Advanced Python for Neuroscientists
Lecture 3: Classification

Summer 2021

Princeton Neuroscience Institute
Instructors: Yisi Zhang & Tyler Giallanza

July 13, 2021

Recap

Lecture 1

- Types of learning: supervised vs. unsupervised
- Supervised learning: regression vs. classification
- Bias-variance trade-off
- Training vs. testing

Lecture 2

- Simple linear regression
- Multiple linear regression
- Variable selection
- Shrinkage

Outline

- Logistic regression
- Linear discriminant analysis
- Tree-based methods
- Support vector machine
- Cross-validation
- Error analysis

Classification problems

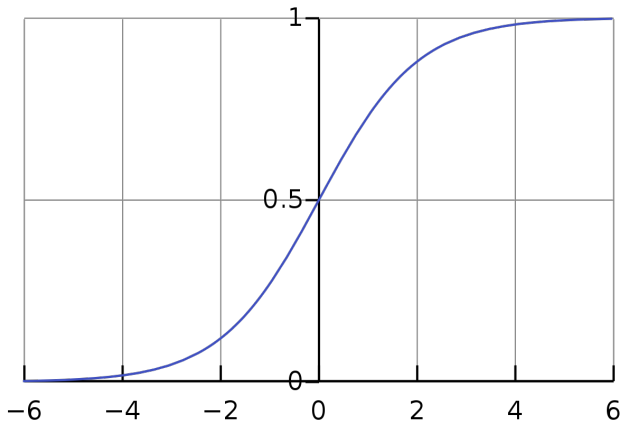
Suppose we observe pairs of (X, y), where class $Y \in a, b, ...$ and X the features that help characterize Y, a classifier aims to determine some decision rule that divides feature space into class-labelled regions.
The decision rules can be based on, e.g.

- prior probabilities $(P(C_1), P(C_2), ...)$
- posterior probabilities $(P(C_1|x), P(C_2|x), ...)$
- likelihood $(P(x|C_1), P(x|C_2), ...)$
- risk (cost associated with assigning the wrong label)
- other

3.1 Logistic regression

One idea is still adopting the regression strategy. Recall that linear regression has the form $Y = X\beta$.
Suppose that Y is binary, we can convert it to *posterior* probability $p(X) = Pr(Y = 1|X) \in [0, 1]$ that takes the form of a *logistic function*,

$$p(X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X)}}.$$



3.1 Logistic regression

Solving for $X\beta$ gives us a *linear* model (logistic regression)

$$log(\frac{p(X)}{1-p(X)}) = \beta_0 + \beta_1 X,$$

where the left-hand side is the log-odds (logit) of the probability.

If Y has $K > 2$ possible classes, we can perform a multinomial logistic regression

$$\log \frac{Pr(Y = k|X = x)}{Pr(Y = K|X = x)} = \beta_{k0} + \beta_k^T x, \quad k = 1, ..., K-1$$

And a given observation is assigned to the class with the highest log-odds.

3.1 Logistic regression

You should also check the K-nearest neighbors (KNN) classifier, which also base on the posterior probability of Y given X but with a different model assumption of the probability:

$$Pr(Y = j|X = x_0) = \frac{1}{K} \sum_{i \in \mathcal{N}_0} I(y_i = j),$$

where $\mathcal{N}_0$ represents the K points in the training set closest to x_0.

