Machine learning in Python
Instructor: Jorge Mendez
Logistics

• All lectures will be carried out synchronously and recorded
• The structure of the course will remain largely the same
• Remember that homework 5 is due Friday!
• We’ll release Homework 6 this Friday
  • It will be due Friday next week
Machine learning
What is machine learning?

• Specifically, statistical machine learning
• Divided into supervised learning, unsupervised learning, and reinforcement learning
  • No support for RL in scikit-learn
• High-level: learn to predict *something* from data
Supervised learning: classification

• Given a set of $n$ data points, each with $d$ features/descriptors/attributes
  • E.g., a set of images represented by their pixel values as features
• Assume they are all collected in a matrix $X_{\text{train}} \in \mathbb{R}^{n \times d}$
• Each data point corresponds to a class
  • E.g., is there a car in the image or not
• The points are matched with labels collected in a vector $y_{\text{train}} \in \{+1, -1\}^n$ or $y_{\text{train}} \in \{0, 1\}^n$
• Goal: learn a function $f$ such that $\hat{y} = f(x_{\text{test}})$ is good at predicting values for $y_{\text{test}}$ corresponding to $x_{\text{test}} \notin X_{\text{train}}$
  • $f$ is known as a classifier
Supervised learning: regression

• Similar setup with a matrix $X_{\text{train}} \in \mathbb{R}^{n \times d}$ and a vector of labels $y_{\text{train}}$
• The labels are now real numbers: $y \in \mathbb{R}^n$
  • E.g., what is the location of the car in the image
• Goal: learn a function $f$ such that $\hat{y} = f(x_{\text{test}})$ is as close as possible to $y_{\text{test}}$ corresponding to $x_{\text{test}} \notin X_{\text{train}}$
  • $f$ is known as a regressor
Unsupervised learning

• Given a data set $X \in \mathbb{R}^{n \times d}$ with no labels
• Goal: discover some structure in the data
• Clustering
  • Find out a set of $k$ clusters of similar points
• Dimensionality reduction
  • Find a related “data set” $Z \in \mathbb{R}^{n \times k}$ with $k \ll d$ such that $Z$ is a good lower dimensional representation for $X$
How to solve ML?

• How do we choose the form of $f$?
  • It can’t be any arbitrary function (or how would we search for it?)

• How do we search for the “right” $f$ once we fix its form?
  • Often treated as an optimization problem

• The answers to these questions lead to a wide variety of machine learning algorithms
ML algorithms in scikit-learn
scikit-learn APIs

- scikit-learn uses a standard set of functions for all regression and classification models
- The two main ones for our purposes
  - `model.fit(X, y)` — train the model with the given data set
  - `model.predict(X_test)` — get predictions for the given test set

Note: the math in the following slides is for intuitive purposes and does not necessarily match the exact formulations used by scikit-learn
Linear models
Linear regression (or Ordinary Least Squares)

• Assumption: linear model

\[ f(x) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_d x_d \]
  - Vectorized: \( f(X) = X \cdot w \)
  - How? Need to add a column of all-ones

• Optimization problem: \( \min_w \sum_{i=1}^n (f(x_i) - y_i)^2 \)
  - Vectorized: \( \min_w \|Xw - y\|_2^2 \)
Linear regression in scikit-learn

• `model = linear_model.LinearRegression()` — create model

• Arguments
  • `fit_intercept=True` — whether to fit $w_0$.
    • *Important!* Many data sets include a column of all-ones. If that is the case, you should set `fit_intercept=False`
  • `normalize=False` — whether to normalize each feature by subtracting the mean and dividing by the l2-norm.
    • Ignored if `fit_intercept=False`. Why?
  • `copy_X=True` — if `False`, X is not copied and may be overwritten
Linear regression in scikit-learn

• Methods
  • `fit(X, y)` — find the optimal weights
  • `predict(X)` — return the predicted $y$
  • `score(X, y)` — return the coefficient of determination ($R^2$)

