Hands-on with COMBSS

Step-by-step Python quick-start for the StatFest 2026 talk

Sarat Moka

2026-05-21

Table of contents

1 One-time setup

These steps assume a Mac or Linux laptop with conda installed. If you already have the `combss` conda env from the R version of this handout, skip to **Install the Python packages** below.

1.1 Create an isolated conda environment

```
conda create -n combss python=3.12 -y
conda activate combss
```

The env is named `combss`. Python 3.12 satisfies the `combss` package's requirement of Python ≥ 3.9.

1.2 Install Spyder (Python IDE) inside the env

```
conda install -c conda-forge spyder -y
```

If `menuinst` complains during installation about a missing `spyder-menu.json` — ignore it. Spyder is installed; only the Applications-folder shortcut is missing. Launch with `spyder` from the terminal (env active).

1.3 Install the Python packages

```
pip install combss numpy scipy pandas scikit-learn matplotlib
```

Sanity check inside Spyder (or any Python in the env):

```
import combss
print(combss.linear.model)     # <class 'combss.linear.model'>
```

1.4 Get the Khan SRBCT data (needed for Demo 3)

Run these in a **terminal** (not Spyder):

```
git clone https://github.com/saratmoka/combss-statfest.git
cd combss-statfest
```

In Spyder: **Tools → Preferences → Current working directory → The following directory** and point at the cloned `combss-statfest` folder. (Or just `os.chdir(...)` in the IPython console.)

1.5 Silence sklearn FutureWarnings (run once per session)

The internal solver in `combss` calls `sklearn.LogisticRegression` with the `penalty` argument, which sklearn 1.8 has deprecated. The demos still run correctly, but each call prints a warning. Suppress them once at the top of your Spyder session:

```
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
```

2 Demo 1 — Linear simulation

Goal. Five truly active predictors out of thirty. COMBSS should recover them at $k = 5$. Python is **0-indexed**, so the true support is $\{0, 1, 2, 3, 4\}$.

Note: numpy's `default_rng` and R's `set.seed` use different random number generators, so even with the same nominal seed the simulated data will differ. We use a Python-specific seed below.

```
import numpy as np
from numpy.linalg import lstsq
from combss import linear

rng = np.random.default_rng(2024)
n, p = 300, 30
beta = np.r_[[3, 2, 1.5, 1, 0.5], np.zeros(p - 5)]
X = rng.standard_normal((n, p))
y = X @ beta + 0.5 * rng.standard_normal(n)
itr, iva = slice(0, 200), slice(200, 300)

fit = linear.model()
fit.fit(X[itr], y[itr], X_val=X[iva], y_val=y[iva], q=10, verbose=False)

print("subset_list[:6]:", fit.subset_list[:6])
print("auto-selected subset:", fit.subset)
print("validation MSE at auto-selected k:", round(fit.mse, 4))
```

Python v2 does not expose a validation MSE path directly. Build it yourself to see the curve and pin $k = 5$:

```
import matplotlib.pyplot as plt

mse_path = []
for s in fit.subset_list:
    b, *_ = lstsq(X[itr][:, s], y[itr], rcond=None)
    mse_path.append(float(np.mean((y[iva] - X[iva][:, s] @ b) ** 2)))

plt.plot(range(1, len(mse_path) + 1), mse_path, "o-")
plt.xlabel("k"); plt.ylabel("Validation MSE")
plt.show()
print("subset at k=5:", fit.subset_list[4])     # expect [0,1,2,3,4]
print("MSE at k=5:", round(mse_path[4], 4))     # expect ~0.22
```