3.1 Logistic regression

Exercise: Fit logistic regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import make_blobs
# make a toy dataset with 2 predictors and 2 classes
X, y = make_blobs(n_samples=30, centers=2, n_features=2,
random_state=0)
clf = LogisticRegression(random_state=0).fit(X, y)
clf.score(X, y)
```

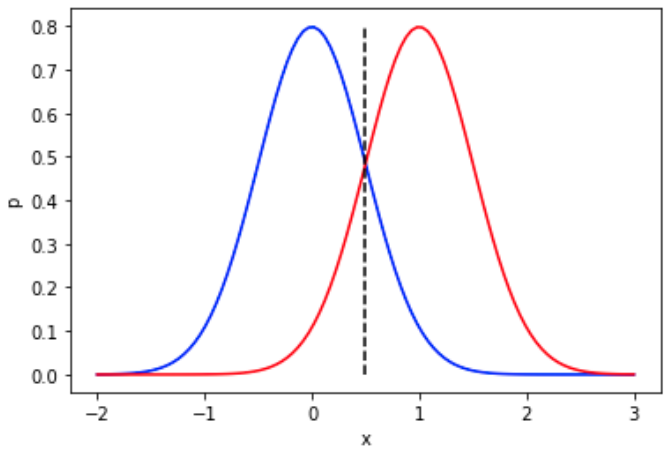
3.2 Linear Discriminant analysis

Instead of directly modeling $Pr(Y = k|X = x)$ as logistic regression and KNN, we can model $Pr(X = x|Y = k)$ and use Bayes' theorem to estimate $Pr(Y = k|X = x)$. The linear discriminant analysis (LDA) classifier adopts this strategy. Why do we need this method?

– The parameter estimate for logistic regression can be unstable when the classes are well-separated and when n is small while X is approximately normally distributed.
– LDA is popular for more than two classes.

3.2 Linear Discriminant analysis

How do we model $Pr(X = x|Y = k)$?



$$f_k(x) \equiv Pr(X = x|y = k) = \frac{1}{\sqrt{2\pi}\sigma_k} exp(-\frac{(x-\mu_k)^2}{2\sigma_k^2})$$

$$Pr(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)}$$

3.2 Linear Discriminant analysis

We estimate model parameters μ_k, σ^2 (if assuming the same variance across classes), and π_k:

$$\hat{\mu}_k = \frac{1}{n_k} \sum_{i:y_i=k} x_i,$$

$$\hat{\sigma}^2 = \frac{1}{n-K} \sum_{k=1}^{K} \sum_{i:y_i=k} (x_i - \hat{\mu}_k)^2,$$

$$\hat{\pi}_k = n_k/n.$$

And assign the observation to the class for which the *discriminant function* is largest:

$$\hat{\delta}_k(x) = x \cdot \frac{\hat{\mu}_k}{\hat{\sigma}^2} - \frac{\hat{\mu}_k^2}{2\hat{\sigma}^2} + \log(\hat{\pi}_k).$$

3.2 Linear Discriminant analysis

If class-specific variance/covariance is assumed, it is called quadratic discriminant analysis (QDA), which should increase prediction accuracy with large training size.

3.2 Linear Discriminant analysis

Exercise: Fit LDA

