Classifiers

Yacine Debbabi

December 6, 2020

Contents

1 Bayes classifier

Consider n observations with features $X \in R^{n \times p}$ and their associated unique labels $(g_i)_{i=1}^n \in 1, ..., K^n$. We aim to build a classifier function minimizing the expected prediction error

$$\hat{G} = \text{argmax}_G \mathbb{E}\left[L(g, G(X)\right] \tag{1}$$

where L is a $K \times K$ loss matrix. That objective function can be expressed pointwise with a 0-1 loss function as

$$\begin{aligned} \mathbb{E}\left[L(g, G(X)|X = x\right] &= \sum_{k=1}^{K} L(g_k, G(x))\mathbb{P}(g_k|x) &(2)\\ &= 1 - \mathbb{P}(G(x)|x) &(3)\end{aligned}$$

The solution, called the Bayes classifier, is given by

$$\hat{G}(x) = \text{argmax}_g \mathbb{P}(g|x), \tag{4}$$

and its error rate is called the Bayes rate. Several classifiers are constructed by approximating the conditional class probability.

2 Baseline estimators

Simple error rate baselines can be obtained by testing dummy prediction strategies. These include (a) "most_frequent", i.e. always predict the most frequent class, (b) "uniform", i.e. generates predictions uniformly at random. Such strategies are implemented in "DummyClassifier".

```
from sklearn.dummy import DummyClassifier

dc = DummyClassifier(strategy="most_frequent").fit(X, y)
dc.score(X, y) # accuracy measure, i.e. % of observations classified correctly
```

With multiple classes, one can train binary classifiers to distinguish between a given class and the rest of the classes (**one-versus-all/rest**), or between two classes (**one-vs-one**). We usually focus on one-vs-all classifiers and identify the highest score to make a class prediction.

Stratified KFold helps with generating class-balanced sets here

3 Logistic regression

The model specifies the log-odds of the K classes via linear functions of x

$$\log \frac{\mathbb{P}(G=k|X=x)}{\mathbb{P}(G=K|X=x)} = \beta_k^0 + \beta_k^T x, \tag{5}$$

ensuring that all class probabilities sum to 1. The model is usually fit by maximum likelihood, i.e. by maximizing the log-likelihood

$$l(\beta) = \sum_{i=1}^{n} \log p_{g_i}(x_i;\beta). \tag{6}$$

When $K=2$, we code the two classes via 0/1 response, so the log-likelihood can be rewritten as

$$l(\beta) = \sum_{i=1}^{n} y_i \log p_{g_i}(x_i;\beta) + (1-y_i)\log\left(1-p_{g_i}(x_i;\beta)\right) \tag{7}$$

$$= \sum_{i=1}^{n} y_i \beta^T x_i - \log\left(1+e^{\beta^T x_i}\right) \tag{8}$$

where the intercept is accounted for by adding a constant feature equal to 1. Note the log-likelihood is concave; this can be seen by differentiating against x_i twice.

The relevance of features can be examined via the logistic regression Z-scores; see below.

```
import statsmodels.api as sm

X = df[["Pclass", "Sex_"]].values
y = df.Survived

X_ = sm.add_constant(X)
lr = sm.Logit(y, X_)
lr_res = lr.fit()

print(lr_res.summary())
```

```
                           Logit Regression Results
==============================================================================
Dep. Variable:                Survived  No. Observations:                  891
Model:                           Logit  Df Residuals:                      888
Method:                            MLE  Df Model:                            2
Date:                 Sun, 06 Dec 2020  Pseudo R-squ.:                  0.3029
Time:                         09:16:16  Log-Likelihood:                -413.60
converged:                       True  LL-Null:                       -593.33
Covariance Type:            nonrobust  LLR p-value:                 8.798e-79
==============================================================================
coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
```

```
const          3.2946      0.297     11.077      0.000       2.712       3.878
x1            -0.9606      0.106     -9.057      0.000      -1.168      -0.753
x2            -2.6434      0.184    -14.380      0.000      -3.004      -2.283
==============================================================================
```

Note the coefficient standard errors used to compute the Z-scores are derived using the asymptotic distribution of MLE coefficient estimates, i.e.

