Department of Computer Engineering
University of Peradeniya

Data Mining and Machine Learning
Lab 06 - Classification

August 23, 2017

1 Introduction to Naïve Bayes

Naïve Bayes methods are a set of supervised learning algorithms based on Bayes' theorem with the "Naïve" assumption of independence between every pair of features. Given a class variable y and a dependent feature vector X_1 through X_n, Bayes' theorem states the following relationship.

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y)P(x_1, \ldots x_n \mid y)}{P(x_1, \ldots, x_n)}$$

In spite of their apparently over-simplified assumptions, naïve Bayes classifiers have worked quite well in many real-world situations, famously document classification and spam filtering.

2 Introduction to K Nearest Neighbors

The k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space.

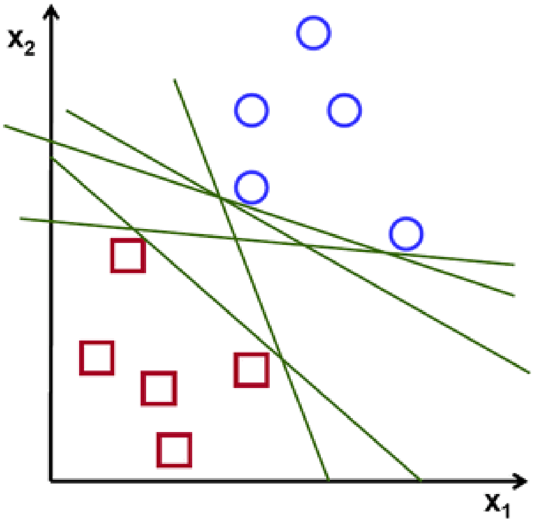
The principle behind nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point, and predict the label from these. The number of samples can be a user-defined constant (k-nearest neighbor learning), or vary based on the local density of points (radius-based neighbor learning). The distance can, in general, be any metric measure: standard Euclidean distance is the most common choice. Neighbors-based methods are known as non-generalizing machine learning methods, since they simply "remember" all of its training data.

3 Introduction to Support Vector Machines

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples.

In which sense is the hyperplane obtained optimal? Let's consider the following simple problem:

For a linearly separable set of 2D-points which belong to one of two classes, find a separating straight line.



In the above picture you can see that there exists multiple lines that offer a solution to the problem. Is any of them better than the others? We can intuitively define a criterion to estimate the worth of the lines:A line is bad if it passes too close to the points because it will be noise sensitive and it will not generalize correctly. Therefore, our goal should be to find the line passing as far as possible from all points.

4 Try Out

Import the necessary libraries required for the analysis and load the iris data set to try out different classification algorithms.

```
import numpy as np
from sklearn import datasets
iris = datasets.load_iris()
X = iris.data
Y = iris.target
```

4.1 Gaussian Naive Bayes

For discrete features like the ones encountered in document classification , multinomial and Bernoulli distributions are popular. When dealing with continuous data, a typical assumption is that the continuous values associated with each class are distributed according to a Gaussian distribution.GaussianNB implements the Gaussian Naive Bayes algorithm for classification. The likelihood of the features is assumed to be Gaussian:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma^2_y}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma^2_y}\right)$$

The parameters σ_y and μ_y are estimated using maximum likelihood.

```
from sklearn.naive_bayes import GaussianNB
clf = GaussianNB() # clf is a classifier
clf.fit(X,Y)
print('Training Accuracy: ',clf.score(X,Y)) # Training accuracy
```

The `score()` function returns the mean accuracy on the given test data and labels. In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

4.2 Multinomial Naive Bayes

The multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification). The multinomial distribution normally requires integer feature counts.

```
from sklearn.naive_bayes import MultinomialNB
clf=MultinomialNB() # clf is a classifier.
clf.fit(X,Y)
print('Training Accuracy: ',clf.score(X,Y)) # Training accuracy
```

4.3 Nearest Neighbor

For finding the nearest neighbors, *sklearn.neighbors* can be used.

```
from sklearn import neighbors, datasets
clf = neighbors.KNeighborsClassifier(n_neighbors=1) # 1-Nearest Neighbor
clf.fit(X,Y) # Model
print('Training Accuracy: ',clf.score(X,Y)) # Training accuracy
```

4.4 Support Vector Machine

```
from sklearn import svm
clf = svm.SVC(kernel='linear', C=1,gamma=1).fit(X, Y) # clf is a
classifier
clf.fit(X,Y) # Model
print('Training Accuracy: ',clf.score(X,Y)) # Training accuracy
```

Parameters:

- **C** : float, optional (default=1.0).Penalty parameter C of the error term.
- **kernel** : string, optional (default='rbf'). Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape (n-samples, n-samples).
- **gamma** : Defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'

4.5 Splitting dataset as training and testing

```
# split data as 2/3 for training and 1/3 for testing
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.333,
random_state=0)
clf.fit(X_train,Y_train) # clf is a classifier.
print('Test Accuracy: ',clf.score(X_test,Y_test)) # Test accuracy
```