```
from sklearn.discriminant_analysis import
LinearDiscriminantAnalysis
from sklearn.datasets import make_blobs
# make a toy dataset with 2 predictors and 2 classes
X, y = make_blobs(n_samples=30, centers=3, n_features=2,
random_state=2)
clf = LinearDiscriminantAnalysis()
clf.fit(X, y)
clf.score(X, y)
```

3.3 Tree-based methods

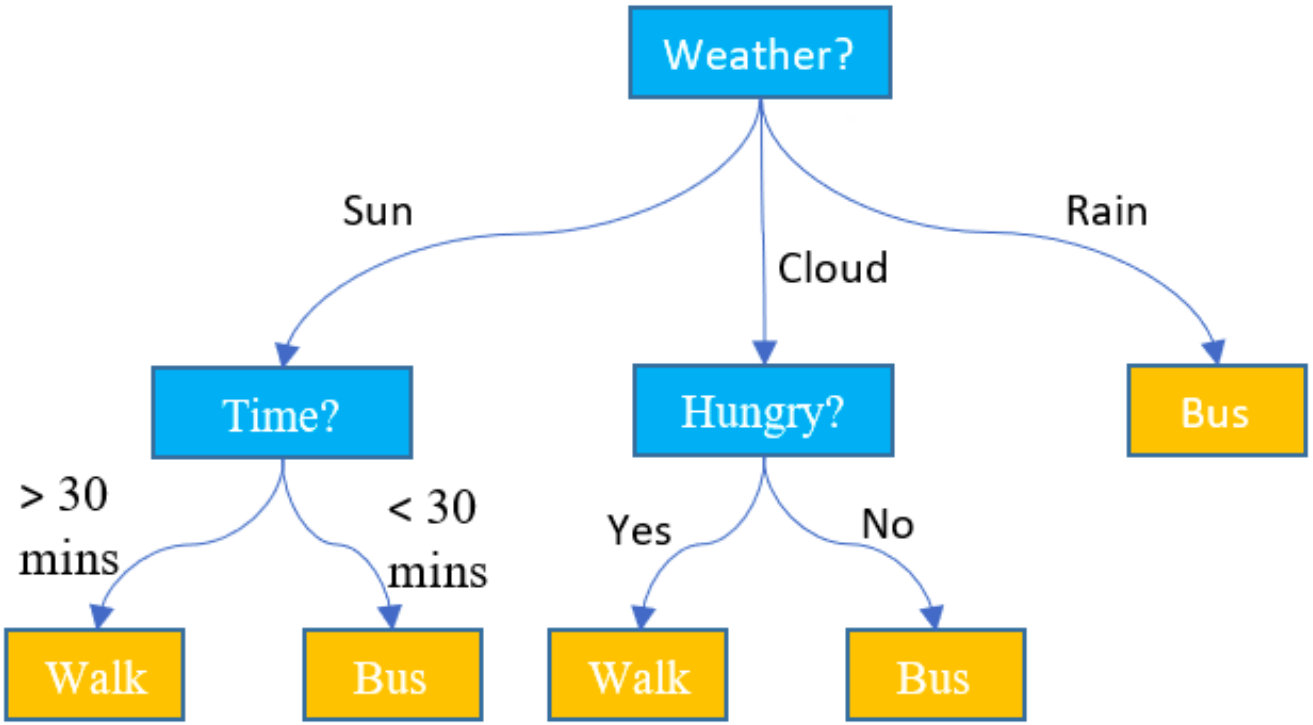
Decision trees

The idea is to strategically divide the predictor space into non-overlapping regions such that the range of a given region can well predict the class of an observation within that region.
In detail, for any predictor X_j and cutpoint s, we define the pair of half-planes

$$R_1(j,s) = X|X_j < s \quad \text{and} \quad R_2(j,s) = X|X_j \geq s,$$

and we seek the value of j and s that minimize the mis-classifications.

3.3 Tree-based methods



3.3 Tree-based methods

We can minimize the *classification error rate* (fraction not belong to the most common class):

$$E = 1 - \max_k(\hat{p}_{mk}),$$

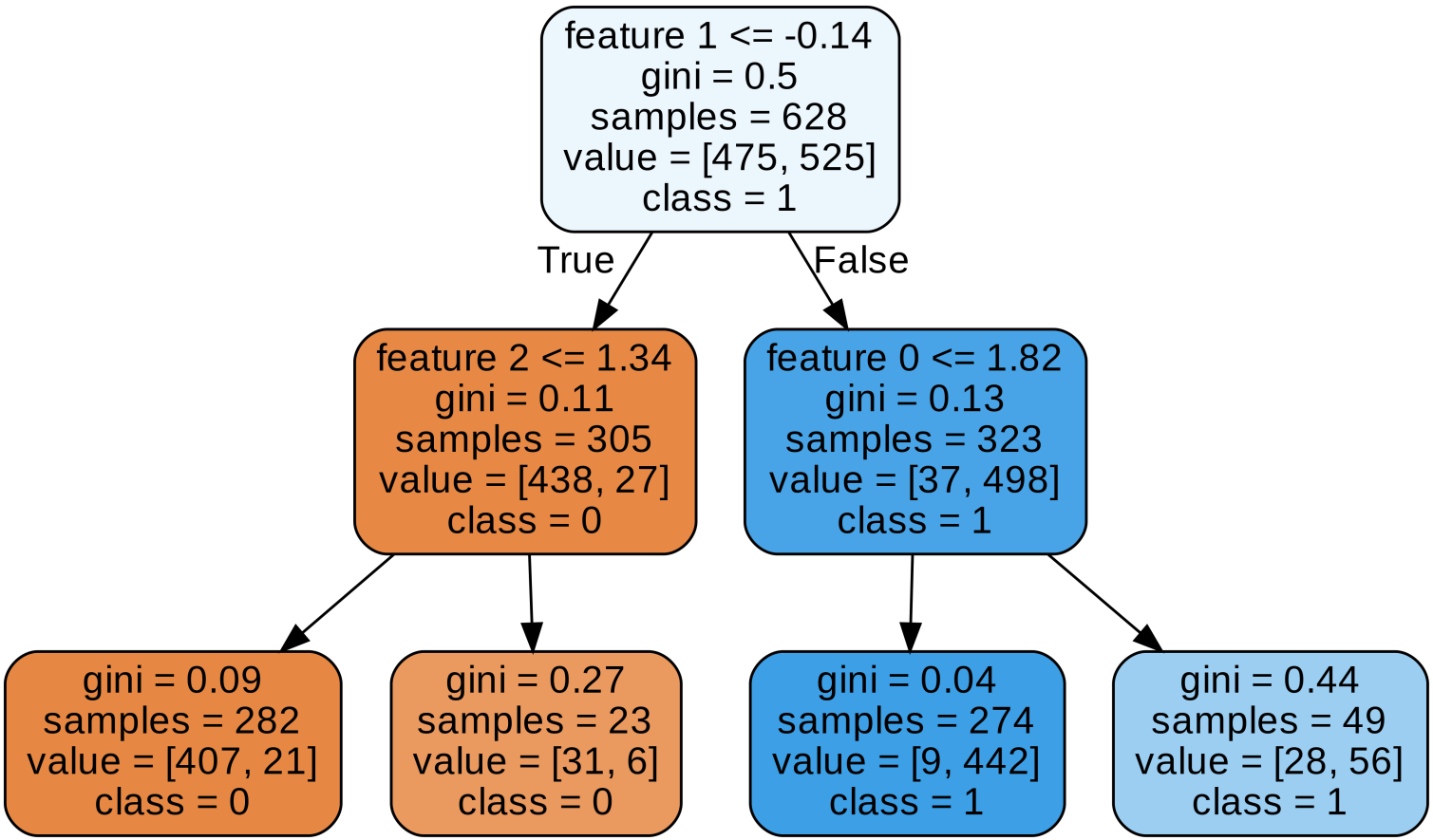
or the *Gini index* (total variance across the K classes):

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}),$$

or the *cross-entropy*:

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}.$$

3.3 Tree-based methods

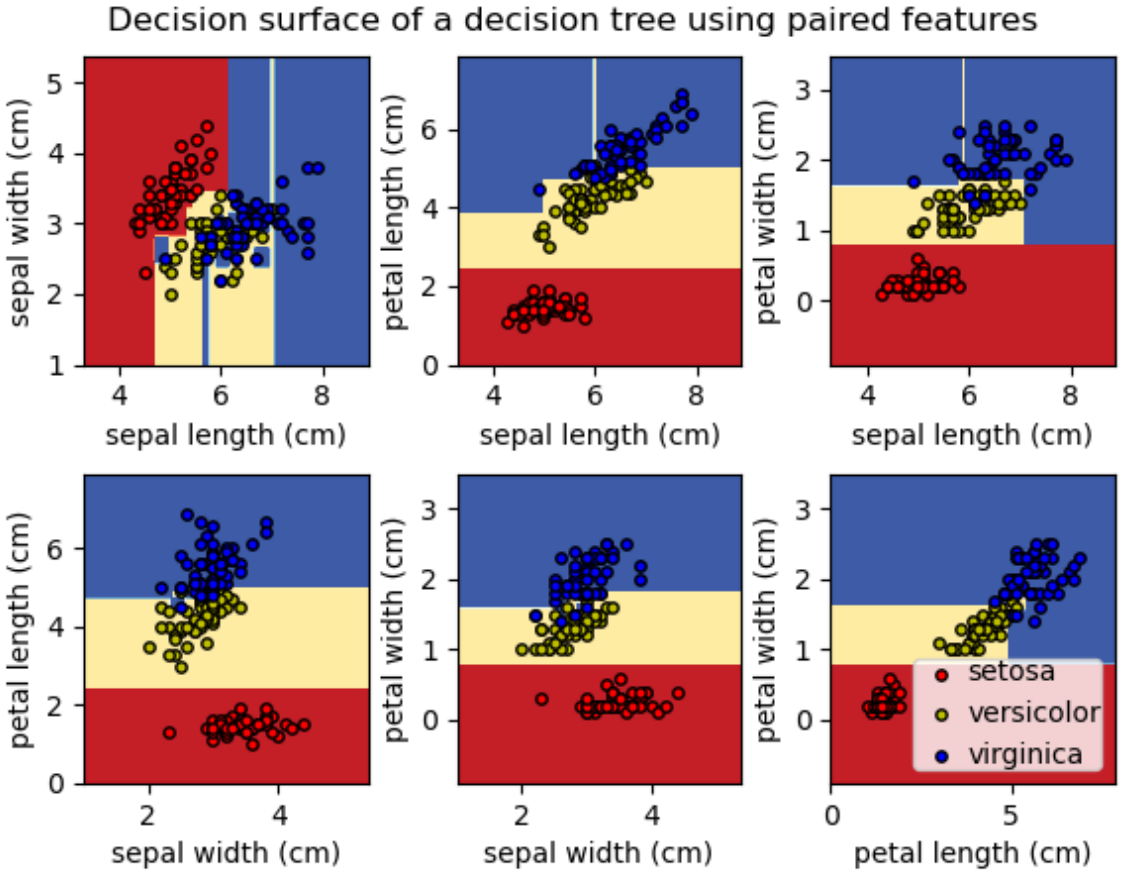


3.3 Tree-based methods

Why use decision trees?

