

genoselect — User Guide

scikit-learn-compatible genomic prediction and selection

Muhammad Farooqi

2026-07-01

Table of contents

1	Introduction	1
2	Installation	2
3	Data conventions	2
4	Quick start	2
5	Reading your data	3
6	Function reference	3
6.1	Simulation: <code>simulate_population</code>	3
6.2	Relationship matrix: <code>vanraden_grm</code>	3
6.3	GBLUP	3
6.4	Model registry: <code>available_models</code> , <code>make_model</code>	4
6.5	Cross-validation: <code>cross_validate</code> , <code>benchmark</code>	4
6.6	Stacked ensemble: <code>StackedEnsemble</code>	5
6.7	Quality control: <code>qc_markers</code> , <code>impute_markers</code>	5
7	Integration with scikit-learn	5
8	Methods and references	5
9	Citation	6

1 Introduction

`genoselect` is a Python package for **genomic prediction and selection** from SNP marker data. Genomic selection has historically been served by R packages (`rrBLUP`, `BGLR`, `sommer`); `genoselect` brings the same methods to Python through a clean, **scikit-learn-compatible** API, so genomic-prediction models compose naturally with `Pipeline`, `GridSearchCV`, and the rest of the ecosystem.

It implements the VanRaden genomic relationship matrix, GBLUP by REML, a common interface to machine-learning predictors, a stacked super-learner ensemble, and breeding-relevant cross-validation, together with population simulation and quality-control utilities.

2 Installation

```
pip install genoselect
```

From source:

```
git clone https://github.com/mqfarooqi1/genoselect
cd genoselect && pip install -e ".[test]"
```

Requirements: Python 3.10+, NumPy, SciPy, scikit-learn, pandas.

3 Data conventions

- **Genotypes** are a 2-D array of shape (n_individuals, n_markers), coded as **allele dosage** 0, 1, or 2 (copies of the reference allele).
- **Phenotypes** are a 1-D array of length n_individuals.
- Rows are individuals; columns are markers. Missing genotypes are NaN and can be imputed with `impute_markers`.

4 Quick start

```
import genoselect as gs

# 1. simulate a population with a heritable trait
pop = gs.simulate_population(n=250, m=800, n_qtl=40, h2=0.5, seed=1)

# 2. fit GBLUP
fit = gs.GBLUP().fit(pop.geno, pop.pheno)
print(fit.h2_)          # estimated genomic heritability
print(fit.gebv_[:5])    # genomic estimated breeding values

# 3. predict new genotypes
pred = fit.predict(pop.geno[:10])

# 4. benchmark several models by cross-validation
cv = gs.benchmark(pop.geno, pop.pheno, k=5, random_state=1)
print(cv.summary())
```

5 Reading your data

Load real genotype files into a 0/1/2 matrix with one call. All readers return a `GenotypeData` with `.geno` ($n_samples \times n_markers$), `.samples`, and `.markers`, and are pure-Python (no extra dependencies).

```
gd = gs.read_vcf("genotypes.vcf.gz")      # number of ALT alleles
gd = gs.read_plink("dataset")              # reads dataset.bed/.bim/.fam
gd = gs.read_hapmap("data.hmp.txt")       # HapMap text format

fit = gs.GBLUP().fit(gd.geno, phenotypes)
```

- `read_vcf` — biallelic diploid sites; dosage = number of ALT alleles.
- `read_plink` — PLINK 1 binary; dosage = copies of the `.bim` A1 allele.
- `read_hapmap` — dosage = copies of the second listed allele.

Missing genotypes are NaN; clean them with `qc_markers` / `impute_markers`.

6 Function reference

6.1 Simulation: `simulate_population`

```
pop = gs.simulate_population(n=200, m=1000, n_qtl=50, h2=0.5, seed=None)
```

Simulates `n` individuals at `m` biallelic markers. `n_qtl` markers are causal with Normal effects; genetic values are scaled to unit variance and Normal noise is added for the target heritability `h2`. Returns a `Population` with fields `geno`, `pheno`, `bv` (true breeding values), `qtl`, `effects`, and `h2`.

