K-NN and Bayes classifiers Exercises

Alice Caplier

Bayes Classifier: main principle

$$Pr(Y = y_k|X = x) = \frac{Pr(X=x \mid Y=y_k)\,\Pr(Y=y_k)}{\Pr(X=x)}$$

=> MAP problem: maximazing $\Pr(Y = y_k \mid X = x)$

Is equivalent to maximazing: $Pr(X = x \mid Y = y_k)\,\Pr(Y = y_k)$

- ✓ *Pr(Y=yk)* is easy to compute: number of occurrences of class y_k in the training set divided by the total number of training samples.
- ✓ $Pr(X = x \mid Y = y_k)$ has to be infered from the training set

Hyp: the observations ($X_1, X_2, \ldots, X_n$) of any observation vector X are **independant**

$$\Rightarrow \quad Pr(X = x \mid Y = y_k) = \prod_{i=1}^{n} \Pr(X = xi \mid Y = y_k)$$

And then a model assumption for each xi distribution (Gaussian or Bernouilli or …)

Bayes Classifier: the training step

For each class yk in Ytrain

Compute the probability p(yk) (the proportion of the yk class in the training set = number of occurrence of class k in the training set divided by the whole number of samples in the training set)

End

For each class yk

For each feature xi in Xtrain

Compute the mean and variance of the feature xi

Compute the associated Gaussian distribution P(xi | yk)

(=> Gaussian modeling of each feature distribution)

End

End

Bayes Classifier: the prediction step

For each sample xi in Xtest

- *Compute the product for all the features of p(yk)*p(xi | yk) for each class yk (independent variable assumption)*
- *Select the class with the highest product (this optimizes p(yk | xi) (cf. Bayes rule)*

End

Bayes Classifier: some code

Hyp: data = X_train and X_test ; labels = Y_train and Y_test

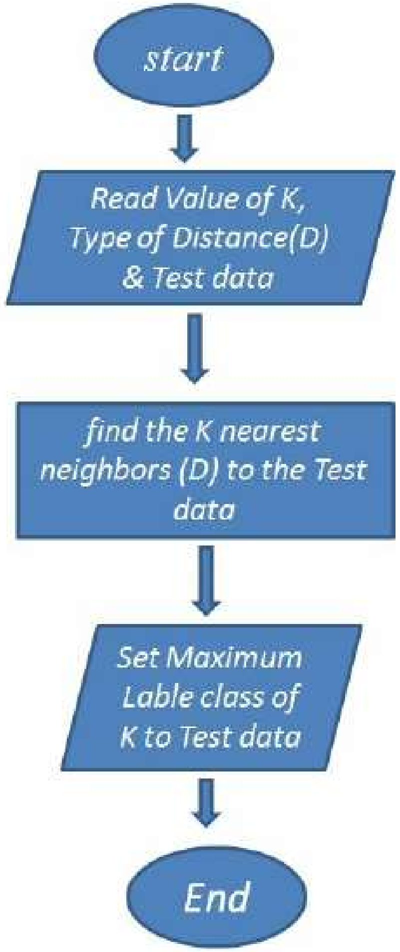
```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn import neighbors
```

```
#Naive bayes classification

gnb = GaussianNB()
gnb_learned = gnb.fit(X_train, Y_train)

Y_train_pred = gnb_learned.predict(X_train)
Y_test_pred = gnb_learned.predict(X_test)

# Accuracies: manual computation
train_error = ((Y_train != Y_train_pred).sum() / len(Y_train)*100)
test_error = ((Y_test != Y_test_pred).sum() / len(Y_test)*100)

print("Train accuracy : %d" %(100 - train_error))
print("Test accuracy : %d" %(100-test_error))

# Accuracies: direct computation
train_acc = accuracy_score(Y_train, Y_train_pred)
test_acc = accuracy_score(Y_test, Y_test_pred)

# Confusion matrix

C = confusion_matrix(Y_test, Y_test_pred)
```

Never train and test your algo on the same data.

Always compare the training accuracy and the testing accuracy

K-NN algorithm and code




```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn import neighbors
```

```
clf = neighbors.KNeighborsClassifier(K)
clf_learned = clf.fit(X_train, Y_train)
Y_pred = clf_learned.predict(X_test)
```

Diabete prediction exercice

Main purposes:

- Implement a naive Bayes classifier and a kNN algo
- Be familiar with accuracy score and confusion matrix
- Be aware of data normalization requirement

Diabete prediction

Data analysis

768 subjects, 8 attributes, 1 decision

- 8 different data features for training and prediction
  - Pregnancies : Number of times pregnant
  - Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test
  - BloodPressure: Diastolic blood pressure (mm Hg)
  - SkinThickness: Triceps skin fold thickness (mm)
  - Insulin: 2-Hour serum insulin (mu U/ml)
  - BMI: Body mass index (weight in kg/(height in m)^2)
  - DiabetesPedigreeFunction: Diabetes pedigree function
  - Age: Age (years)

- A single outcome: diabete or not diabete (One class classification problem)
  - Outcome: Class variable (0 or 1)

Diabete prediction: dataset analysis

In the dataset, 268 of 768 are 1, the others are 0

From the dataset, two mandatory parts :

- Training set used to train the models
- Testing set : unknown data to evaluate the generalization ability of the learned model

Diabete prediction: Data normalization

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1
5	116	74	0	0	25.6	0.201	30	0

Feature values might be very different from one feature to the other

⇒ During the Euclidean computation process, some features differences might be very huge wrt to other => if no data normalization, those features are going to have the biggest influence

⇒ **Data normalization is required for K-NN classifier**

```python
from sklearn import preprocessing
#scaler = preprocessing.StandardScaler().fit(X_train)
scaler = preprocessing.MinMaxScaler().fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Classification results

Bayes classifier

- 77% of global Acc on the training and the testing set

  => good training and good generalization ability
- Confusion matrix

128 23 => class 0, 128 right classif and 30 wrong classif => class 0 Acc = 85%

30 49 => class 1, 49 right classif and 23 wrong classif => class 1 Acc = 62%

=> algo more powerfull to detect non diabete than diabete cases probably because more non diabete samples in the training set

K-NN classifier

72% of Acc with a 3-NN classifier without feature normalization

77% of Acc with a 3-NN classifier with feature min-max normalization

⇒ Normalization is often necessary

⇒ KNN simple efficient algo. But very time consumming for large datasets.

Evaluation

A 10 minutes MCQ based on all the exercice results at the beginning of the next class

To keep in mind

Bayes classifier

- ✓ *Works for large dataset*
- ✓ *Quite simple classifier*

K-NN classifier

- ✓ *Computational time*
- ✓ *Pbl of choosing the K value*

Evaluation protocol

- ✓ *Look at the class repartition first (balanced or unbalanced classes)*
- ✓ *Compare the training error and the testing error (underfitting vs. overfitting)*
- ✓ *When giving accuracy, precise training or testing accuracy*
- ✓ *Think about cross validation if the dataset is small*
- ✓ *Be careful with the global accuracy when dealing with unbalanced classes. Think about exploring the confusion matrix*