- Can deal with non-linear relationships.
- Easy to interpret.
- Can handle qualitative predictors.

3.3 Tree-based methods



3.3 Tree-based methods

However, decision-trees generally do not have the same level of predictive accuracy as other approaches. This can be improved by aggregating many decision trees using, e.g., random forest.

The idea is constructing a bunch of random trees using bootstrapped samples and predict by consensus.

Bootstrapping: random sampling with replacement

3.3 Tree-based methods

In each bootstrap cycle, recursively build a tree through:

- i. Select m (usually taking $\sqrt{p}$) variables at random from p variables.
- ii. Pick the best variable/cutpoint among the m.
- iii. Split the node into two.

After we have an ensemble of trees $\{T_b\}_1^B$, for a new x the class prediction of the bth random-forest tree is
$\hat{C}_{rf}^B(x) =$ majority vote $\{\hat{C}_b(x)\}_1^B$.

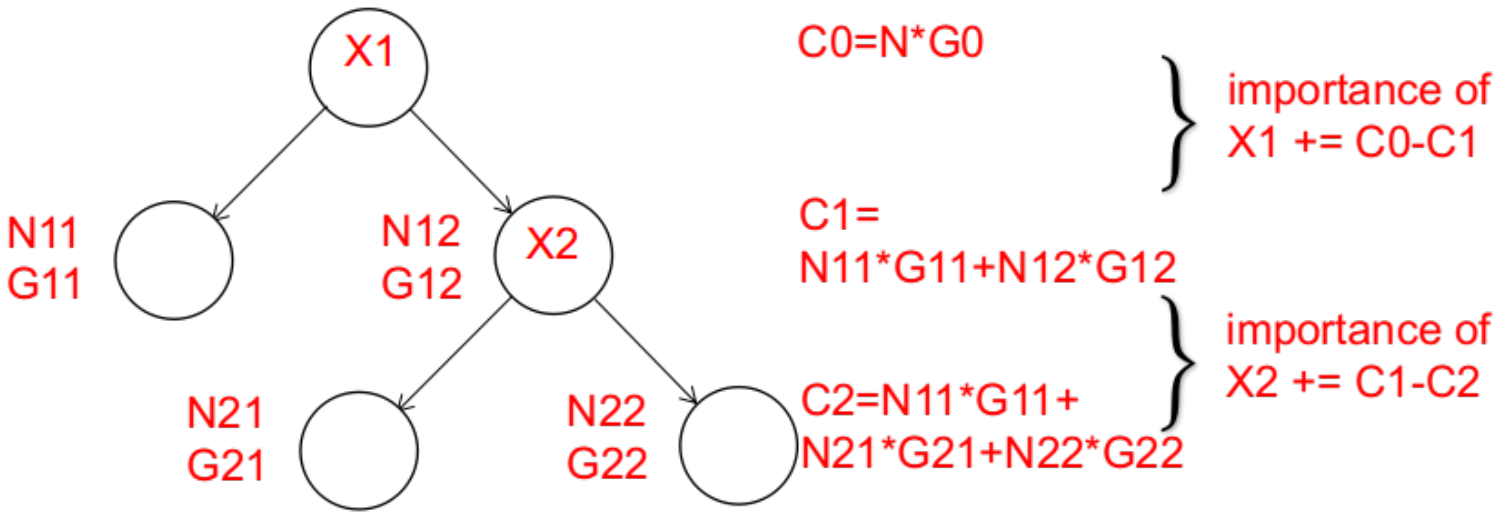
3.3 Tree-based methods

Exercise: Fit random forest

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
X, y = make_classification(n_samples=1000, n_features=4,
n_informative=2, n_redundant=0, random_state=0, shuffle=False)
forest = RandomForestClassifier(max_depth=2, random_state=0)
forest.fit(X, y)
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in
forest.estimators_], axis=0)
```

