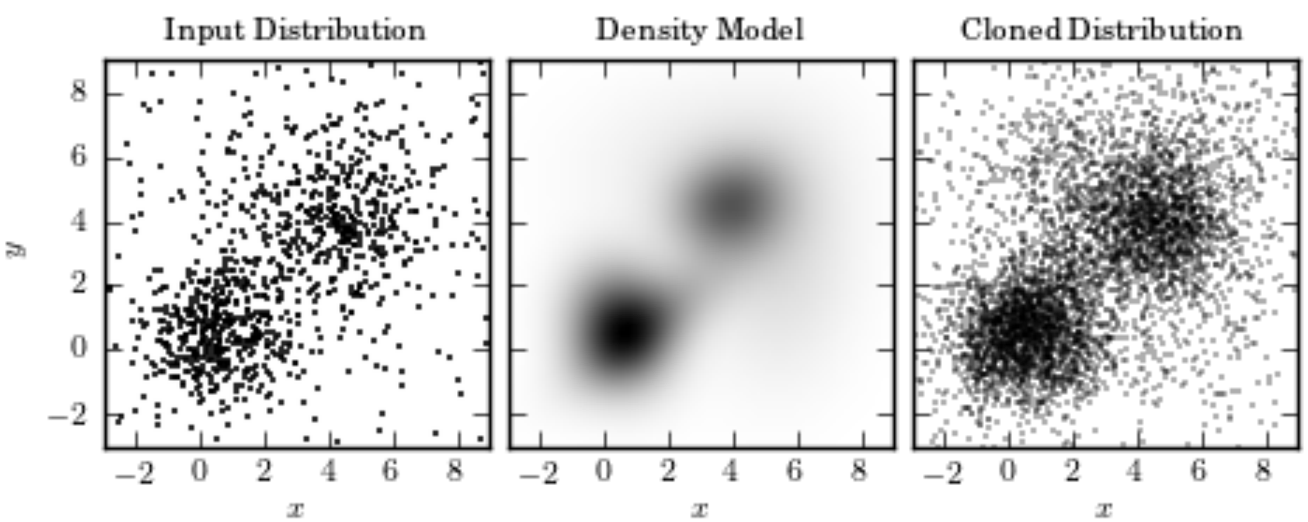
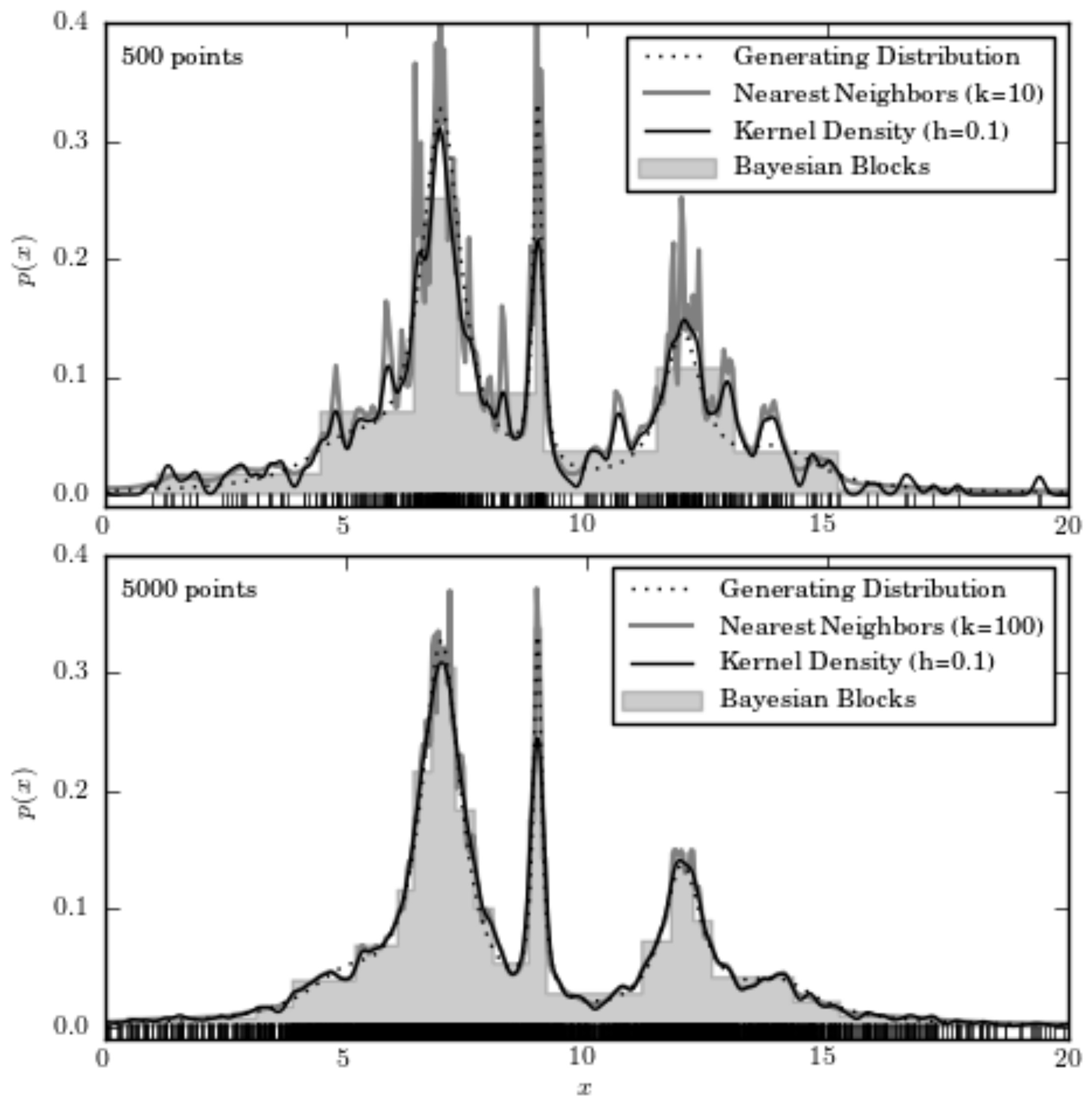
DATA MINING & MACHINE LEARNING:

CLASSIFICATION

CLASSIFICATION: **UNSUPERVISED**

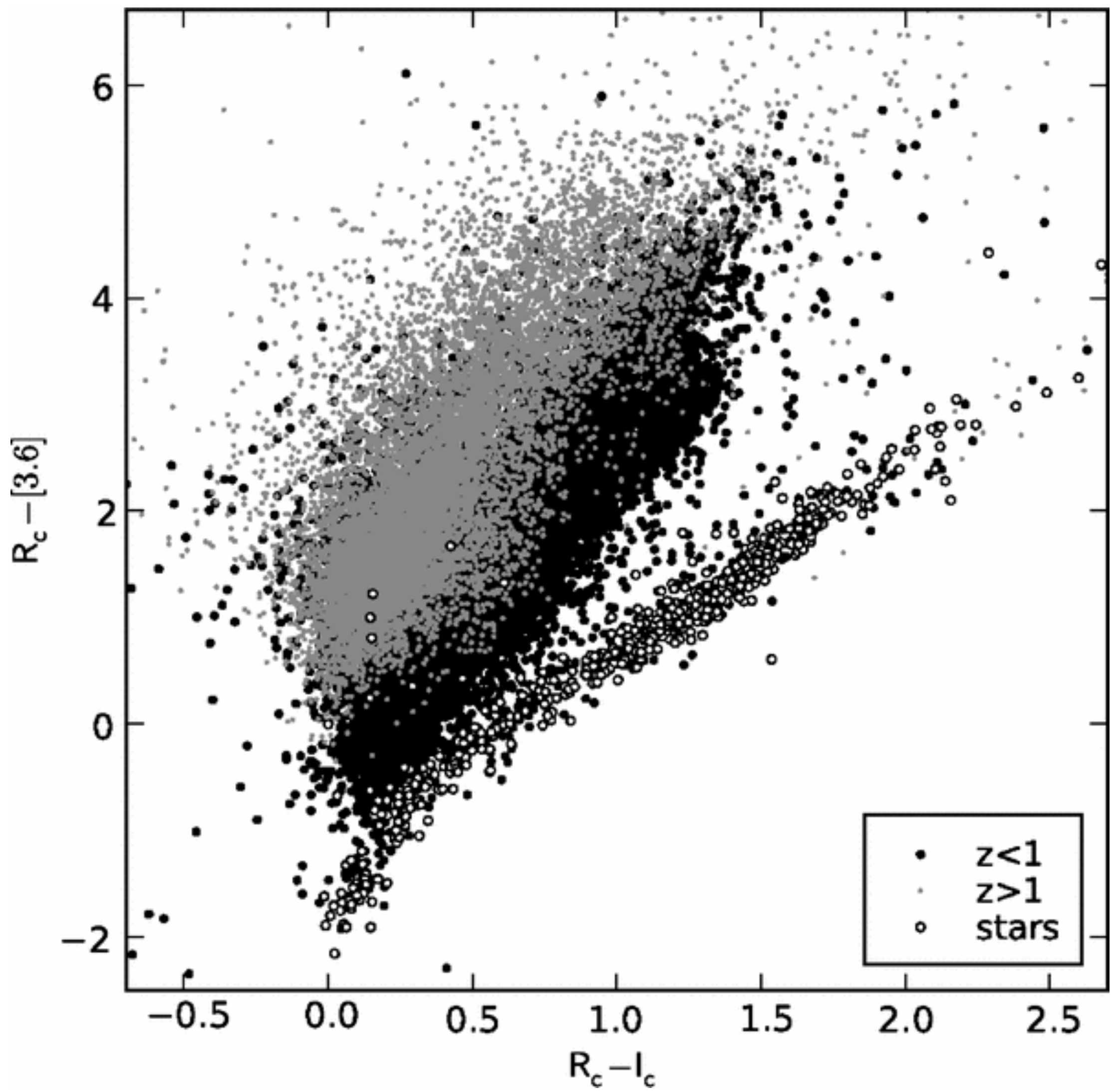
- E.g., determining the inherent structure in data.