• Attributes
  • `coef_` — the weights $w$
  • `intercept_` — the intercept or bias term $w_0$
Live Example
Related: Ridge regression

• OLS assumes the features are linearly independent
  • Else, numerical instabilities

• Solution: solve \( \min_{w} \|Xw - y\|^2_2 + \alpha \|w\|^2_2 \) with \( \alpha > 0 \) → Ridge regression
  • \texttt{model} = \texttt{linear\_model.RidgeRegression(alpha=1.0)}
  • Greater \( \alpha \) places more emphasis on reducing the weight values
  • \( \alpha \) is called a \textit{regularization} parameter
Related: LASSO

- OLS assumes all features are relevant
  - What if we have more features than we need?
- Solution: solve $\min \| Xw - y \|_2^2 + \alpha \| w \|_1$ with $\alpha > 0 \rightarrow$ LASSO
  - `model = linear_model.Lasso(alpha=0.1)` or `linear_model.LassoLars(alpha=0.1)`
  - Induce sparsity in the resulting weights
How to set $\alpha$?

• Find the one that performs best
• We care about *generalization* performance: how does our model perform on *test* data
• Problem: should *never* use actual test data to select optimal $\alpha$
  • Statistically invalid
• Solution: k-fold cross validation
k-fold cross validation

• Partition the *training* set into *k* folds
• Treat each fold in turn as the test set
  • I.e., train on all data *except* one fold, then evaluate on this fold
• Find the average test error (e.g., square error) across all folds
• Repeat for many values of $\alpha$ and pick the best one
k-fold cross validation

• This is a general method for picking any algorithm hyper-parameter
• More generally, it is a method for estimating the test error
• The greater $k$ is (up to $n$), the better the estimate (but the more expensive it is to run!)
10-fold cross validation example
k-fold cross validation in scikit-learn

• For ridge regression
  • `reg_model = linear_model.RidgeCV(alphas, cv=None)`
  • Pass in a list of $\alpha$ values to try
  • `cv=None` — pass in the number of folds ($k$)
    • If `None`: leave-one-out cross validation ($k = n$)
  • `reg_model.alpha_` — the best $\alpha$

• For LASSO
  • `reg_model = linear_model.LassoCV(alphas, cv=None)`
  • `cv=None` does 3-fold cross validation
Live Example
Logistic regression

• Linear model for *classification*

• Unlike with linear regression, we can’t assume \( y = X \cdot w \)
  • Why?

• “Equivalent” assumption: log-odds are linear
  • \( p = P(y = 1) \)
  • \( \log\frac{p}{1-p} = X \cdot w \)
  • If \( Xw > 0 \), then \( p > 1 - p \) so we set \( \hat{y} = 1 \)

• Optimization problem: maximize the log-likelihood of the data
  • Won’t bore you with the details
Logistic regression in scikit-learn

• `model = linear_model.LogisticRegression()` — create model

• Arguments
  • `fit_intercept=True` — whether to fit $w_0$.
  • `penalty='l2'` — use ‘l2’ (default) or ‘l1’ penalty, or ‘none’
  • `C=1.0` — Regularization parameter, inverse of $\alpha$ in linear regression
Logistic regression in scikit-learn

• Methods
  • `decision_function(X)` — predict the confidence scores for the samples
  • `fit(X, y)` — find the optimal weights
  • `predict(X)` — return the predicted y
  • `predict_proba(X)` or `predict_log_proba(X)` — return the predicted probability or log-probability estimates
  • `score(X, y)` — return the average accuracy over test data
Logistic regression in scikit-learn

• Attributes
  • coef_ — the weights $w$
  • intercept_ — the intercept or bias term $w_0$
  • classes_ — the observed classes in the data set
Live Example
Support Vector Machines
Why do we need non-linearities?
Adding non-linearities to functions