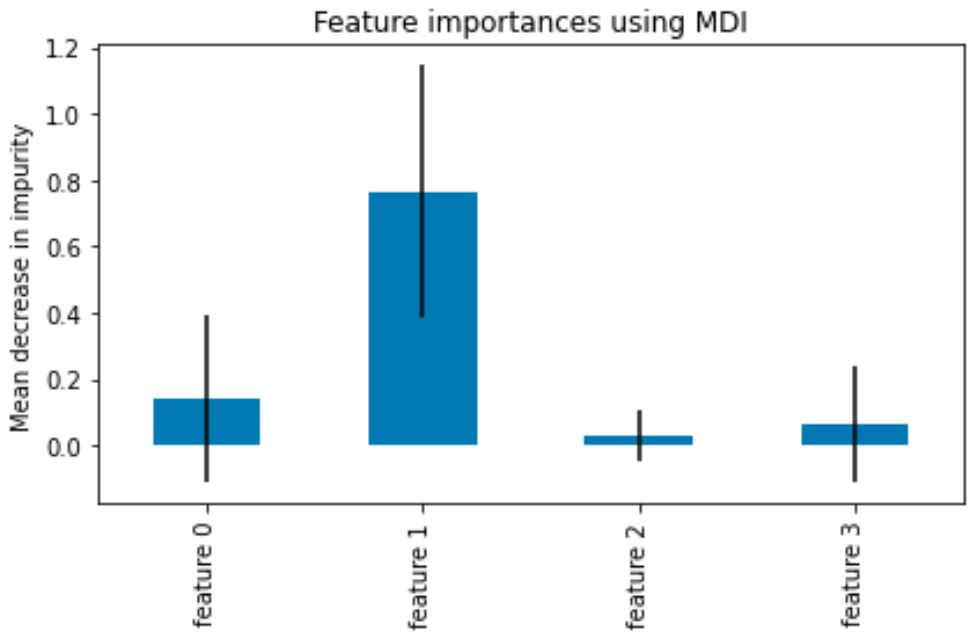
3.3 Tree-based methods

Random forest can be used to access feature importance.



