
Income = medium,
Student = yes
Credit_rating = Fair)

age	income	student	credit_rating	comp
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Naïve Bayes Classifier: An Example

age	income	student	credit_rating	com
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

$P(C_i)$: P(buys_computer = "yes") = 9/14 = 0.643

P(buys_computer = "no") = 5/14= 0.357

Compute $P(X|C_i)$ for each class

P(age = "<=30" | buys_computer = "yes") = 2/9 = 0.222

P(age = "<= 30" | buys_computer = "no") = 3/5 = 0.6

P(income = "medium" | buys_computer = "yes") = 4/9 = 0.444

P(income = "medium" | buys_computer = "no") = 2/5 = 0.4

P(student = "yes" | buys_computer = "yes) = 6/9 = 0.667

P(student = "yes" | buys_computer = "no") = 1/5 = 0.2

P(credit_rating = "fair" | buys_computer = "yes") = 6/9 = 0.667

P(credit_rating = "fair" | buys_computer = "no") = 2/5 = 0.4

X = (age <= 30 , income = medium, student = yes, credit_rating = fair)

$P(X|C_i)$: P(X|buys_computer = "yes") = 0.222 x 0.444 x 0.667 x 0.667 = 0.044

P(X|buys_computer = "no") = 0.6 x 0.4 x 0.2 x 0.4 = 0.019

$P(X|C_i)*P(C_i)$: P(X|buys_computer = "yes") * P(buys_computer = "yes") = 0.028

P(X|buys_computer = "no") * P(buys_computer = "no") = 0.007

Therefore, X belongs to class ("buys_computer = yes")

Avoiding the Zero-Probability Problem

- Naïve Bayesian prediction requires each conditional prob. be **non-zero**. Otherwise, the predicted prob. will be zero

$$P(X \mid C_i) = \prod_{k=1}^{n} P(x_k \mid C_i)$$

- Ex. Suppose a dataset with 1000 tuples, income=low (0), income= medium (990), and income = high (10)
- Use **Laplacian correction** (or Laplacian estimator)
  - *Adding 1 to each case*
    - Prob(income = low) = 1/1003
    - Prob(income = medium) = 991/1003
    - Prob(income = high) = 11/1003
  - The "corrected" prob. estimates are close to their "uncorrected" counterparts

Naïve Bayes Classifier: Comments

- Advantages
  - Easy to implement
  - Good results obtained in most of the cases
- Disadvantages
  - Assumption: class conditional independence, therefore loss of accuracy
  - Practically, dependencies exist among variables
    - E.g., hospitals: patients: Profile: age, family history, etc.
      - Symptoms: fever, cough etc., Disease: lung cancer, diabetes, etc.
    - Dependencies among these cannot be modeled by Naïve Bayes Classifier
- How to deal with these dependencies? Bayesian Belief Networks

Model Evaluation and Selection

- Evaluation metrics: How can we measure accuracy?
- Use **validation test set** of class-labeled tuples instead of training set when assessing accuracy
- Methods for estimating a classifier accuracy:
  - Holdout method
  - Cross-validation
  - Bootstrap

FN - Actually yes predicted no
FP - Actually no predicted yes
False Negative (FN): The number of instances incorrectly predicted as negative (but are actually positive)

Classifier Evaluation Metrics: Confusion Matrix

A confusion matrix is a table used in machine learning and statistics to evaluate the performance of a classification algorithm.

Confusion Matrix:

Predicted

Actual class\Predicted class	C_1	$\neg C_1$
C_1	**True Positives (TP)**	**False Negatives (FN)**
$\neg C_1$	**False Positives (FP)**	**True Negatives (TN)**

Example of Confusion Matrix:

Actual class\Predicted class	buy_computer = yes	buy_computer = no	Total
buy_computer = yes	**6954**	**46**	7000
buy_computer = no	**412**	**2588**	3000
Total	7366	2634	10000

Positive, Negative - predicted

TP here stands for True Positive predictions, for a binary classification problem like classifying the fraudulent transactions as 1, TP will give the count of the number of 1 s that were correctly classified as 1, i.e., number of fraudulent transactions that were classified as fraudulent. TN stands for true negative predictions, i.e., number of 0 s, non-fraudulent transactions, classified as 0. FP (False Positive) is the count of number of non-fraudulent transactions that were classified as fraudulent and FN (False Negative) is the count of number of fraudulent transactions that were classified as non-fraudulent.