$$\sqrt{n}(\hat{\beta}_{\text{MLE}} - \beta) \to N(0, I^{-1}), \tag{9}$$

where I is the Fisher information matrix defined by

$$I_{j,k} := \mathbb{E}\left[-\frac{\partial^2 \ln f(X;\beta)}{\partial\beta_j \partial\beta_k}\right]. \tag{10}$$

4 Linear discriminant analysis

This classifier is built as a Bayes classifier by assuming $X|G = k \sim N(\mu_k, \Sigma)$. Note the covariance does not vary with the class. The conditional class probabilities are estimated with the Bayes rule as

$$\mathbb{P}(G = k|x) = \frac{f_k(x)\Pi_k}{\sum_{g=1}^{K} f_g(x)\Pi_g}, \tag{11}$$

and we assign the class with highest probability of occurence conditional on $X = x$. Taking the log ratio of two class probabilities shows the decision boundary is linear, i.e.

$$\begin{aligned} \log\frac{\mathbb{P}(G = k|X = x)}{\mathbb{P}(G = l|X = x)} &= \log\frac{f_k(x)}{f_l(x)} + \log\frac{\pi_k}{\pi_l} &(12)\\ &= \log\frac{\pi_k}{\pi_l} - \frac{1}{2}(\mu_k + \mu_l)^T\Sigma^{-1}(\mu_k - \mu_l) + x^T\Sigma^{-1}(\mu_k - \mu_l). &(13)\end{aligned}$$

The means and covariance matrix are directly estimated from the data. This makes the techniques sensitive to outliers. The classification is equivalent to computing distances to cluster means in a transformed space. Distances orthogonal to the $K-1$-dimension affine subspace where the K cluster centers lie are irrelevant. This suggests dimensionality can be reduced by projecting the data onto that affine space. That projection can be arrived to following the Fisher decomposition described below. COMPLETE INFO TO LINK THE TWO TECHNIQUES.

The Fischer decomposition consists of finding a sequence of orthonormal directions where the between-class variance is maximized relative to the within-class variance, i.e.

$$\text{argmax}_a \frac{a^T B a}{a^T W a}, \tag{14}$$

where the between-class and within-class variance are given by

$$\begin{aligned} B &= \sum_{g=1}^{K} N_g(\bar{x}_g - \bar{x})^T(\bar{x}_g - \bar{x}), &(15)\\ W &= \sum_{g=1}^{K}\sum_{g_i=g}(x_i - \bar{x}_g)^T(x_i - \bar{x}_g). &(16)\end{aligned}$$

```
import seaborn as sns
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

clf = LinearDiscriminantAnalysis().fit(X, y)
df[["C1","C2"]] = clf.transform(X) # coordinates in the Fisher directions

sns.scatterplot(x='C1', y='C2', data=df, hue='Species')
plt.show()
```

More information on LDA can be found here.

5 Performance measures

Let us denote TP, FP, TN and FN the number of true positives, false positives, true negatives and false negatives. To build some intuition, we consider the detection of a sickness via a test (+|-). We define the precision, i.e. P(sick|+), as