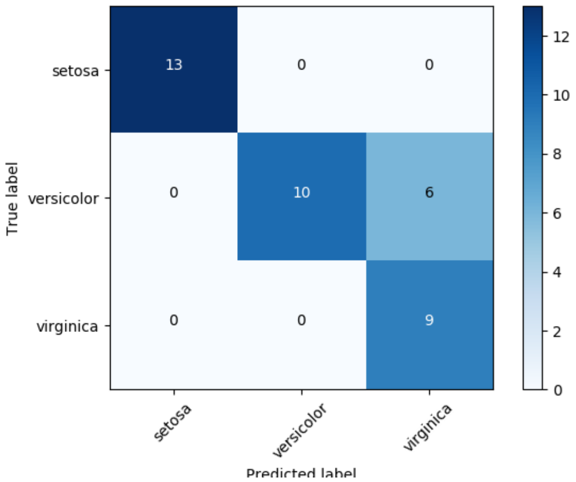
4.6 Use $cross - val - score$ helper function for cross validation

In k-fold cross-validation, the original sample is randomly partitioned into k equal size subsamples. Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining k-1 subsamples are used as training data. The cross-validation process is then repeated k times (the folds), with each of the k subsamples used exactly once as the validation data. The k results from the folds can then be averaged (or otherwise combined) to produce a single estimation.

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(clf, X, Y, cv=10) # 10-fold cross validation
print(scores) # Results for all the folds
print("10CV Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(),scores.std()*2))
```

4.7 Compute confusion matrix to get the details of the classification

Confusion matrix is a specific table layout that allows visualization of the performance of an algorithm. Each column of the matrix represents the instances in a predicted class while each row represents the instances in an actual class.The diagonal elements represent the number of points for which the predicted label is equal to the true label, while off-diagonal elements are those that are mislabeled by the classifier.

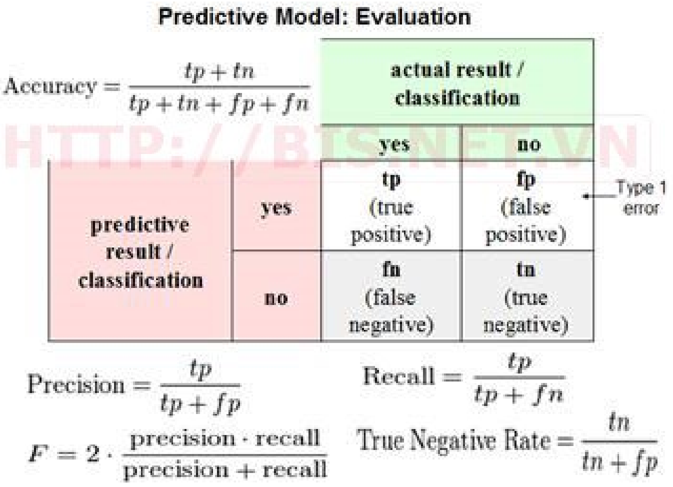


```
from sklearn.metrics import confusion_matrix
```

```
Y_pred = clf.fit(X_train, Y_train).predict(X_test)
print(confusion_matrix(y_test, y_pred))
```

4.8 Precision recall details

In information retrieval, precision is a measure of result relevancy, while recall is a measure of how many truly relevant results are returned. High scores for both show that the classifier is returning accurate results (high precision), as well as returning a majority of all positive results (high recall).



Predictive Model: Evaluation

$$\text{Accuracy} = \frac{tp + tn}{tp + tn + fp + fn}$$

		actual result / classification	
		yes	no
predictive result / classification	yes	tp (true positive)	fp (false positive)
	no	fn (false negative)	tn (true negative)

$$\text{Precision} = \frac{tp}{tp + fp} \qquad \text{Recall} = \frac{tp}{tp + fn}$$

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \qquad \text{True Negative Rate} = \frac{tn}{tn + fp}$$

```
from sklearn.metrics import classification_report
target_names =['setosa', 'versicolor', 'virginica'] # provide class labels
print(classification_report(Y_test, Y_pred, target_names=target_names))
```

5 Lab Exercise

Create a model for predicting animal type

1. Load the zoo dataset. Observe attributes and their values.

2. Evaluate algorithms using the following testing options:

    - the training set
    - by splitting the data as 2/3 for training and 1/3 for testing
    - 10-fold cross validation

    Use the decision tree classifier and record the classification accuracies for the three testing options. Which one of the testing option provides more realistic future performance? Why?

3. Compute confusion matrix all the classifiers used in the lab and explain each classifier output.

4. Record the 10-fold cross validation accuracies of the classifiers.

5. Try another classification algorithm available in scikit-learn. Compare the result of the chosen algorithm with previous classifier outputs.

6 Submission

Submit a single .py file as [12|13]xxxlab06.py where xxx is your registration number. Add answers for questions 2 to 5 as comments in the same file.

7 Important

Make sure that you have the basic understanding of various classification algorithms. This lab is really important for successive labs. If you do not understand any concepts, make sure you get some help from instructors.

8 Deadline

August 29, 23:59:59 GMT+5:30.