• What if the true function (e.g., decision boundary) is non-linear?
• A simple trick: feature transformations
  • $X_{\text{new}} = \text{non-linearity}(X_{\text{old}})$
• E.g., add all quadratic features
  • Time and space complexity: $O(d^2)$
  • Worse for higher-order polynomials
• Solution: kernel trick
  • High level: do not compute feature transformations, but learn model on transformed features directly
Support Vector Machines

- Primarily a classification method, but applicable to regression
- Incorporates the kernel trick, so complex non-linearities can be added
  - Cool trick: Gaussian kernel corresponds to *infinite* features
  - Any data set becomes linear with a sufficiently fine-grained Gaussian kernel
  - Note: it is not always a good idea to make your data perfectly linearly separable!
Support Vector Machines

- Adds another constraint: focuses only on points near the decision boundary
SVMs on scikit-learn

- `model = svm.SVC()` — create SVM classification model
- Arguments
  - `C=1.0` — regularization parameter, as in logistic regression
  - `kernel='rbf'` — the type of kernel to use
    - `'rbf'` — Gaussian or radial basis function kernel
    - `'linear'` — Introduces no non-linearities
    - `'poly'` — polynomial non-linearities
  - `degree` — of the polynomial in `'poly'`
  - `gamma` — smaller gamma means more complex non-linearities for `'rbf'`
  - `probability=False` — whether to enable probability estimates
  - `decision_function='ovr'` — how to handle multi-class classification
    - `'ovr'` — one-vs-rest: one model for each class
    - `'ovo'` — one-vs-one: one model for each pair of classes (quadratically many!)
SVMs on scikit-learn

• Methods — same as with logistic regression
  • Probability estimates only available if `probability=True`

• Attributes
  • `coef_` and `intercept_` — only for linear kernel
  • `support_` and `support_vectors_` — indices and values for the support vectors (the points closest to the decision boundary)
  • `n_support_` — number of support vectors for each class
Live Example
Clustering
Clustering in scikit-learn

- 10 different algorithms implemented in scikit-learn
- Different assumptions about the structure of the clusters
- Our focus: k-means
K-means clustering

• Assign each of \( n \) points to one of \( k \) clusters
• Goal: minimize the within-cluster squared distance
• Each cluster \( j \) consists of a centroid \( \mu_j \) and a set of assigned points \( C_j \)
• K-means is a simple two-step iterative method

Repeat until convergence:
1. For all \( j \): Set the centroid \( \mu_j \) to the mean of the assigned points \( C_j \)
2. For all \( i \): Assign the point \( x_i \) to the cluster \( j \) with the nearest centroid \( \mu_j \)
K-means clustering in scikit-learn

• model = cluster.KMeans()

• Arguments
  • n_clusters=8 — number of clusters
  • init='kmeans++' — how to initialize the clusters ('kmeans++' is better)
  • n_init=10 — number of different runs (best run is picked)
K-means clustering in scikit-learn

• Methods
  • `fit(X)` — run k-means clustering
  • `fit_predict(X)` — run k-means clustering and return cluster labels per sample
  • `fit_transform(X)` — run k-means clustering and transform X to cluster distances
  • `predict(X)` — return cluster labels per sample
  • `score(X)` — negative sum of distances to cluster centroids
  • `transform(X)` — transform X to cluster distances
    • A simple form of dimensionality reduction
K-means clustering in scikit-learn

• Attributes
  • `cluster_centers_` — array of coordinates of the cluster centers
  • `labels_` — array of cluster assignments
  • `inertia_` — sum of squared distance to centroid (negative of score())

• Note: scikit-learn also implements mini-batch K-means, which approximates K-means but is faster for large data sets
Live Example
Takeaways

• scikit-learn provides a set of common APIs for training multiple ML models

• It is just as easy to do regression, classification, or clustering

• Linear and logistic regression for linear models

• SVMs for linear or non-linear models

• K-means clustering for separating un-labeled data into groups