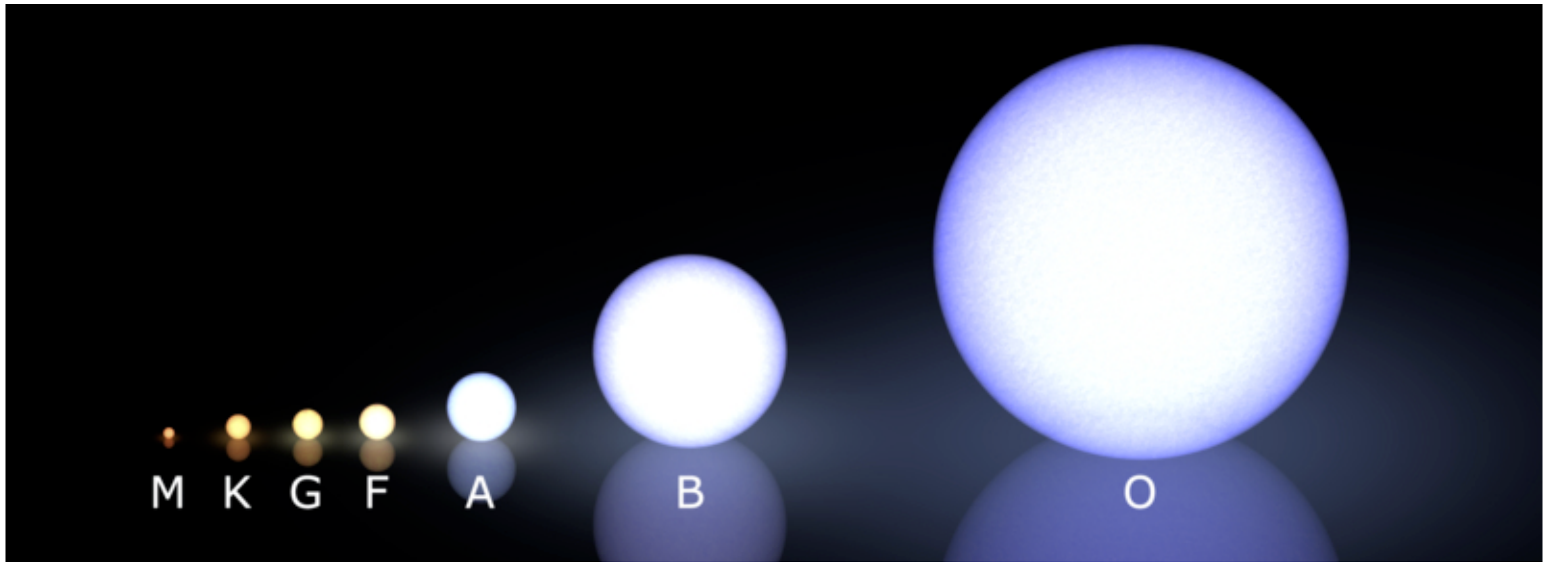
CLASSIFICATION: SUPERVISED

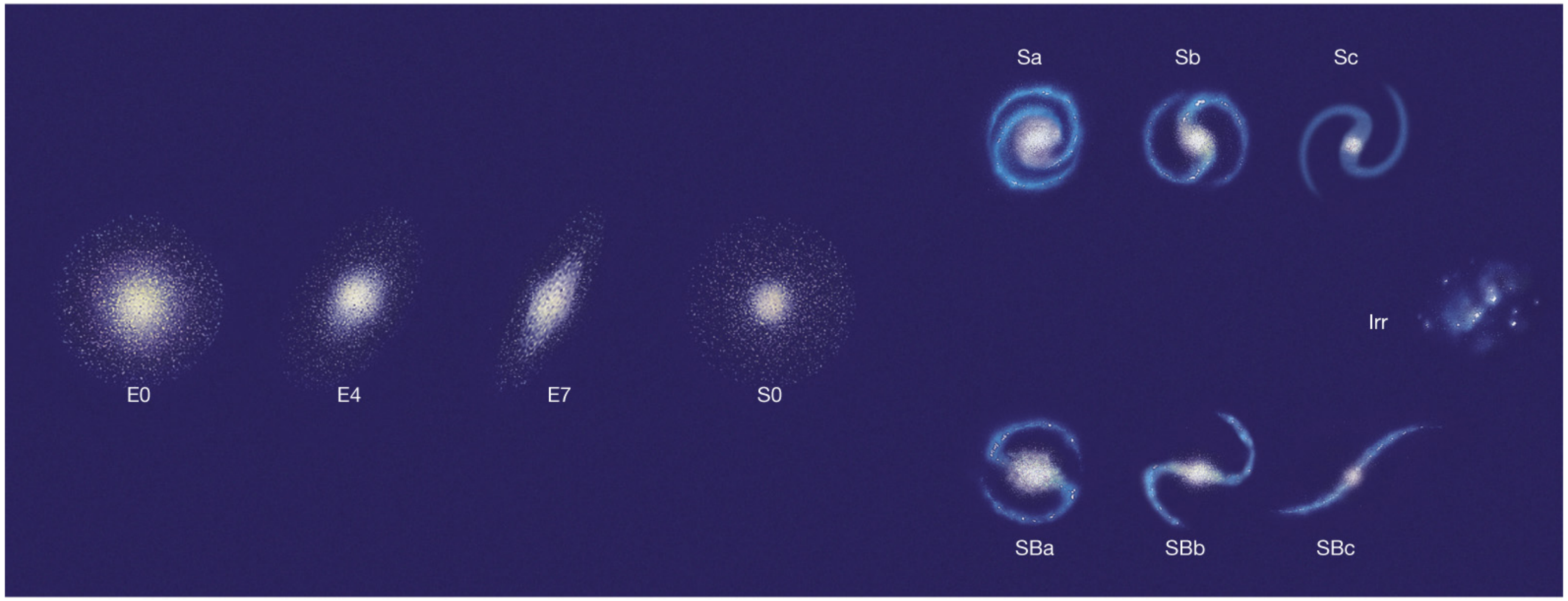
- E.g., using existing classifications to determine relationships between other characteristics / observables in a dataset.



CLASSIFICATION: SUPERVISED

- For more impact, keep it simple!!





CLASSIFICATION: **SUPERVISED**

- Astronomical examples: photometric classification using training set data.
    - RR Lyrae vs. Main-Sequence Stars
    - QSOs vs. stars
    - Photometric redshifts

SUPERVISED CLASSIFICATION:

- **Generative:** Data + models for each class
- **Discriminative:** Data + unsupervised determination of best boundaries defining each class. (Best for high-dimensionality)

SUPERVISED CLASSIFICATION:

- **Zero-one Loss:**
    - Correct classification = 0
    - Incorrect classification = 1
    - Classification Risk = the expectation value of the loss (i.e., *probability of misclassification; error rate*)

SUPERVISED CLASSIFICATION:

- **Completeness:**

$$\frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

- **Contamination:**

$$\frac{\text{false positives}}{\text{true positives} + \text{false positives}}$$

GENERATIVE CLASSIFICATION:

- Bayes' theorem describing the relation between ***y*** labels from ***k*** classes, and features in the data ***x*** with ***N*** points and ***D*** dimensions:

$$p(y_k \mid x_i) = \frac{p(x_i \mid y_k)\, p(y_k)}{\sum_i p(x_i \mid y_k)\, p(y_k)}$$

GENERATIVE CLASSIFICATION:

- Probability of any point having class ***k***:

$$p_k(x) = p(x \mid y = y_k)$$

- The goal is to find the most accurate density estimator for a given classification scheme.