$$P = \frac{TP}{TP + FP}, \tag{17}$$

the recall / true positive rate, i.e. P(+|sick), as

$$R = \frac{TP}{TP + FN}, \tag{18}$$

and the false positive rate, i.e. P(+|healthy) as

$$\text{FPR} = \frac{FN}{FP + TN}. \tag{19}$$

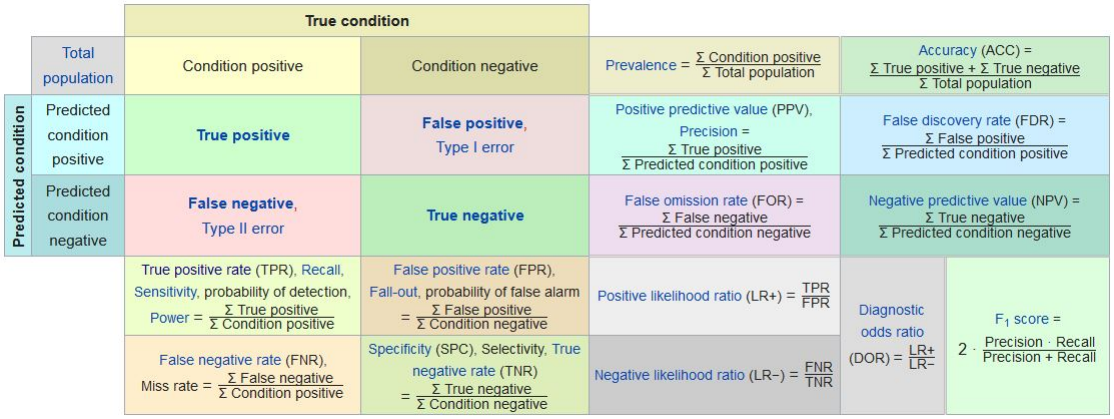

Figure 1: Performance metrics summary; from Wikipedia.

Classifier performance can be evaluated using precision vs. recall curves or ROC curves (recall/TPR vs. FPR). The curves are obtained by varying the classification threshold. It is preferrable to use ROC curves with well class-balanced datasets and precision/recall when one class is more significant than the other. Confusion matrices, i.e. $C_{i,j}$:= number of observations of class i predicted to be in j, also help understand what errors the classifier is making.

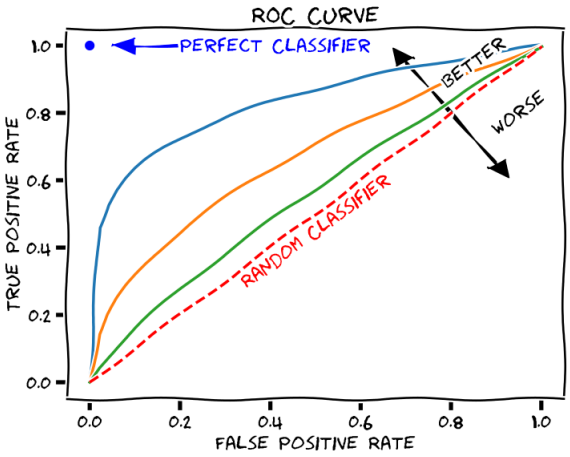

Figure 2: ROC (receiver operating characteristic) curve examples; from Wikipedia.

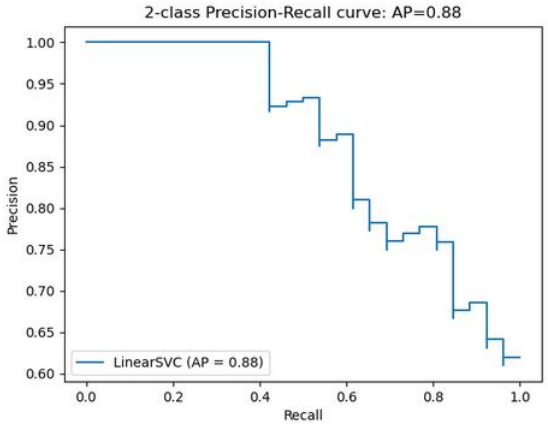

Figure 3: Precision/recall curve example; from Scikit-Learn.

```
from sklearn.linear_model import LogisticRegression

# binary classification

y_ = y=="Iris-setosa"
lr = LogisticRegression().fit(X, y_)
y_hat = lr.predict(X)
y_score = lr.predict_proba(X)[:,1]

from sklearn.metrics import confusion_matrix

confusion_matrix(y_, y_hat)
# C_ij:= number of observations of class i predicted to be in j

from sklearn.metrics import roc_curve, roc_auc_score, SCORERS

fpr, tpr, _ = roc_curve(y_, y_score)
plt.plot(fpr, tpr, 'bo-')
```

```
plt.show()

# multi-class classification

from sklearn.model_selection import cross_val_score

lr = LogisticRegression().fit(X, y)
y_scores = lr.predict_proba(X)
roc_auc_score(y, y_scores, multi_class='ovr')
cross_val_score(lr, X, y, scoring="roc_auc_ovr")
```