5 Basic Practice (part 2 of 3)

Learning Algorithm Selection

Consider these ideas:

- **Explainability**: If you must be able to explain how predictions are made, consider $k$NN, linear regression, and decision tree algorithms. (This may sacrifice performance.)

- **Memory**: If data do not fit in memory or arrive in an ongoing stream (e.g. stock market information), consider an incremental learning algorithm.
  
e.g. A model trained by stochastic gradient descent (§4) can run another iteration to shift the model toward a new training example.

- **Data set size**: Neural networks (STAT 453) and gradient boosting (coming in §7) can use many examples and features. SVM and others cannot handle large data sets as well.

- **Categorical vs. numeric features**: Choose a suitable algorithm for your feature types, or convert features to the right type (e.g. one-hot encoding or binning, above).

- **Nonlinearity**:
  
  - If data are linearly separable, consider SVM with the linear kernel.
  
  - If a linear model is suitable, consider linear or logistic regression.
  
  - Otherwise consider a deep neural network or ensemble method (§7).

- **Training speed**: Neural networks are slow to train. Regression and decision trees are faster. Random forests (§7) are parallelizable.

- **Prediction speed**: SVMs, regression, and (some) neural networks are fast at prediction. $k$NN, ensemble algorithms (§7), and deep or recurrent neural networks can be slow.

Use a validation set (next) to help decide among these ideas.

To learn more:


Three Data Sets: Training, Validation, Test

Randomly shuffle examples and split into three subsets consisting of, e.g., 80%, 10%, and 10% of the available data.

- Use training data to set parameters of an algorithm (or of several algorithms).
- Use validation data to choose the best hyperparameters for your algorithm (or to choose the best algorithm and its hyperparameters).
- Use test data to evaluate your chosen model (once, as more than once is cheating).

Other proportions may be useful, e.g. 70/15/15 for a small data set or 95/2.5/2.5 for a large one.

Python

- from sklearn.model_selection import train_test_split

  train_test_split(*arrays, test_size=None, random_state=None, stratify=None)

  - returns 2*len(arrays) split indexables, one train/test pair per input indexable
  - uses the integer size or float proportion in test_size, 0.25 by default, for test outputs
  - has reproducible results if random_state is set to an integer, e.g. random_state=0
  - splits in a stratified way, giving each split ≈ the same proportion of samples of each class, if stratify is set to vector of class labels; e.g. stratify=y

e.g. Split X into X_train and X_test while splitting y into y_train and y_test:

X_train, X_test, y_train, y_test = train_test_split(X, y)

e.g. To get an 80/10/10 split, split 80/20 and then split the 20 into 10/10:

X = np.arange(100) # quick fake data: X = 0 to 99 and y = 90 zeros, 10 ones
y = np.concatenate([np.full(shape=90, fill_value=0),
                    np.full(shape=10, fill_value=1)])

# split 80% training data, 20% "_tmp" for validation & test
X_train, X_tmp, y_train, y_tmp = train_test_split(X, y,
                                                    test_size=.2, random_state=0, stratify=y)

# of remaining 20%, split in half to get 10% validation, 10% test
X_valid, X_test, y_valid, y_test = train_test_split(X_tmp, y_tmp,
                                                    test_size=.5, random_state=0, stratify=y_tmp) # try without random_state, stratify

print(f'X_train={X_train},
      X_valid={X_valid},
      X_test={X_test}')

To learn more:


1 Nobody cares about training performance as such because no model beats just memorizing the data.
2 Good luck.
3 A parameter *args can receive a sequence of positional arguments (and **kwargs can receive keyword arguments).
4 Indexables include list, numpy array, and DataFrame.
Underfitting and Overfitting

- A model \textit{underfits} (or has \textit{high bias}) when it makes many mistakes on training data because:
  - the model is too simple for the data
  - the engineered features are not informative enough

- A model \textit{overfits} (or has \textit{high variance}) when it fits training data but not new data because:
  - the model is too complex for the data
  - the model has too many features relative to \( N \)

To address overfitting:
- try a simpler model
- reduce the dimensionality of the data (coming in §9)
- add more training data
- regularize (below)

Regularization

Regularization addresses overfitting by inducing a simpler model, often resulting in a \textit{bias-variance tradeoff} in which bias increases but variance decreases, improving accuracy on unseen examples.

- \textit{L1} regularization adds a penalty term \(|w| = \sum_{j=1}^{D} |w^{(j)}|\), the \( L1 \) norm.