GENERATIVE CLASSIFICATION:

- The regression function:

$$\hat{y} = f(y \mid x)$$

- Discriminant function:

    - The analog of the regression function, where **y** is categorical (i.e., **y={0,1}**)

$$g(x) = f(y \mid x) = \int yp(y \mid x)dy$$

$$= 1 \bullet p(y = 1 \mid x) + 0 \bullet p(y = 0 \mid x)$$

$$= p(y = 1 / x)$$

GENERATIVE CLASSIFICATION:

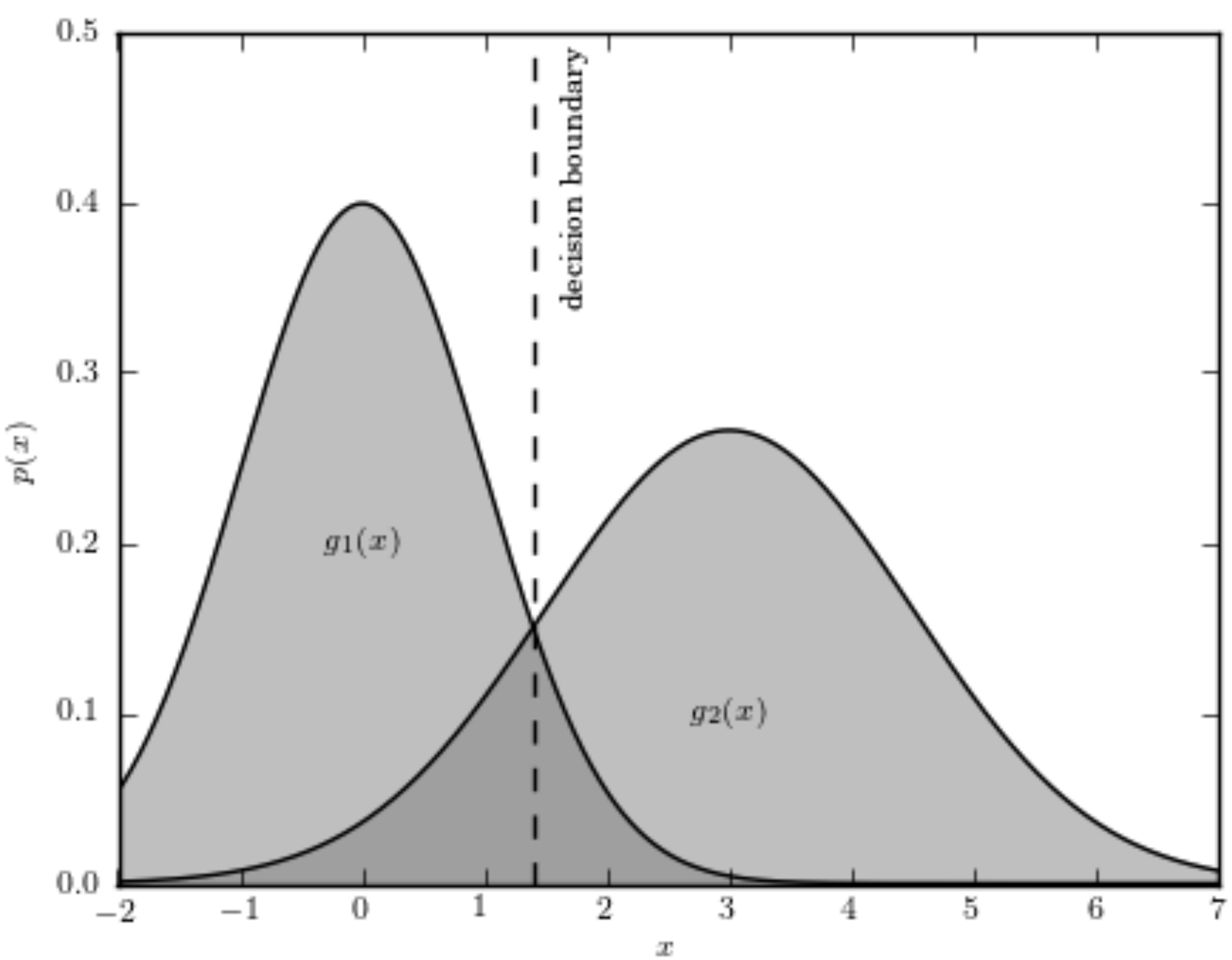
- Applying Bayes' rule to the Discriminant function:

$$g(x) = \frac{p(x \mid y=1)\,p(y=1)}{p(x \mid y=1)\,p(y=1) + p(x \mid y=0)\,p(y=0)}$$

$$= \frac{\pi_1 p_1(x)}{\pi_1 p_1(x) + \pi_0 p_0(x)}$$

GENERATIVE CLASSIFICATION:

- **Bayes Classifier:** Making the discriminant function yield a binary prediction:



$$\hat{\mathbf{y}} = \begin{cases} \mathbf{1} \text{ if } g(x) > 1/2 \\ \mathbf{0} \text{ otherwise,} \end{cases}$$

$$= \begin{cases} \mathbf{1} \text{ if } p(y=1 \mid x) > p(y=0 \mid x) \\ \mathbf{0} \text{ otherwise,} \end{cases}$$

$$= \begin{cases} \mathbf{1} \text{ if } \pi_1 p_1(x) > \pi_0 p_0(x) \\ \mathbf{0} \text{ otherwise.} \end{cases}$$

GENERATIVE CLASSIFICATION:

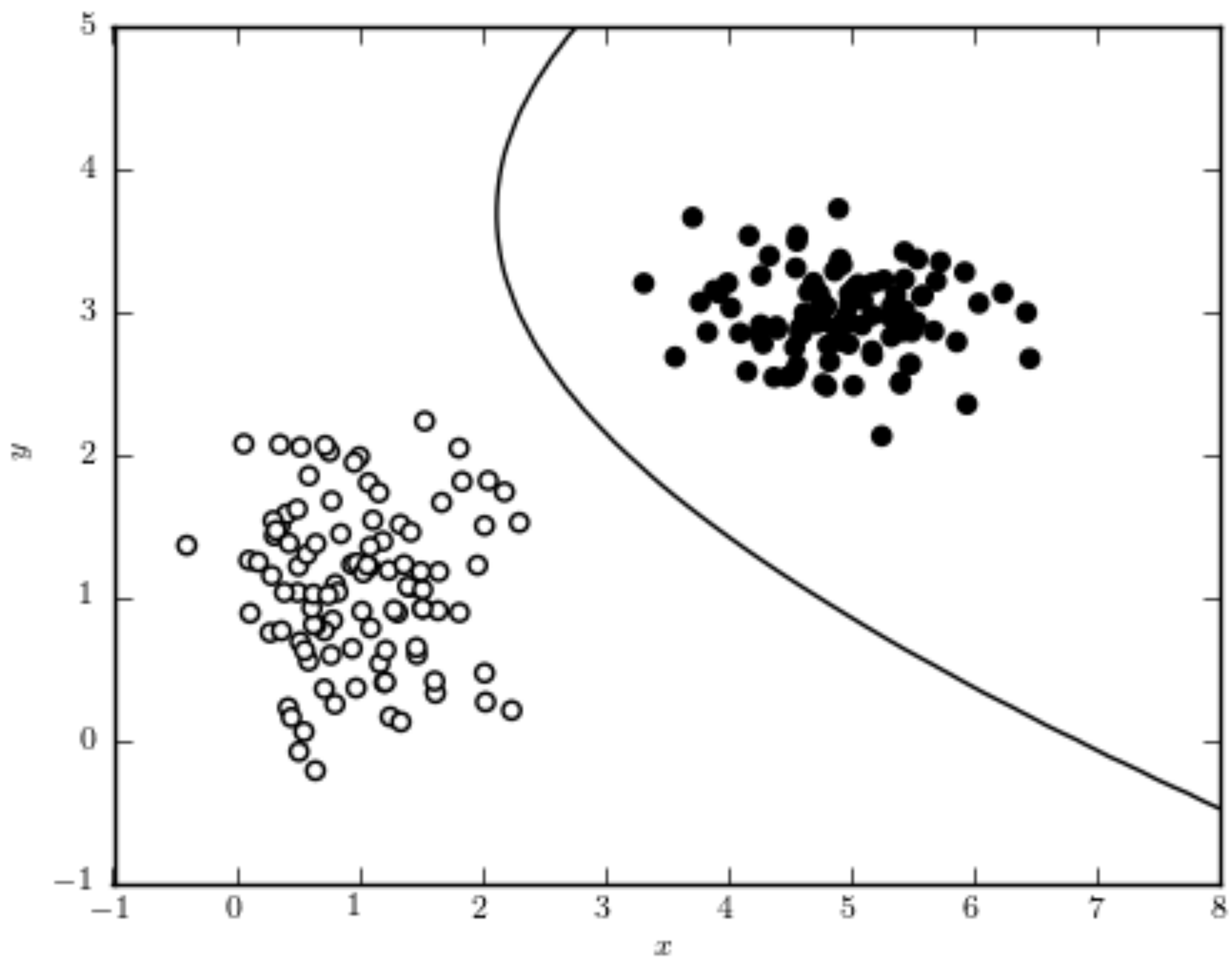
- **Naive Bayes:** Assuming that all of the dimensions of the data are conditionally independent.

$$\hat{y} = \arg\max_{y_k} \frac{\prod_i p_k(x^i)\pi_k}{\sum_j \prod_j p_j(x^i)\pi_j}$$

- … just be careful about running out of parameter space!