6.2 Relationship matrix: `vanraden_grm`

```
G = gs.vanraden_grm(geno, min_maf=0.0)
```

The VanRaden (2008) genomic relationship matrix $G = WW^T / (2 \sum p(1-p))$, where $W = M - 2p$ is the centred marker matrix. Use `min_maf` to drop low-frequency markers.

6.3 GBLUP

```
fit = gs.GBLUP(min_maf=0.0).fit(geno, y)
fit.predict(geno_new)
```

Genomic BLUP, fitted by REML using the Endelman (2011) spectral method. Useful fitted attributes:

Attribute	Meaning
<code>h2_</code>	Estimated genomic heritability $V_u/(V_u + V_e)$
<code>Vu_</code> , <code>Ve_</code> , <code>lambda_</code>	Genetic / residual variance and their ratio
<code>gebv_</code>	Genomic estimated breeding values (training set)
<code>marker_effects_</code>	RR-BLUP marker effects (for new-genotype prediction)
<code>beta_</code> , <code>intercept_</code>	Fixed-effect estimates

A functional interface mirroring the R package is also available:

```
fit = gs.gblup(y, geno=geno)      # returns a fitted GBLUP
sol = gs.gblup(y, K=G)           # returns the REML solution dict
```

6.4 Model registry: `available_models`, `make_model`

```
gs.available_models()
# ['gblup', 'elastic_net', 'random_forest', 'gradient_boosting', 'ensemble']

model = gs.make_model("elastic_net", random_state=0)
model.fit(geno, y)
model.predict(geno_new)
```

Every model follows the scikit-learn `fit/predict` API on a 0/1/2 marker matrix.

6.5 Cross-validation: `cross_validate`, `benchmark`

```
cv = gs.cross_validate(geno, y,
                      models=["gblup", "elastic_net", "random_forest"],
                      k=5, reps=1, scheme="kfold", random_state=0)
cv.summary()      # tidy DataFrame: model, mean, sd, n_folds
```

Accuracy is the **predictive ability** — the correlation between predicted and observed phenotypes on held-out folds. Use `scheme="leave_group_out"` with a `groups` array (e.g. families or environments) for forward-prediction scenarios. `benchmark(...)` is a convenience wrapper that runs all models.

6.6 Stacked ensemble: StackedEnsemble

```
ens = gs.StackedEnsemble(base_models=None, inner_k=5, random_state=0)
ens.fit(geno, y)
ens.weights_      # non-negative weights, summing to 1
ens.predict(geno_new)
```

A super-learner: out-of-fold predictions from an inner cross-validation are combined by non-negative least squares (van der Laan et al., 2007). Base models default to GBLUP, elastic net, random forest, and gradient boosting.

6.7 Quality control: qc_markers, impute_markers

```
geno_qc, keep = gs.qc_markers(geno, maf=0.05, max_missing=0.1, impute=True)
geno_full = gs.impute_markers(geno)      # mean-impute NaNs
```

7 Integration with scikit-learn

Because the estimators are standard scikit-learn regressors, they slot into the wider ecosystem:

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(gs.GBLUP(), pop.geno, pop.pheno, cv=5,
                        scoring="r2")
```

8 Methods and references

- VanRaden, P. M. (2008). Efficient methods to compute genomic predictions. *Journal of Dairy Science* 91, 4414–4423. doi:10.3168/jds.2007-0980
- Endelman, J. B. (2011). Ridge regression and other kernels for genomic selection with R package rrBLUP. *The Plant Genome* 4, 250–255. doi:10.3835/plantgenome2011.08.0024
- Meuwissen, T. H. E., Hayes, B. J. & Goddard, M. E. (2001). Prediction of total genetic value using genome-wide dense marker maps. *Genetics* 157, 1819–1829. doi:10.1093/genetics/157.4.1819
- van der Laan, M. J., Polley, E. C. & Hubbard, A. E. (2007). Super learner. *Statistical Applications in Genetics and Molecular Biology* 6(1).

9 Citation

Farooqi, M. (2026). *genoselect: scikit-learn-compatible genomic prediction and selection*.
<https://github.com/mqfarooqi1/genoselect>

`genoselect` is released under the MIT License.