3.3 Tree-based methods

Feature importance

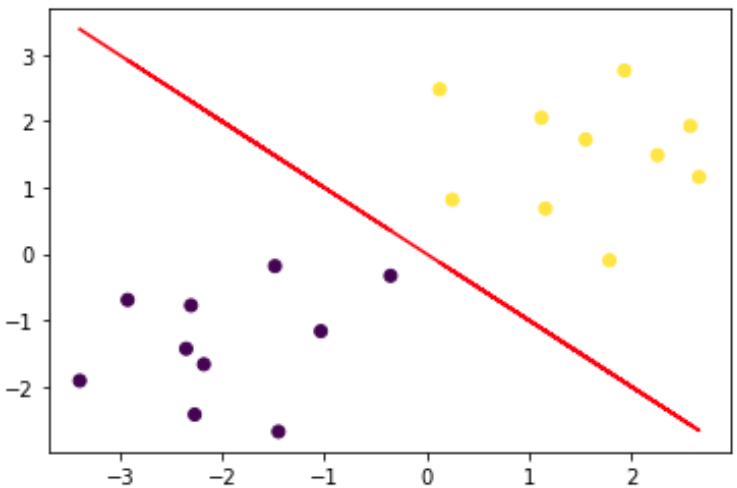


3.4 Support Vector Machine

Support vector machine (SVM) solves the classification problems based on the concept of *separating hyperplanes*.

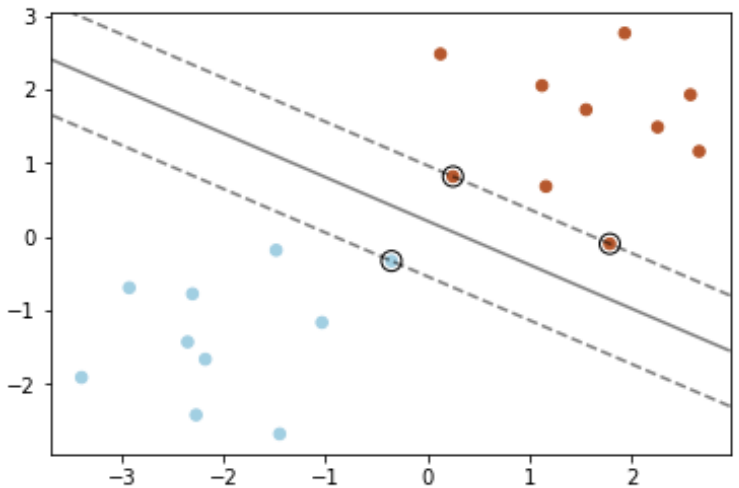
In a p-dimensional space, a hyperplane is described by a equation

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p = 0.$$



3.4 Support Vector Machine

The maximal margin classifier finds a hyperplane separating the two classes that is farthest from the training observations. That is, the minimal distance from the obervation to the hyperplane (margin) is maximized.



3.4 Support Vector Machine

For n training observations $x_1, ...x_n \in \mathbb{R}^p$ and associated class-labels $y_1, ...y_n \in \{-1, 1\}$,

$$\text{maximize}_{\beta_0,\beta_1,...,\beta_p} \quad M$$

subject to $\sum_{j=1}^{p} \beta_j^2 = 1$,

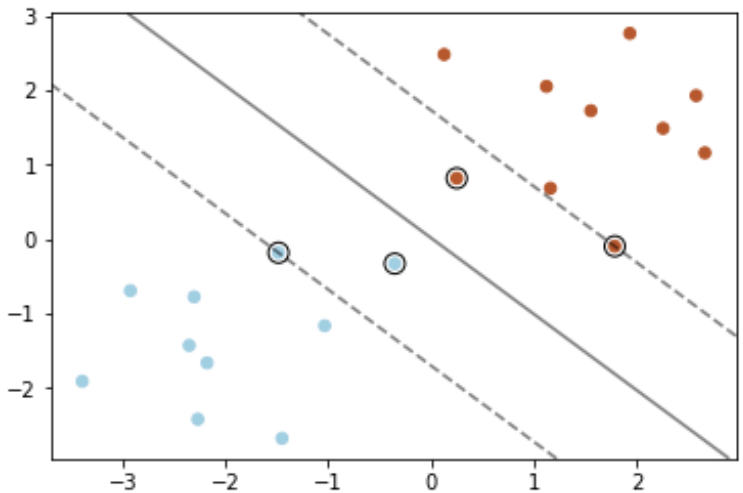
$$y_i(\beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip}) \geq M \quad \forall i = 1, ..., n$$

However, this method suffers from two issues: (1) it cannot handle nonseparable cases; (2) it is sensitive to individual observations.