GENERATIVE CLASSIFICATION:

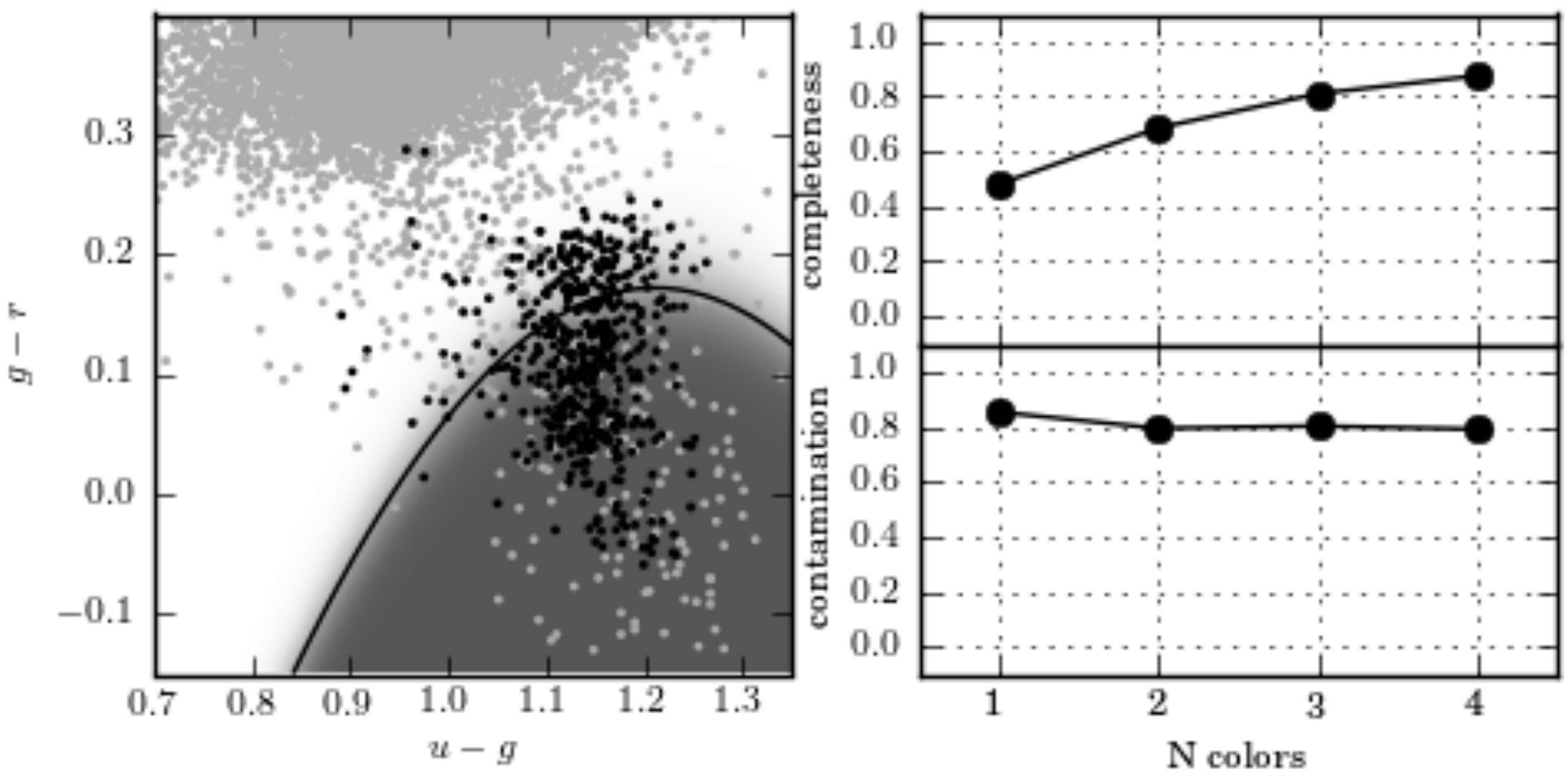
- **<u>Gaussian Naive Bayes:</u>** Assuming that all of the dimensions of the data are conditionally independent, and can be modeled with axis-aligned multivariate Gaussian distributions.



GENERATIVE CLASSIFICATION:

python:
```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X,y)
y_pred = gnb.predict(X)
```

- **<u>Gaussian Naive Bayes:</u>** Assuming that all of the dimensions of the data are conditionally independent, and can be modeled with axis-aligned multivariate Gaussian distributions.

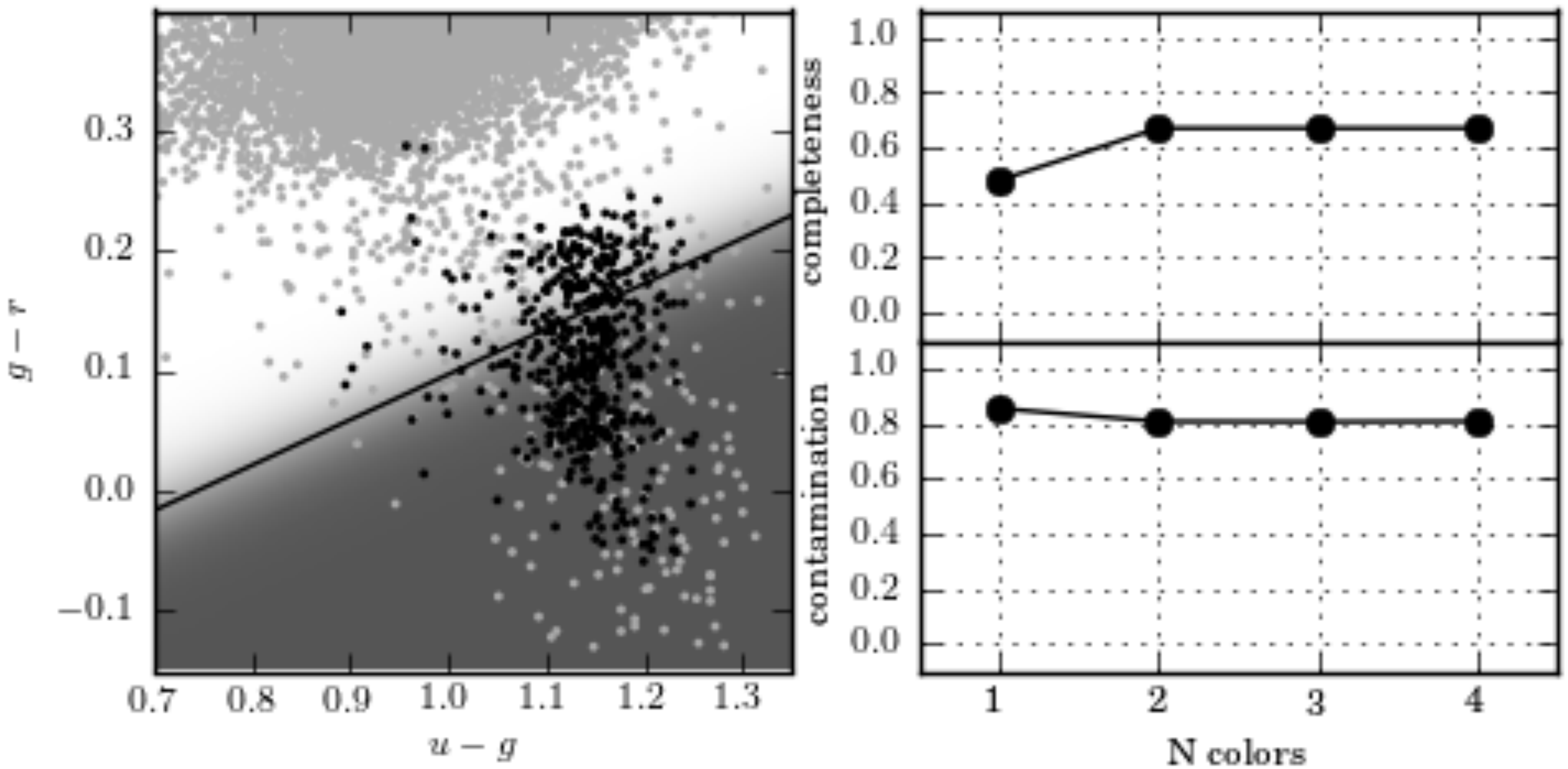


python: from sklearn.lda import LDA

```
lda = LDA()
lda.fit(X,y)
y_pred = lda.predict(X)
```

GENERATIVE CLASSIFICATION:

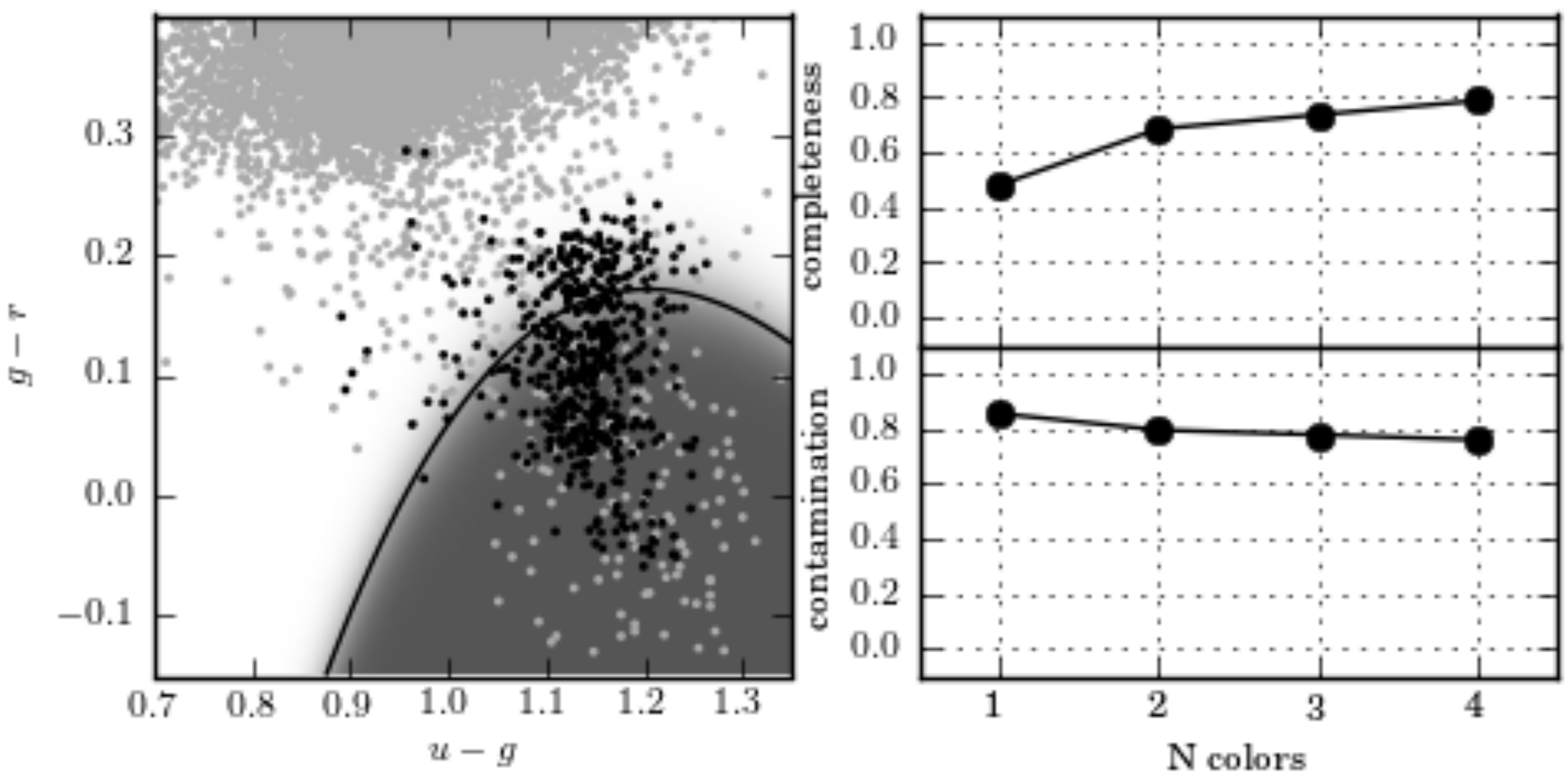
- **Linear Discriminant Analysis (LDA):** Assumes that the distributions have identical covariances for all K classes, making all classes a set of shifted Gaussians and the discriminant a linear function.



```
python: from sklearn.qda import QDA
        qda = QDA()
        qda.fit(X,y)
        y_pred = qda.predict(X)
```

GENERATIVE CLASSIFICATION:

- **Quadratic Discriminant Analysis (QDA):** Relaxes the assumption that the covariances of the Gaussians are constant, making the discriminant a quadratic function.

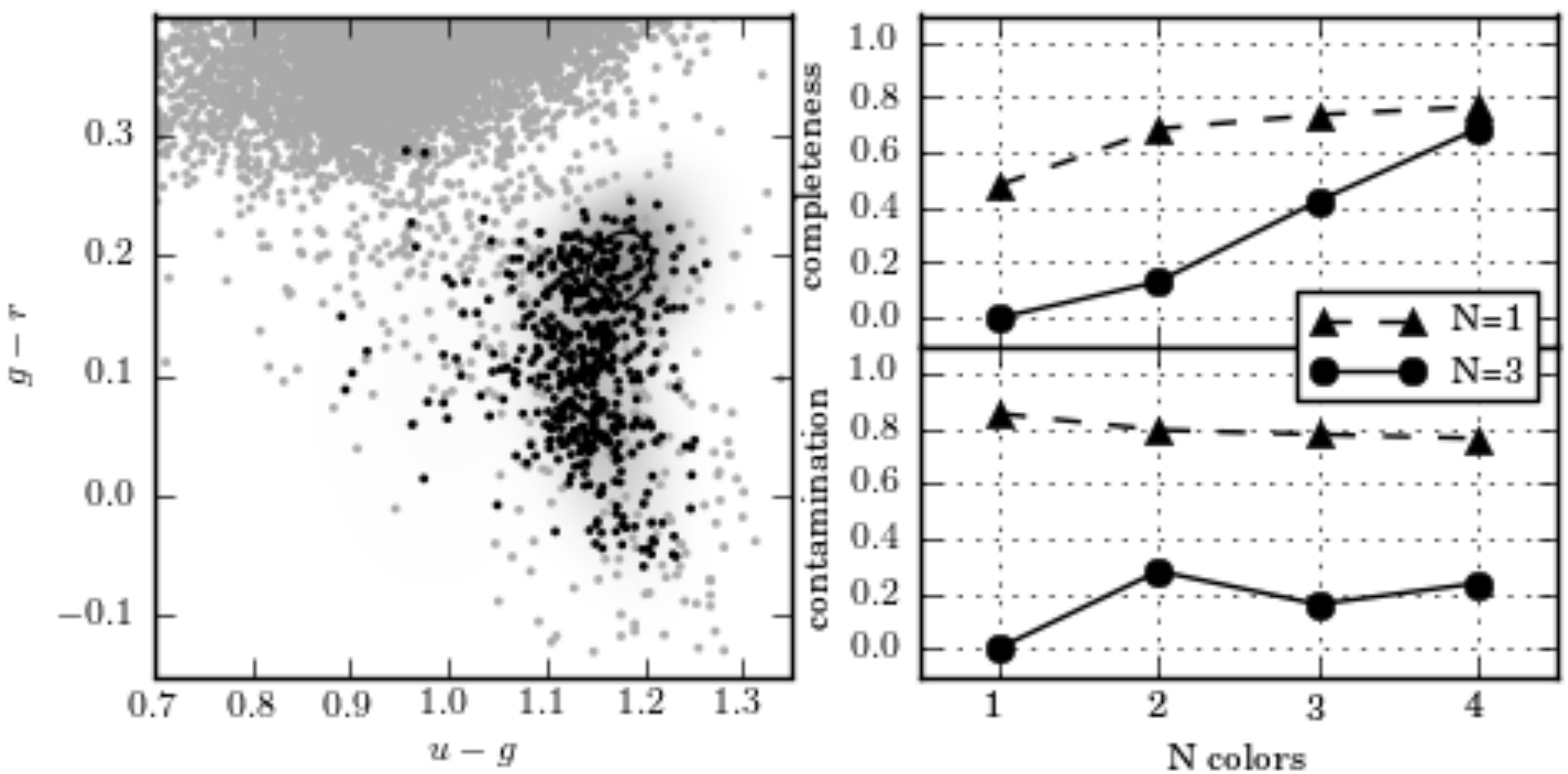


python: from astroML.classification import GMMBayes

```
gmmb = GMM(3) # clusters per class
gmmb.fit(X,y)
y_pred = gmmb.predict(X)
```

GENERATIVE CLASSIFICATION:

- **Gaussian Mixture Model Bayes Classifier:** Force the number of Gaussian components constrained to a simple case. *Using multiple components achieves much lower contamination!*



GENERATIVE CLASSIFICATION:

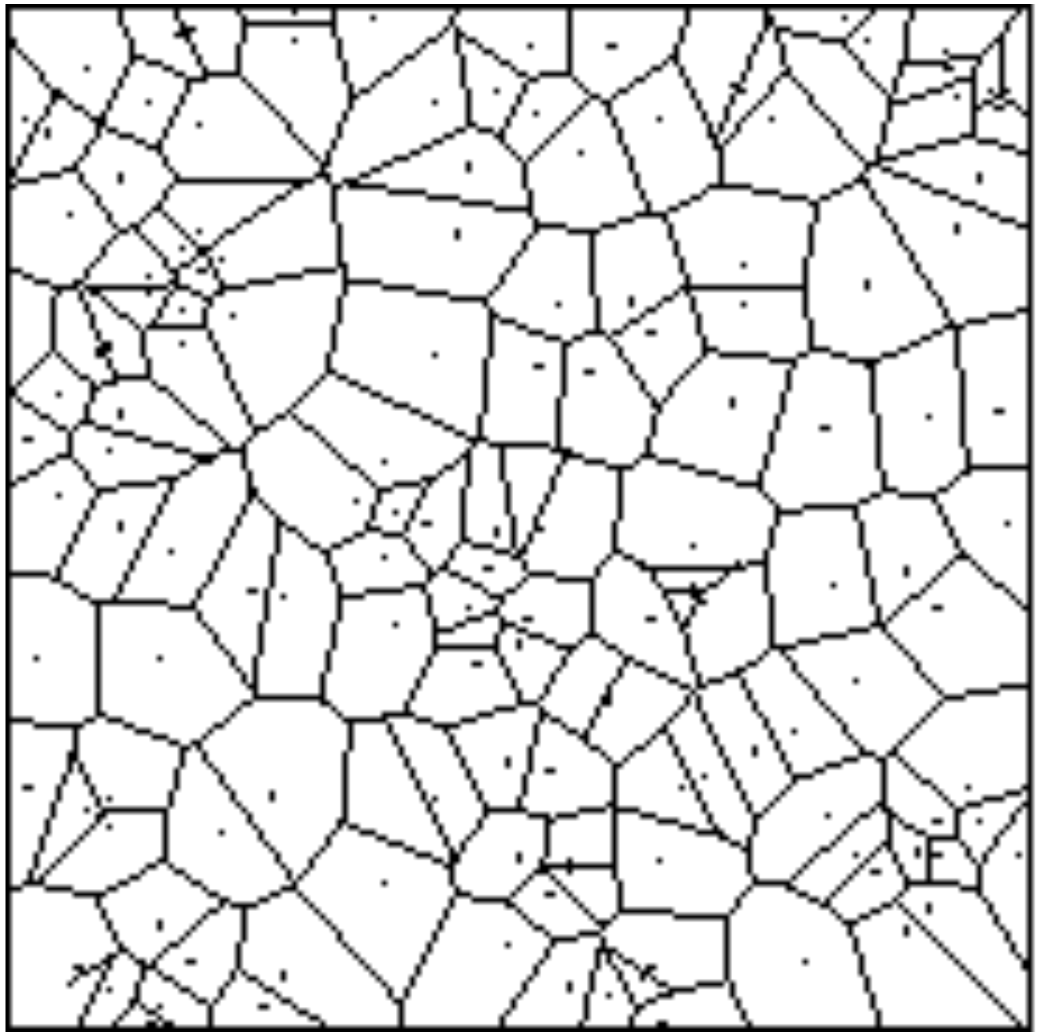
- **Kernel Density Estimation (KDE):** a nonparametric Bayes classifier that models each class with a kernel density estimate; taking GMM to the limit, measuring multiply with a Gaussian kernel centered on each data point.

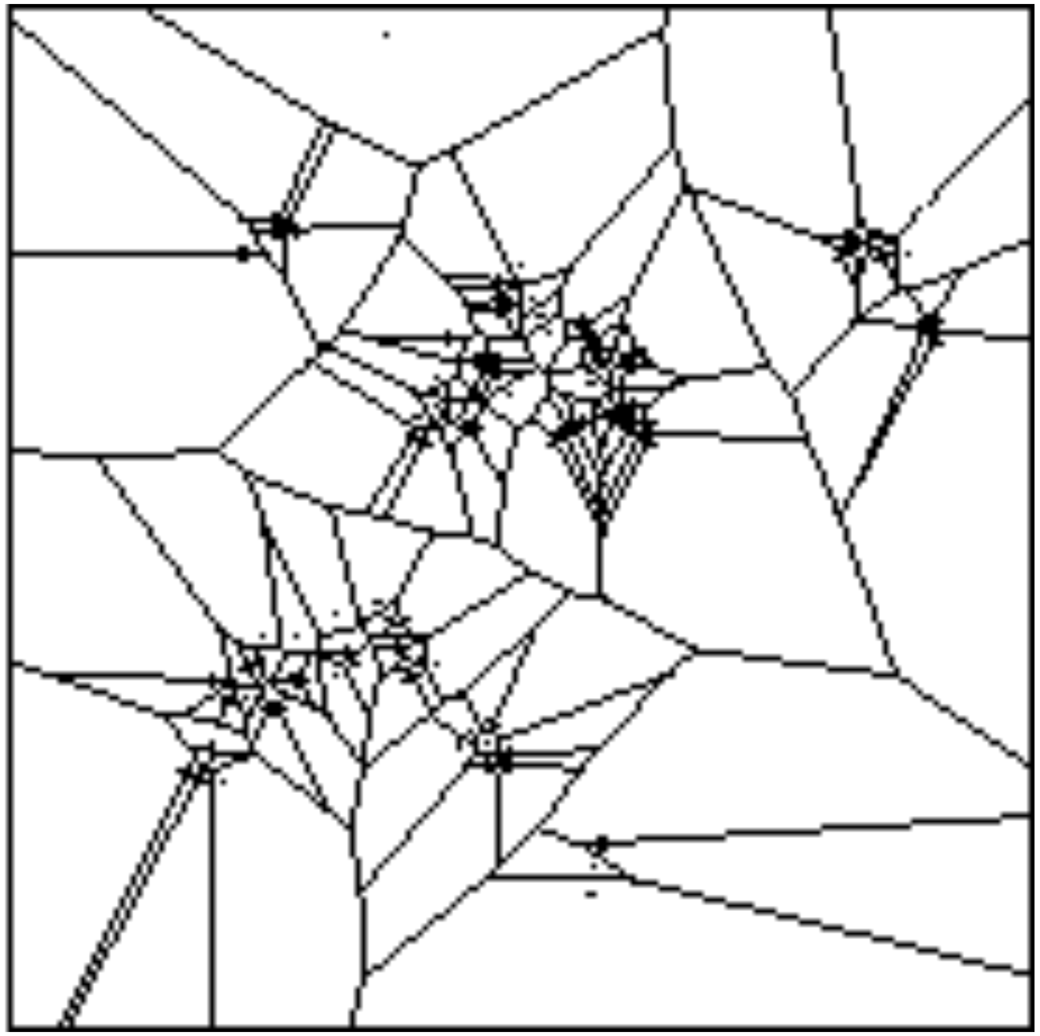
python: from sklearn.neighbors import KNeighborsClassifier

```
knc = KNeighborsClassifier(5) # nearest
knc.fit(X,y)
y_pred = knc.predict(X)
```

GENERATIVE CLASSIFICATION:

- **K-Nearest Neighbor:** a nonparametric Bayes classifier that uses a Voronoi tessellation of the attribute space to produce a decision boundary between the nearest-neighbor points.

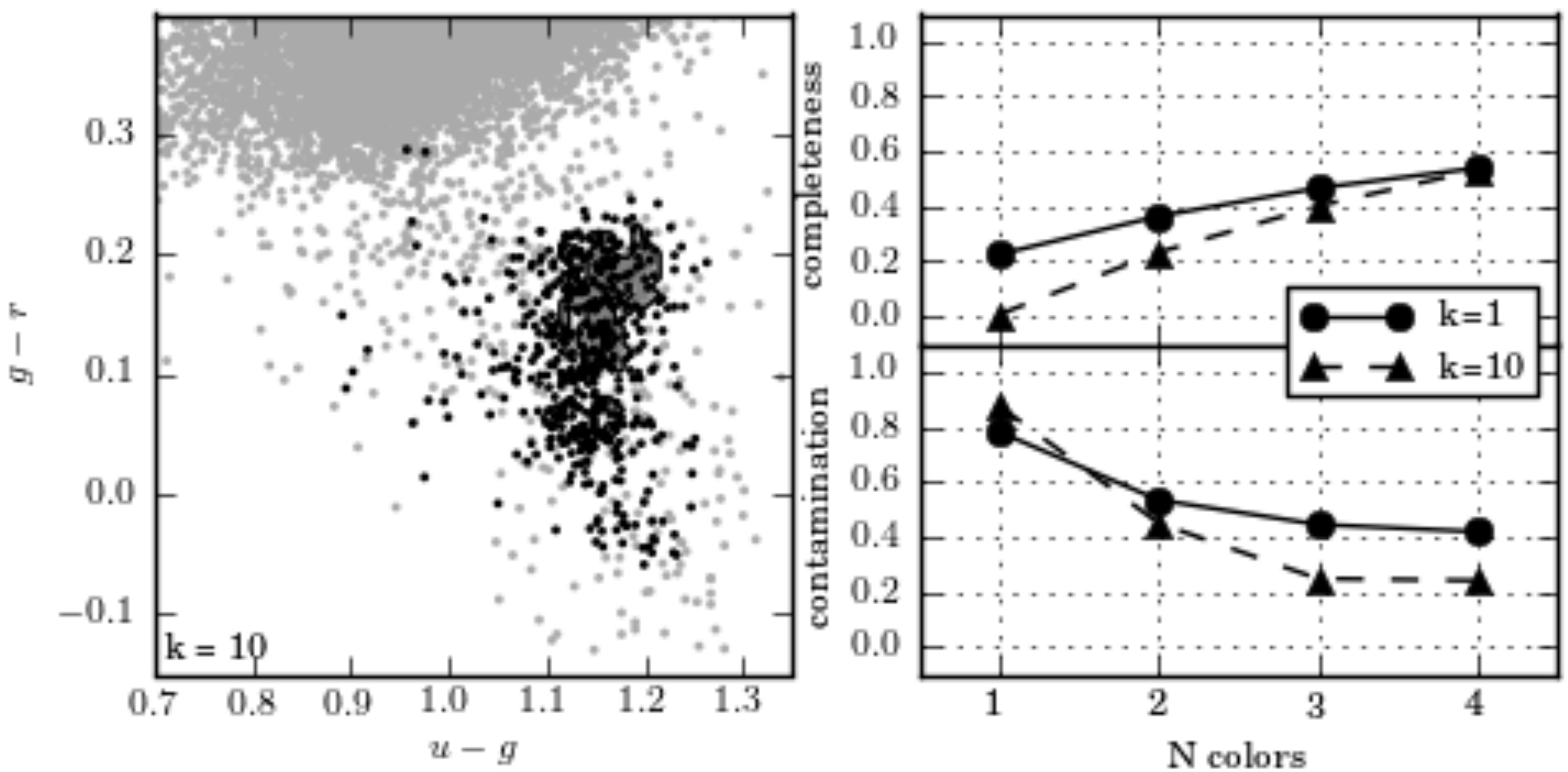




GENERATIVE CLASSIFICATION:

python:
```
from sklearn.neighbors import KNeighborsClassifier
knc = KNeighborsClassifier(5) # nearest
knc.fit(X,y)
y_pred = knc.predict(X)
```

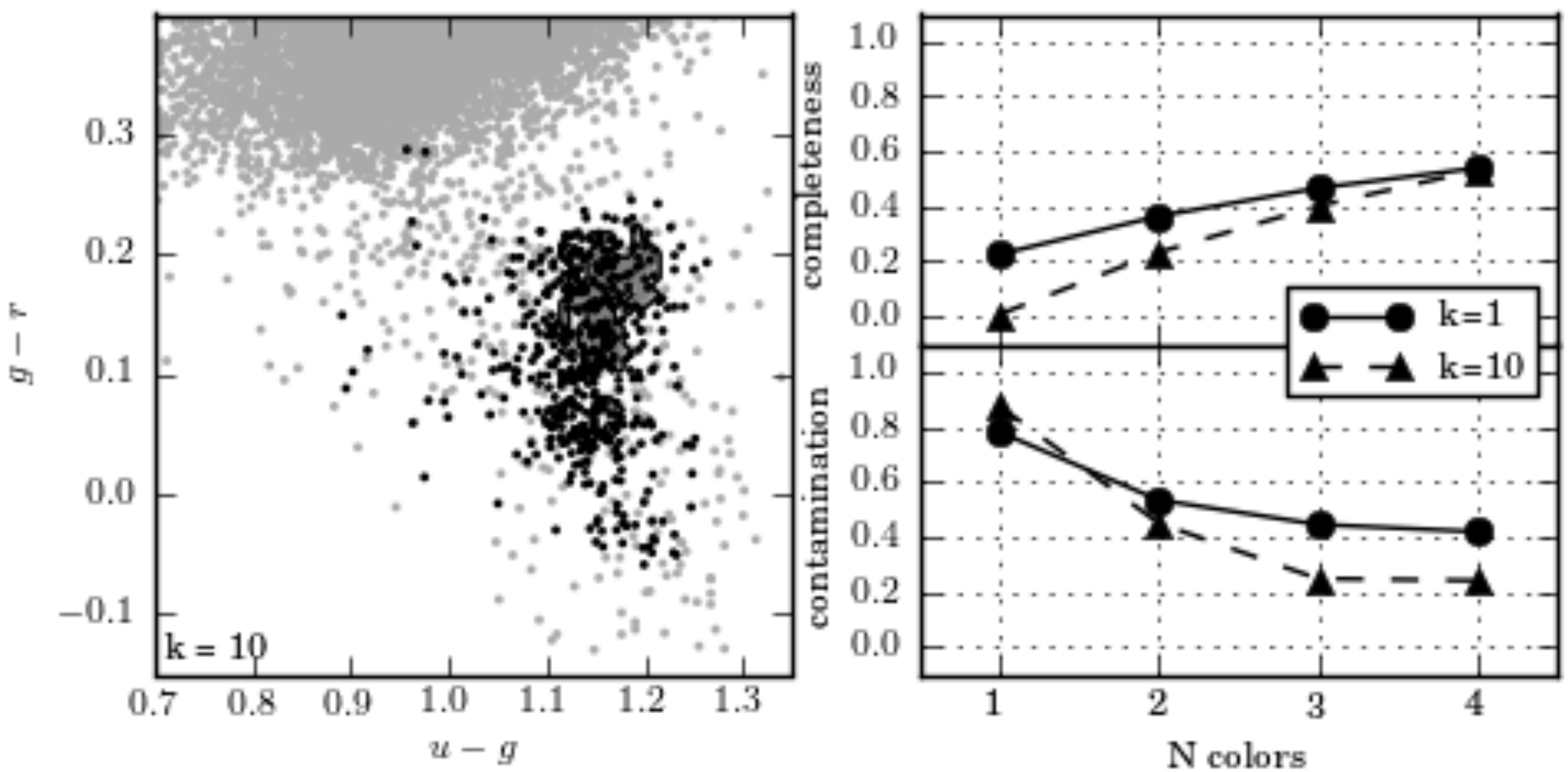
- **K-Nearest Neighbor:** a nonparametric Bayes classifier that uses a Voronoi tessellation of the attribute space to produce a decision boundary between the nearest-neighbor points.



python:
```
from sklearn.neighbors import KNeighborsClassifier
knc = KNeighborsClassifier(5) # nearest
knc.fit(X,y)
y_pred = knc.predict(X)
```

GENERATIVE CLASSIFICATION:

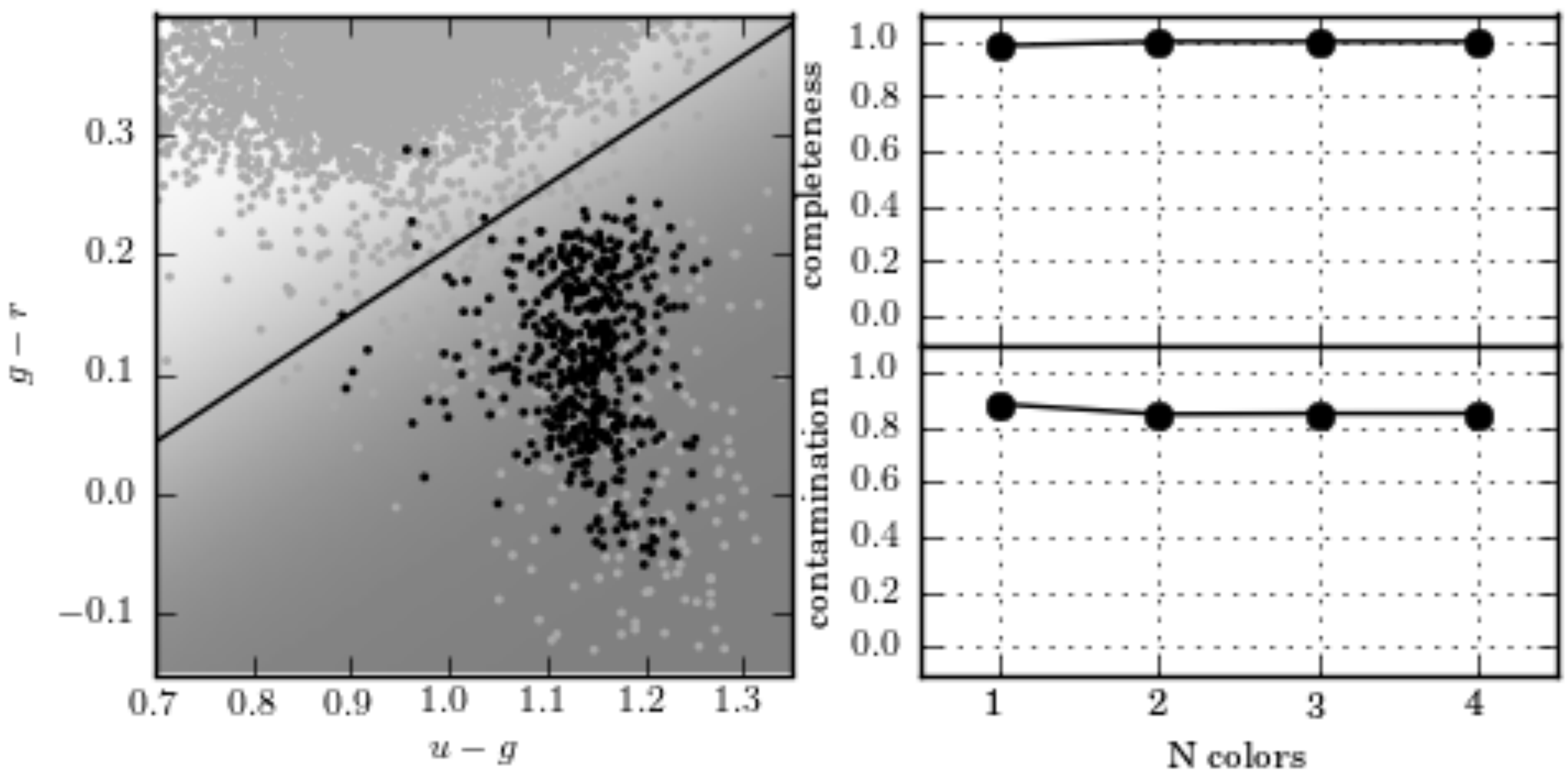
- **K-Nearest Neighbor:** Creates a complex decision boundary, and has high variance when parameter space is under-sampled.