- \textit{L2} adds a penalty term \(|w|^2 = \sum_{j=1}^{D} (w^{(j)})^2\), the square of the \( L2 \) norm.

Recall (§3) that ordinary least squares (OLS) finds \( \min_w b \frac{1}{N} \sum_{i=1}^{N} [f_w, b(x_i) - y_i]^2 \). Regularized regression models include:

- \textit{Lasso}\(^5\) regression uses \( L1 \) regularization, finding \( \min_{w,b} \left( \frac{1}{N} \sum_{i=1}^{N} [f_{w,b}(x_i) - y_i]^{2} + \alpha |w| \right) \), where \( \alpha \geq 0\(^6\)
  - \( \alpha = 0 \implies \) the model is not regularized.
  - \( \alpha \) large causes \textit{feature selection}, yielding a \textit{sparse} model with most \( w^{(j)} \) set to zero, leading to underfitting and explainability\(^7\)

\(^5\)“Lasso” refers to “least absolute shrinkage and selection operator.”
\(^6\)Burkov uses \( C \). I use \( \alpha \) to match scikit-learn. scikit-learn defines \( w \) to exclude the intercept \( w_0 = b \) and then says we find \( \min_w \left( \frac{1}{N} \|Xw - y\|^2 + \alpha |w| \right) \). I think it is referring variables transformed to have zero mean.
\(^7\)Page 71 of The Elements of Statistical Learning (Hastie, Tibshirani, and Friedman) shows a geometric argument for the \( 2D \) case. The least squares error function \( \text{MSE} \) has elliptical contours centered at \( \mathbf{w}^* = (w_1^*, w_2^*) \). For any threshold \( t \), Lasso’s \( |w| \leq t \) is \( |w_1| + |w_2| \leq t \), a diamond. Ridge’s \( ||w|| \leq t \) is \( w_1^2 + w_2^2 \leq t \), a disk. The diamond is likely to be intersected by an elliptical contour of \( \text{MSE} \) at a corner where \( w_1 = 0 \) or \( w_2 = 0 \).
• **Ridge** regression uses L2 regularization, finding 
\[ \min_{w,b} \left( \frac{1}{N} \sum_{i=1}^{N} [f_{w,b}(x_i) - y_i]^2 + \alpha ||w||^2 \right), \]
where \( \alpha \geq 0 \).
Ridge usually outperforms Lasso on unseen examples. Its objective function is differentiable, so gradient descent can be used.
Here is a nice figure showing the effect of \( \alpha \) on \( w \):


**Python**

- from sklearn import linear_model  # Recall OLS: model = linear_model.LinearRegression()
  model = linear_model.Lasso(alpha=1.0)  # Use only one of
  model = linear_model.Ridge(alpha=1.0)  # these two lines.
- model.fit(X, y), .coef_, .intercept_, .predict(X), .score(X, y) are like OLS in §3

To learn more:

- User guide:
  - Lasso: https://scikit-learn.org/stable/modules/linear_model.html#lasso

- Reference manual:

For context, recall from §3 that we have already done regularization:

• Regarding logistic regression:
  - We modeled \( \hat{P}_{w,b}(y = 1|x) = f_{w,b}(x) = \frac{1}{1 + e^{-(wx+b)}} \).
  - We minimized negative log likelihood, \( -\sum_{i=1}^{N} \ln \left( f_{w,b}(x_i)^{y_i} [1 - f_{w,b}(x_i)]^{1-y_i} \right) \).
  - We added L2 regularization by minimizing \( \frac{1}{2} ||w||^2 + C \sum_{i=1}^{N} \max(0, 1 - y_i (wx_i - b)) \).

• Regarding SVM:
  - Hard-margin SVM minimized \( ||w|| \), subject to \( y_i (wx_i + b) \geq 1 \), which maximized road width \( \frac{2}{||w||} \) subject to the constraints. It required linearly-separable data.
  - Soft-margin SVM introduced hinge loss \( \max(0, 1 - y_i (wx - b)) \) and then minimized \( \frac{1}{2} ||w||^2 + C \sum_{i=1}^{N} \max(0, 1 - y_i (wx_i - b)) \).
  - Maybe we could have started with a classifier that minimized average hinge loss \( \frac{1}{N} \sum_{i=1}^{N} \max(0, 1 - y_i (wx_i - b)) \). Then add the penalty \( ||w||^2 \) to get soft-margin SVM.

*Also see https://xkcd.com/2048*