3.4 Support Vector Machine

The support vector classifier (SVC) allows for a soft margin, which is more robust against individual observations.

The width of the margin is tuned by a parameter C. Only data on or between the margins (support vectors) contribute to the estimate of the hyperplane, thus, wide margin means lower variance and higher bias.



3.4 Support Vector Machine

For n training observations $x_1, ...x_n \in \mathbb{R}^p$ and associated class-labels $y_1, ...y_n \in \{-1, 1\}$,

$$\text{maximize}_{\beta_0,\beta_1,...,\beta_p,\epsilon_1,...\epsilon_n} \quad M$$

subject to $\sum_{j=1}^{p} \beta_j^2 = 1$,

$$y_i(\beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip}) \geq M(1 - \epsilon_i),$$

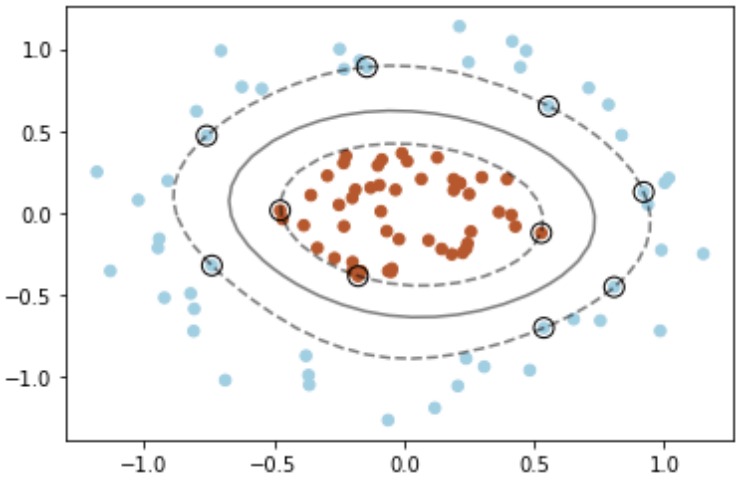
$$\epsilon_i \geq 0, \quad \sum_{i-1}^{n} \epsilon_i \leq C, \quad C \geq 0.$$

3.4 Support Vector Machine

The SVM enlarges the feature space of SVC using *kernels*, and the boundary function has the form

$$f(x) = \beta_0 + \sum_{i \in S} \alpha_i K(x, x_i),$$

where $K(x, x_i)$ takes the form, e.g., $K(x_i, x_{i'}) = < x_i, x_{i'} >$, $(1 + < x_i, x_{i'} >)^d$, $exp(-\gamma ||x_i - x_{i'}||^2)$,...



3.4 Support Vector Machine

Exercise: Fit SVM

```
from sklearn import svm
from sklearn.datasets.samples_generator import make_circles
X, y = make_circles(100, factor=.3, noise=.1)
clf = svm.SVC(kernel='rbf', C=1E6)
clf.fit(X, y)
```

3.5 Cross-validation

Take the SVM case for example, how do we determine which kernel function and parameter C we should choose? Recall that our goal is finding a model that has the lowest *test error rate*. This can be estimated empirically using cross-validation (CV).

Regression: $MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)$.

Classification: $Err = \frac{1}{n}\sum_{i=1}^{n} I(y_i \neq \hat{y}_i)$

3.5 Cross-validation

How to split the data?
We can randomly divide the set of observation (the training set) into k groups (folds) of approximately equal size. The error rate is estimated on one fold with the model fitted on the rest $k-1$ folds. This procedure is repeated k times on each of the k folds.

Regression: shuffle the data and divide.
Classification: stratify the data such that the proportion of each class is approximately the same across the k-folds.
Typically, we use $k = 5$ or $k = 10$.