python:
```
from sklearn.linear_model import LogisticRegression
logr = LogisticRegression(penalty='l2')
logr.fit(X,y)
y_pred = logr.predict(X)
```

DISCRIMINATIVE CLASSIFICATION:

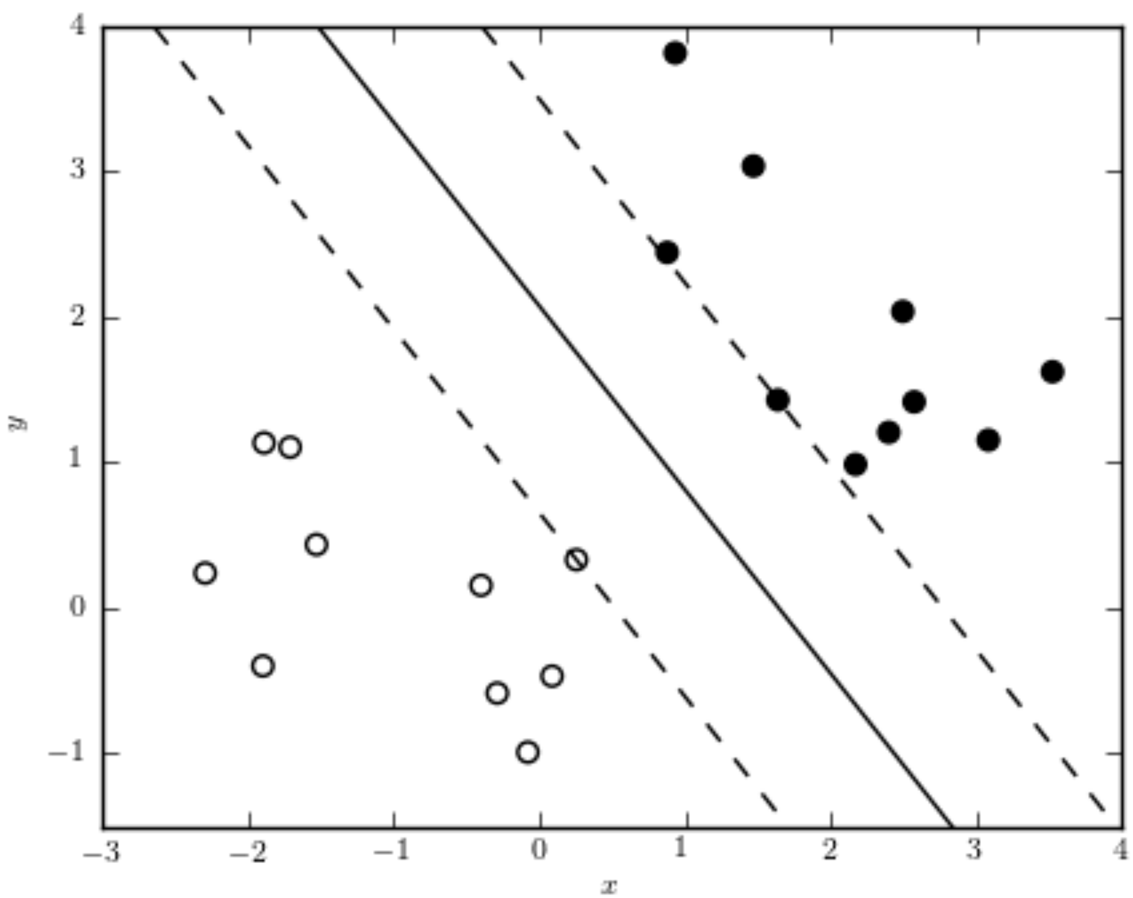
- **Logistic Regression:** model parameters are chosen to effectively minimize classification error rather than density estimation error.



```
python:  from sklearn.linear_model import LogisticRegression
         logr = LogisticRegression(penalty='l2')
         logr.fit(X,y)
         y_pred = logr.predict(X)
```

DISCRIMINATIVE CLASSIFICATION:

- **Support Vector Machines:** hyperplanes that maximize the distance of points from either class ("margin").

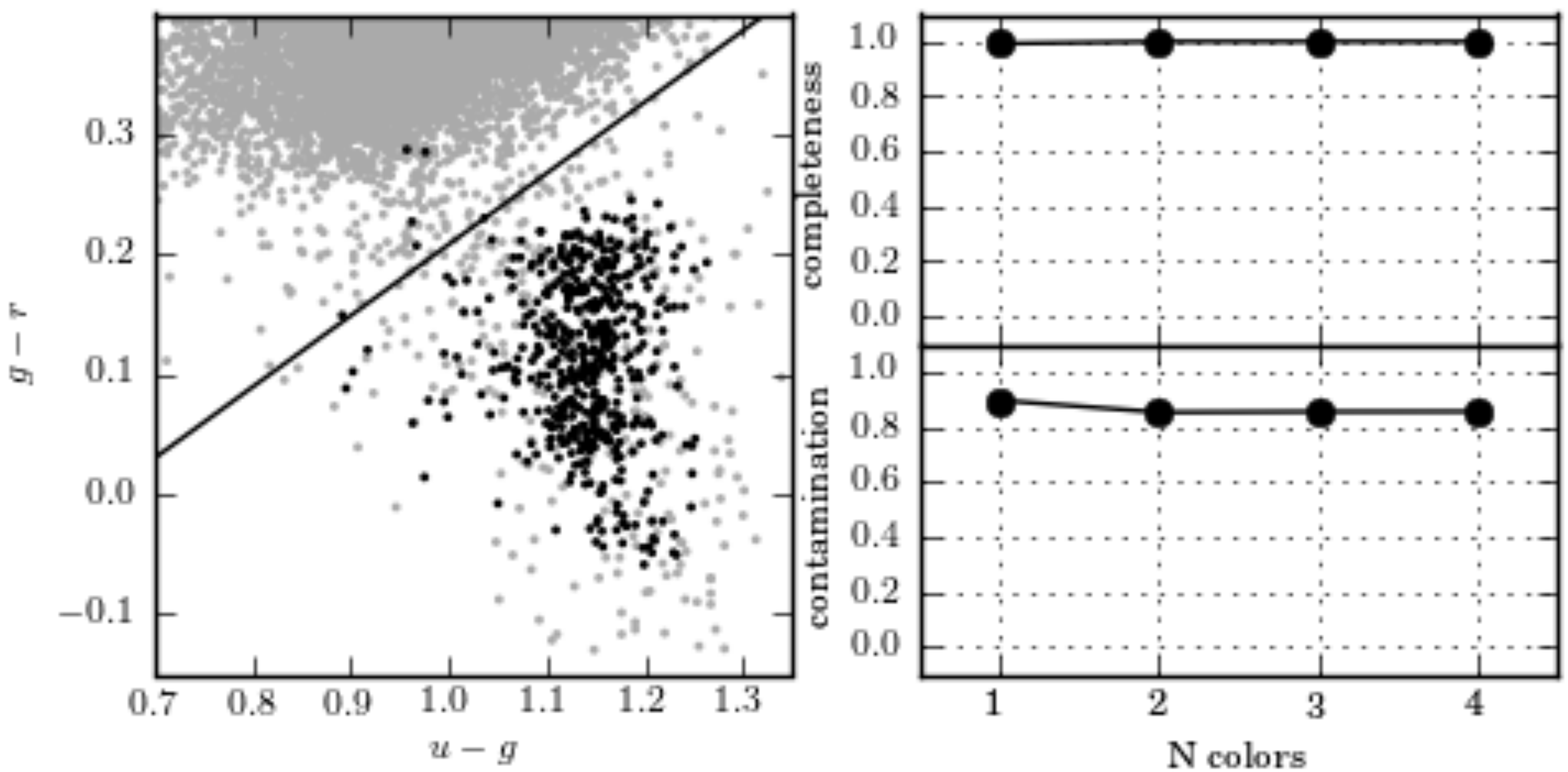


python: from sklearn.linear_model import LogisticRegression

```
logr = LogisticRegression(penalty='l2')
logr.fit(X,y)
y_pred = logr.predict(X)
```

DISCRIMINATIVE CLASSIFICATION:

- **Support Vector Machines:** hyperplanes that maximize the distance of points from either class ("margin"). *Achieves the highest completeness, but with bad contamination.*



python:
```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier(max_depth=6)
model.fit(X,y)
y_pred = model.predict(X)
```

DISCRIMINATIVE CLASSIFICATION:

- **Decision Trees:** Applying decision boundaries hierarchically. *Achieves reasonable completeness, with better control of contamination.*