Sensitivity, also known as True Positive Rate, Recall, or Sensitivity, is a performance metric used in binary and multiclass classification models.
It measures the ability of a model to correctly identify positive instances out of the total actual positive instances in a dataset.
Specificity is a performance metric used in binary and multiclass classification models. It measures the ability of a model to correctly identify negative instances out of the total actual negative instances in a dataset. Specificity is also referred to as True Negative Rate

Accuracy, Error Rate, Sensitivity and Specificity

A\P	C	¬C	
C	**TP**	**FN**	**P**
¬C	**FP**	**TN**	**N**
	P'	**N'**	**All**

- **Classifier Accuracy,** or recognition rate: percentage of test set tuples that are correctly classified
  - **Accuracy = (TP + TN)/All**
- **Error rate:** *1 – accuracy*, or
  - **Error rate = (FP + FN)/All**

Class Imbalance Problem:

- One class may be *rare*, e.g. fraud, or HIV-positive
- Significant *majority of the negative class* and minority of the positive class
- **Sensitivity**: True Positive recognition rate
  - **Sensitivity = TP/P** Sensitivity = TP/(TP+FN)
- **Specificity**: True Negative recognition rate
  - **Specificity = TN/N**

Precision and Recall, and F-measures

- **Precision**: exactness – what % of tuples that the classifier labeled as positive are actually positive

$$precision = \frac{TP}{TP + FP}$$

- **Recall:** completeness – what % of positive tuples did the classifier label as positive. Perfect score is 1.0

$$recall = \frac{TP}{TP + FN}$$

- ***F* measure (*F_1* or *F*-score)**: harmonic mean of precision and recall,

$$F = \frac{2 \times precision \times recall}{precision + recall}$$

- ***F_β*:** weighted measure of precision and recall. Assigns ß times as much weight to recall as to precision

$$F_\beta = \frac{(1 + \beta^2) \times precision \times recall}{\beta^2 \times precision + recall}$$

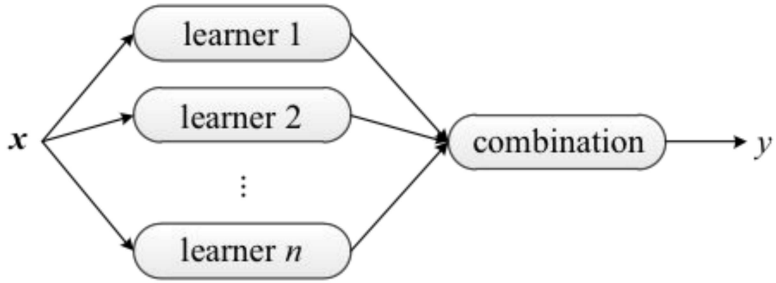
Classifier Evaluation Metrics: Example

Actual Class\ Predicted class	cancer = yes	cancer = no	Total	Recognition(%)
cancer = yes	**90**	**210**	300	30.00 (*sensitivity*
cancer = no	**140**	**9560**	9700	98.56 (*specificity)*
Total	230	9770	10000	96.40 (*accuracy*)

Precision = 90/230 = 39.13% *Recall* = 90/300 = 30.00%

Ensemble Methods: Increasing the Accuracy

combining multiple models



- An ensemble for classification is a composite model, made up of a combination of classifiers.
- The individual classifiers vote, and a class label prediction is returned by the ensemble based on the collection of votes.
- Ensembles tend to be more accurate than their component classifiers.
- Popular ensemble methods
  - Bagging: averaging the prediction over a collection of classifiers
  - Boosting: weighted vote with a collection of classifiers
  - Ensemble: combining a set of heterogeneous classifiers

Why it work ?

Assume 3 independent classifiers, each of which can classify a sample x correctly with 70% accuracy. The probability that their ensemble classifier correctly classify x is:

$$0.7^3 + {}^3C_2 \text{ X } 0.7^2 \text{ X } 0.3 = 0.78$$

What if we use 5 independent classifiers ?