3.5 Cross-validation

Exercise: Stratify vs. random split

```
import numpy as np
from sklearn.datasets import make_classification
from sklearn.model_selection import StratifiedKFold, KFold
X, y = make_classification(n_samples=1000, n_features=4,
n_informative=2, n_redundant=0, random_state=0, shuffle=False)
skf = StratifiedKFold(n_splits=10)
skf.get_n_splits(X, y)
kf = KFold(n_splits=10)
```

3.6 Error analysis

It is of interest to know how the incorrect labels are assigned. This can be represented by a confusion matrix. For example, if a data point is mis-classified as class 'A' while it actually belongs to class 'B', it is a false positive of 'A'; if it is actually 'A' but is classified as 'B', it is a false negative of 'A'.

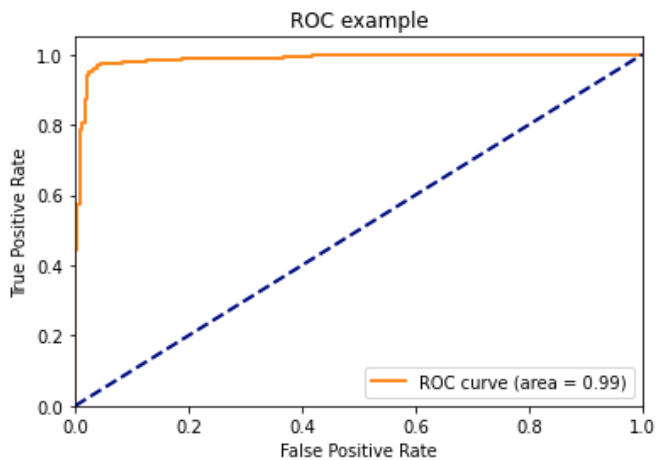
Total=N+P	Pred. pos. (PP)	Pred. neg. (PN)
Act. pos. (P)	True pos. (TP)	False neg. (FN)
Act. neg. (N)	False pos. (FP)	True neg. (TN)

3.6 Error analysis

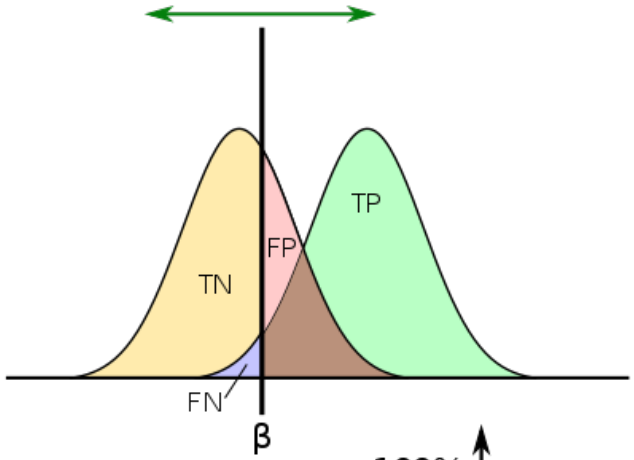
The two types of errors can be displayed simultaneously using a ROC curve, which plots true positive rate (TPR) against false positive rate (FPR).

$$TPR = \frac{TP}{P}, \quad FPR = \frac{FP}{N}.$$

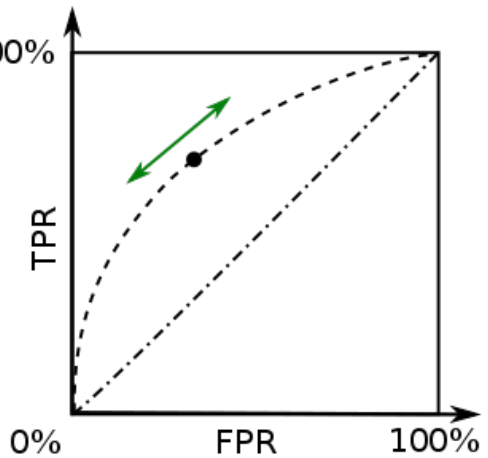
Ideally, we want a model that gives us 100% TP and 0 FP, which is the top left corner of the ROC space.



3.6 Error analysis



TP	FP
FN	TN



3.6 Error analysis

Exercise: ROC curve of LDA classifier

```
from sklearn.metrics import roc_curve, auc
from sklearn.discriminant_analysis import
LinearDiscriminantAnalysis
# assume we obtained training and test data
clf = LinearDiscriminantAnalysis()
clf.fit(X_train, y_train)
y_score = clf.fit(X_train, y_train).decision_function(X_test)
# ROC curve
fpr, tpr, _ = roc_curve(y_test, y_score)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
```

Homework

- Make sure you understand all the exercises above.
- Run through the codes here that should replicate all the figures
  https://github.com/yisiszhang/AdvancedPython/blob/main/colab/Lecture3.ipynb
- Increase the noise of the SVM data, use cross-validation to determine C.
- Report test error of this problem.