EE 467 — Lab 6 API Reference

Ensemble Learning and Random Forest

Colab Setup (pip + dataset unpack)

Docs: `https://colab.research.google.com/`

- Use `%pip install numpy pandas scikit-learn matplotlib seaborn` to install required libraries (if not already installed).
- Use `!tar -xf credit-card.tar.xz` to unpack the dataset archive in Colab.
- Keep `creditcard.csv` in the current working directory so `pd.read_csv("./creditcard.csv")` works.

Pandas (`pandas`)

Docs: `https://pandas.pydata.org/docs/`

- Use `pandas.read_csv(...)` (`https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html`) to load `creditcard.csv`.
- Use `df.drop(["Class"], axis=1).values` (`https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.drop.html`) to create the feature matrix.
- Use `df["Class"].values` to create the label vector (fraud vs. non-fraud).

Train/Test Split (`sklearn.model_selection`)

Docs: `https://scikit-learn.org/stable/modules/classes.html#module-sklearn.model_selection`

- Use `train_test_split(X, y, test_size=..., random_state=...)` (`https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html`) to create `feat_train`, `feat_test`, `y_train`, `y_test`.
- Caution: keep `random_state` fixed so results are reproducible across runs.

Feature Scaling (`sklearn.preprocessing`)

Docs: `https://scikit-learn.org/stable/modules/preprocessing.html`

- Use `StandardScaler()` (`https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html`) to scale `Time` after converting seconds to hour-of-day.
- Use `RobustScaler()` (`https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html`) to scale `Amount` (robust to outliers).
- Use `scaler.fit_transform(arr[:, None])` to scale a single column (scalers expect 2D input).
- Caution: in real ML workflows, fit scalers on **training** only; this lab follows the provided starter code for consistency.

Utility Functions (`lab_6_util`)

Docs: (provided in the lab starter / local module)

- Use `timeit("...")` as a context manager to time code blocks.
- Use `evaluate_model(model, name, X_test, y_test)` to print accuracy, precision, recall, and F1-score.
- Caution: only call `evaluate_model` on a **trained** model (after `fit`).

Baseline Model: Logistic Regression (`sklearn.linear_model`)

Docs: `https://scikit-learn.org/stable/modules/linear_model.html`

- Use `LogisticRegression(max_iter=200)` (`https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html`) to avoid non-convergence warnings on large datasets.
- Use `model.fit(X_train, y_train)` then `evaluate_model(...)`.

Voting Ensemble (`sklearn.ensemble.VotingClassifier`)

Docs: `https://scikit-learn.org/stable/modules/ensemble.html`

- Use `VotingClassifier(estimators=[...], voting="hard")` (`https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html`) to build a majority-vote ensemble.
- For the sub-classifiers, use:
  - `LogisticRegression(max_iter=200)` (docs above),
  - `GaussianNB()` (`https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html`),
  - `DecisionTreeClassifier()` (`https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html`).
- Use `voting_ensemble.fit(X_train, y_train)` then `evaluate_model(...)`.
- Use `voting_ensemble.named_estimators_` to access trained sub-models by name and evaluate each one.
- Soft voting: set `voting="soft"` (requires sub-models that support `predict_proba`).

Bagging (`sklearn.ensemble.BaggingClassifier`)

Docs: `https://scikit-learn.org/stable/modules/ensemble.html#bagging`

- Use `BaggingClassifier(estimator=..., n_estimators=..., max_samples=...)` (`https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html`) to train many copies of the same base estimator on random subsets.
- Typical knobs used in this lab:

  - `n_estimators`: number of sub-classifiers,
  - `max_samples`: fraction (or count) of training samples per classifier,
  - `bootstrap`: whether to sample with replacement,
  - `bootstrap_features` + `max_features`: sub-sample features per classifier.
- Common pattern: create a dictionary `bagging_models = { "setting": model, ... }` and loop over it to `fit` + `evaluate_model`.

Random Forest (`sklearn.ensemble.RandomForestClassifier`)

Docs: `https://scikit-learn.org/stable/modules/ensemble.html#random-forests`

- Use `RandomForestClassifier(n_estimators=...)` (`https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`) to train an ensemble of decision trees (bagging + feature randomness).
- Use `n_jobs=...` to train trees in parallel (speed-up on multi-core machines).
- If the starter defines `N_ENSEMBLE_CPUS = max(os.cpu_count()//2, 1)`, pass `n_jobs=N_ENSEMBLE_CPUS`.

Boosting (`sklearn.ensemble`)

Docs: `https://scikit-learn.org/stable/modules/ensemble.html#boosting`

- Use `AdaBoostClassifier()` (`https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html`) for adaptive boosting (re-weights samples over rounds).
- Use `GradientBoostingClassifier(n_estimators=...)` (`https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifi`
  `html`) for gradient boosting (sequentially fits learners to correct errors).
- Use `fit` then `evaluate_model` to compare boosting models to other ensembles.

Stacking (`sklearn.ensemble.StackingClassifier`)

Docs: `https://scikit-learn.org/stable/modules/ensemble.html#stacking`

- Use `StackingClassifier(estimators=[...], final_estimator=..., n_jobs=...)` (`https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.StackingClassifier.html`) to build a meta-model that learns from base-model predictions.
- Base models used in this lab often include:
  - `RandomForestClassifier(n_estimators=40)` (docs above),
  - `LogisticRegression(max_iter=200)` (docs above),
  - `LinearSVC(max_iter=1500)` (`https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html`).
- Meta-model (final estimator): commonly `LogisticRegression()`.
- Caution: some models (e.g., `LinearSVC`) do not provide `predict_proba`; stacking can use alternative scores internally (per scikit-learn behavior).

Evaluation (`sklearn.metrics`)

Docs: `https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics`

- Use `classification_report(y_true, y_pred)` (`https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html`) to print precision/recall/F1 per class.
- Use `confusion_matrix(y_true, y_pred)` (`https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html`) to inspect TP/FP/FN/TN counts.
- For highly imbalanced fraud detection, focus on **precision/recall/F1** (accuracy alone can be misleading).

Common TODO Patterns in This Lab

Docs: `https://scikit-learn.org/stable/modules/ensemble.html`

- Voting ensemble: create `VotingClassifier` with three named estimators; `fit` once; evaluate ensemble, then evaluate each sub-model via `named_estimators_`.
- Soft voting: switch `voting="hard"` to `voting="soft"` and re-evaluate (requires probability outputs).
- Bagging grid: store multiple `BaggingClassifier` configurations in a dictionary and loop over `fit` + `evaluate_model`.
- Random forest: train two models with different `n_estimators` (e.g., 40 vs 100); use `n_jobs=N_ENSEMBLE_CPUS` to reduce training time.
- Stacking: define base estimators list + meta-estimator; call `fit` once; evaluate the final stacking ensemble on `feat_test`.