3 Demo 2 — High-dimensional logistic ($p \gg n$)

Goal. $n = 200$, $p = 1000$, ten truly active predictors with AR(1) correlation. COMBSS recovers indices $\{0, 1, \ldots, 9\}$ at $k = 10$.

```
import numpy as np
from combss import logistic

rng = np.random.default_rng(2026)
n, p, k0, rho = 200, 1000, 10, 0.5
Sigma = rho ** np.abs(np.subtract.outer(np.arange(p), np.arange(p)))
L = np.linalg.cholesky(Sigma + 1e-8 * np.eye(p))
X = rng.standard_normal((n, p)) @ L.T
beta = np.r_[np.ones(k0), np.zeros(p - k0)]
eta = X @ beta
y = rng.binomial(1, 1 / (1 + np.exp(-eta)))
itr, iva = slice(0, 150), slice(150, 200)

fit_hd = logistic.model()
fit_hd.fit(X[itr], y[itr], X_val=X[iva], y_val=y[iva],
           q=15, verbose=False)

print("subset at k = 10:", fit_hd.subset_list[9])

# True-positive curve (parallel to the R version)
import matplotlib.pyplot as plt
tp_path = [len(set(s).intersection(range(k0))) for s in fit_hd.subset_list]
plt.plot(range(1, len(tp_path) + 1), tp_path, "o-")
plt.axhline(k0, ls="--", color="grey")
plt.xlabel("k"); plt.ylabel("True positives (out of 10)")
plt.show()
```

4 Demo 3 — Khan SRBCT (multinomial, $p = 2308$, 4 classes)

Goal. Pick a small panel of genes that classifies four cancer subtypes on a held-out test set of size 20.

Working directory must contain the `data/` folder from the repo.

```
import numpy as np
import pandas as pd
from combss import multinomial

train = pd.read_csv("data/Khan_train.csv")
test  = pd.read_csv("data/Khan_test.csv")
X_train = train.drop(columns="y").to_numpy()
y_train = train["y"].to_numpy()
X_test  = test.drop(columns="y").to_numpy()
y_test  = test["y"].to_numpy()

fit_khan = multinomial.model()
fit_khan.fit(X_train, y_train, X_val=X_test, y_val=y_test,
             q=20, verbose=False)

print("genes selected:", len(fit_khan.subset))
print("subset:", fit_khan.subset)
print("test accuracy:", round(fit_khan.accuracy, 3))

# Accuracy at each k - refit each subset with sklearn
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression

acc_path = []
for s in fit_khan.subset_list:
    clf = LogisticRegression(max_iter=2000).fit(X_train[:, s], y_train)
    acc_path.append((clf.predict(X_test[:, s]) == y_test).mean())

plt.plot(range(1, len(acc_path) + 1), acc_path, "o-")
plt.xlabel("k (genes)"); plt.ylabel("Test accuracy")
plt.show()
```

5 Demo 4 — Head-to-head with the lasso

Goal. Run COMBSS and `LassoCV` on the **same** simulated data (identical to Demo 1) and compare.

```python
import numpy as np
from sklearn.linear_model import LassoCV
from combss import linear

# Identical data to Demo 1
rng = np.random.default_rng(2024)
n, p = 300, 30
beta = np.r_[[3, 2, 1.5, 1, 0.5], np.zeros(p - 5)]
X = rng.standard_normal((n, p))
y = X @ beta + 0.5 * rng.standard_normal(n)
itr, iva = slice(0, 200), slice(200, 300)

# Lasso
cv_l = LassoCV(cv=10).fit(X[itr], y[itr])
sel_l = np.where(cv_l.coef_ != 0)[0]
mse_l = np.mean((y[iva] - cv_l.predict(X[iva])) ** 2)
print(f"Lasso  : selected = {len(sel_l)}, val_mse = {mse_l:.4f}")

# COMBSS at k=5 (use the manual MSE path from Demo 1 to inspect other k)
fit = linear.model()
fit.fit(X[itr], y[itr], X_val=X[iva], y_val=y[iva], q=10, verbose=False)
from numpy.linalg import lstsq
s5 = fit.subset_list[4]
b, *_ = lstsq(X[itr][:, s5], y[itr], rcond=None)
mse_c = float(np.mean((y[iva] - X[iva][:, s5] @ b) ** 2))
print(f"COMBSS : selected = {len(s5)}, val_mse = {mse_c:.4f}")
```

Expected. Lasso picks roughly a dozen features at validation MSE around 0.25. COMBSS at $k = 5$ recovers exactly $\{0, 1, 2, 3, 4\}$ with validation MSE around 0.22 — the same punchline as the R run.

6 Where to go next

- **Talk companion site** — https://saratmoka.github.io/combss-statfest contains the same demos plus the methodology pages, comparisons against SCAD and MCP, and the rice GWAS application.
- **R version of this handout** — `handout-r.pdf` walks through the same demos in R inside RStudio.
- **Source** — https://github.com/saratmoka/combss-statfest.