Naive Bayes

Naive Bayes is a very simple but powerful classification method. For a given object x, Naive Bayes calculates x's probability to belong to each class y_i ($i = 1, \cdots, k$), using the Bayes' theorem:

$$P(y_i \mid x) = \frac{P(y_i)\, P(x_1, \cdots, x_m \mid y_i)}{P(x_1, \cdots, x_m)}.$$

Additionally, it assumes that the features are independent from each other (which is the reason why it is called naive):

$$P(x_1, \cdots, x_m \mid y_i) = \prod_{j=1}^{m} P(x_j \mid y_i).$$

So, we obtain:

$$P(y_i \mid x) = \frac{P(y_i) \prod_{j=1}^{m} P(x_j \mid y_i)}{P(x_1, \cdots, x_m)}.$$

For a given object x, Naive Bayes will output the the Maximum a Posteriori (MAP) estimate:

$$\hat{y} = \arg\max_y P(y \mid x) = \arg\max_y \frac{P(y) \prod_{j=1}^{m} P(x_j \mid y)}{P(x_1, \cdots, x_m)}.$$

Note that $P(x_1, \cdots, x_m)$ is constant. Thus, we can drop it to obtain:

$$\hat{y} = \arg\max_y P(y_i) \prod_{j=1}^{m} P(x_j \mid y).$$

Practical example

First of all, we do all the necessary imports and load the Mushroom dataset.

In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import accuracy_score, roc_curve, auc

# setting random seed
seed = 10

data = pd.read_csv('data/mushroom.csv')

# We drop the 'stalk-root' feature because it is the only one containing missing values.
data = data.drop('stalk-root', axis=1)
data.head()
```

Out[1]:

	cap-shape	cap-surface	cap-color	bruises?	odor	gill-attachment	gill-spacing	gill-size	gill-color	stalk-shape	...	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil colo
0	x	s	n	t	p	f	c	n	k	e	...	w	w	p	w
1	x	s	y	t	a	f	c	b	k	e	...	w	w	p	w
2	b	s	w	t	l	f	c	b	n	e	...	w	w	p	w
3	x	y	w	t	p	f	c	n	n	e	...	w	w	p	w
4	x	s	g	f	n	f	w	b	k	t	...	w	w	p	w

5 rows × 22 columns

Unfortunately, scikit-learn does not implement the classical Naive Bayes algorithm which calculates the conditional probabilities $P(x_j | y_i)$ as the proportion of objects from class y_i that assume each particular categorical value for feature j. However, scikit-learn contains the BernoulliNB class which assumes that data is distributed according to multivariate Bernoulli distributions.

So, for the Mushroom dataset, we can transform each categorical feature to dummy variables. **Note that such a conversion clearly violates the indepence assumption between features.** However, Naive Bayes has been proven to achieve good performance in several applications where indepence is violated (for example, in text classication).

In [2]:

```
# Creating a new DataFrame representation for each feature as dummy variables.
dummies = [pd.get_dummies(data[c]) for c in data.drop('label', axis=1).columns]

# Concatenating all DataFrames containing dummy variables.
binary_data = pd.concat(dummies, axis=1)

# Getting binary_data as a numpy.array.
X = binary_data.values

# Getting the labels.
le = LabelEncoder()
y = le.fit_transform(data['label'].values)

# Splitting the binary dataset into train and test sets.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.34, stratify=y, random_state=seed)

# Creates a BernoulliNB. binarize=None indicates that there is no need to binarize the input data.
nb = BernoulliNB(binarize=None)
nb.fit(X_train, y_train)
y_pred = nb.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print('BernoulliNB accuracy score: {}'.format(accuracy))
```

```
BernoulliNB accuracy score: 0.9464350343829171
```

In [3]:

```
# Getting the probabilities for each class.
y_prob = nb.predict_proba(X_test)

# Calculating ROC curve and ROC AUC.
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_prob[:, 1])
roc_auc = auc(false_positive_rate, true_positive_rate)

# Plotting ROC curve.
lw = 2
plt.plot(false_positive_rate, true_positive_rate, color='blue', lw=lw, label='ROC curve (area = {:.4f})'.format(roc_auc)
)
plt.plot([0, 1], [0, 1], color='red', lw=lw, linestyle='--', label='Random classifier')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
plt.legend(loc="lower right")

plt.show()
```