Bagging

Also bootstrap aggregation
one of the technique in bagging is Random Forest

Algorithm: Bagging. The bagging algorithm—create an ensemble of classification models for a learning scheme where each model gives an equally weighted prediction.

Input:

- D, a set of d training tuples;
- k, the number of models in the ensemble;
- a classification learning scheme (decision tree algorithm, naïve Bayesian, etc.).

In sampling with replacement, each item selected from the population is returned to the population before the next selection.

Output: The ensemble—a composite model, $M*$.

Method:

(1) **for** $i = 1$ to k **do** // create k models:
(2) create bootstrap sample, D_i, by sampling D with replacement;
(3) use D_i and the learning scheme to derive a model, M_i;
(4) **endfor**

Lets suppose, for a given problem statement, we have D datasets
We will create multiple models
For each and every model, we provide sample of dataset D'
For another model, dataset is resampled and provided to second model
this is called as row sampling with replacement
so, each model will get trained on different set of data
Once model is prepared, we need to predict some value on given test data
Test data is sent across all models and each of them will make its own prediction

To use the ensemble to classify a tuple, X:

let each of the k models classify X and return the majority vote;

Correct prediction is considered to be one with majority vote

Boosting

- **Weights** are assigned to each training tuple.
- Training sets are sampled with replacement, according to weights.
- A series of k classifiers is iteratively learned.
- After a classifier M_i is learned, the weights are updated to allow the subsequent classifier, M_{i+1}, to **pay more attention to the training tuples that were misclassified by M_i**
- The final **M* combines the votes** of each individual classifier, where the weight of each classifier's vote is a function of its accuracy.
- Comparing with bagging: Boosting tends to have greater accuracy, but it also risks overfitting the model to misclassified data

Adaboost (Freund and Schapire, 1997)

Given a set of *d* class-labeled tuples, ($\mathbf{X_1}$, y_1), …, ($\mathbf{X_d}$, y_d)

Initially, all the weights of tuples are set the same (1/d)

Generate k classifiers in k rounds. At round i,

- Tuples from D are sampled (with replacement) to form a training set D_i of the same size
- Each tuple's chance of being selected is based on its weight
- A classification model M_i is derived from D_i
- Its error rate is calculated using D as a test set
- If a tuple is misclassified, its weight is increased, o.w. it is decreased

Error rate: err($\mathbf{X_j}$) is the misclassification error of tuple $\mathbf{X_j}$. Classifier M_i error rate is the sum of the weights of the misclassified tuples:

$$error(M_i) = \sum_j^d w_j \times err(\mathbf{X_j})$$

The weight of classifier M_i's vote is

$$\log \frac{1 - error(M_i)}{error(M_i)}$$

Random Forest (Breiman 2001)

- Bagging is not well combined with decision trees, because the ID3 Algorithm will generate highly similar/correlated trees
- Randomness is added to the tree induction Algorithm, as follows:
  - Before each split, a random selection of L attributes out of the available D attributes is made.
  - The split attribute is only selected among these L attributes.
- When L is much smaller than D, the trees in the forest are highly independent, so the ensemble functions well. But a single classifier will have a poor accuracy.
- What happens when L is close to D ?
- What if D is small ?

Credits and Readings

- These slides use material from:
  - Han, J., Kamber, M. and Pei, J. Data Mining Concepts and Techniques, Morgan Kaufmann Publishers, Burlington.

problem with decision tree
if data is changed slightly, it will create different decision tree
i.e. our data is highly sensitive to training data

In random forest, we combine multiple decision trees which is less sensitive to data
why forest -> because we use multiple trees
n number of datasets are created to created n models,
each dataset contains same number of rows by performing random sampling with replacement

only subset of features are used to train each decision tree
for new input provided, we pass this dataset to each trees and note down prediction
and the result with be determined by maximum voting

why random? used two random processes - bootstrapping and random feature selection
bootstrapping makes sure that we are not using same data each time
feature selection - subset of feature is selected each time
if same feature is selected, each time they may end up having same decision node and that will behave very similarity which increases variance
how many features to select - researchers have found that number of features equal to log or square root of feature works well
how to use this for regression? while combining prediction, take average