Cheat Sheet: Building Supervised Learning Models

Common supervised learning models

Process Name	Brief Description	Code Syntax

One vs One classifier (using logistic regression)	**Process:** This method trains one classifier for each pair of classes.
	Key hyperparameters:
	- `estimator`: Base classifier (e.g., logistic regression)
	Pros: Can work well for small datasets.
	Cons: Computationally expensive for large datasets.
	Common applications: Multiclass classification problems where the number of classes is relatively small.
		<pre>from sklearn.multiclass import OneVsOneClassifier
		from sklearn.linear_model import LogisticRegression
		model = OneVsOneClassifier(LogisticRegression())</pre>

One vs All classifier (using logistic regression)	**Process:** Trains one classifier per class, where each classifier distinguishes between one class and the rest.
	Key hyperparameters:
	- `estimator`: Base classifier (e.g., Logistic Regression)
	- `multi_class`: Strategy to handle multiclass classification (`ovr`)
	Pros: Simpler and more scalable than One vs One.
	Cons: Less accurate for highly imbalanced classes.
	Common applications: Common in multiclass classification problems such as image classification.
		<pre>from sklearn.multiclass import OneVsRestClassifier
		from sklearn.linear_model import LogisticRegression
		model = OneVsRestClassifier(LogisticRegression())</pre>
		or
		<pre>from sklearn.linear_model import LogisticRegression
		model_ova = LogisticRegression(multi_class='ovr')</pre>

Decision tree classifier	**Process:** A tree-based classifier that splits data into smaller subsets based on feature values.
	Key hyperparameters:
	- `max_depth`: Maximum depth of the tree
	Pros: Easy to interpret and visualize.
	Cons: Prone to overfitting if not pruned properly.
	Common applications: Classification tasks, such as credit risk assessment.
		<pre>from sklearn.tree import DecisionTreeClassifier
		model = DecisionTreeClassifier(max_depth=5)</pre>

Decision tree regressor	**Process:** Similar to the decision tree classifier, but used for regression tasks to predict continuous values.
	Key hyperparameters:
	- `max_depth`: Maximum depth of the tree
	Pros: Easy to interpret, handles nonlinear data.
	Cons: Can overfit and perform poorly on noisy data.
	Common applications: Regression tasks, such as predicting housing prices.
		<pre>from sklearn.tree import DecisionTreeRegressor
		model = DecisionTreeRegressor(max_depth=5)</pre>

Linear SVM classifier	**Process:** A linear classifier that finds the optimal hyperplane separating classes with a maximum margin.
	Key hyperparameters:
	- `C`: Regularization parameter
	- `kernel`: Type of kernel function (`linear`, `poly`, `rbf`, etc.)
	- `gamma`: Kernel coefficient (only for `rbf`, `poly`, etc.)
	Pros: Effective for high-dimensional spaces.
	Cons: Not ideal for nonlinear problems without kernel tricks.
	Common applications: Text classification and image recognition.
		<pre>from sklearn.svm import SVC
		model = SVC(kernel='linear', C=1.0)</pre>

K-nearest neighbors classifier	**Process:** Classifies data based on the majority class of its nearest neighbors.
	Key hyperparameters:
	- `n_neighbors`: Number of neighbors to use
	- `weights`: Weight function used in prediction (`uniform` or `distance`)
	- `algorithm`: Algorithm used to compute the nearest neighbors (`auto`, `ball_tree`, `kd_tree`, `brute`)
	Pros: Simple and effective for small datasets.
		<pre>from sklearn.neighbors import KNeighborsClassifier
		model = KNeighborsClassifier(n_neighbors=5, weights='uniform')</pre>

Process Name	Brief Description	Code Syntax
	Cons: Computationally expensive as the dataset grows. **Common applications:** Recommendation systems, image recognition.	
Random Forest regressor	**Process:** An ensemble method using multiple decision trees to improve accuracy and reduce overfitting. **Key hyperparameters:** - \`n_estimators\`: Number of trees in the forest - \`max_depth\`: Maximum depth of each tree **Pros:** Less prone to overfitting than individual decision trees. **Cons:** Model complexity increases with the number of trees. **Common applications:** Regression tasks such as predicting sales or stock prices.	``` from sklearn.ensemble import RandomForestRegressor model = RandomForestRegressor(n_estimators=100, max_depth=5) ```
XGBoost regressor	**Process:** A gradient boosting method that builds trees sequentially to correct errors from previous trees. **Key hyperparameters:** - \`n_estimators\`: Number of boosting rounds - \`learning_rate\`: Step size to improve accuracy - \`max_depth\`: Maximum depth of each tree **Pros:** High accuracy and works well with large datasets. **Cons:** Computationally intensive, complex to tune. **Common applications:** Predictive modeling, especially in Kaggle competitions.	``` import xgboost as xgb model = xgb.XGBRegressor(n_estimators=100, learning_rate=0.1, max_depth=5) ```

Associated functions used

Method Name	Brief Description	Code Syntax
OneHotEncoder	Transforms categorical features into a one-hot encoded matrix.	``` from sklearn.preprocessing import OneHotEncoder encoder = OneHotEncoder(sparse=False) encoded_data = encoder.fit_transform(categorical_data) ```
accuracy_score	Computes the accuracy of a classifier by comparing predicted and true labels.	``` from sklearn.metrics import accuracy_score accuracy = accuracy_score(y_true, y_pred) ```
LabelEncoder	Encodes labels (target variable) into numeric format.	``` from sklearn.preprocessing import LabelEncoder encoder = LabelEncoder() encoded_labels = encoder.fit_transform(labels) ```
plot_tree	Plots a decision tree model for visualization.	``` from sklearn.tree import plot_tree plot_tree(model, max_depth=3, filled=True) ```
normalize	Scales each feature to have zero mean and unit variance (standardization).	``` from sklearn.preprocessing import normalize normalized_data = normalize(data, norm='l2') ```
compute_sample_weight	Computes sample weights for imbalanced datasets.	``` from sklearn.utils.class_weight import compute_sample_weight weights = compute_sample_weight(class_weight='balanced', y=y) ```
roc_auc_score	Computes the Area Under the Receiver Operating Characteristic Curve (AUC-ROC) for binary classification models.	``` from sklearn.metrics import roc_auc_score auc = roc_auc_score(y_true, y_score) ```

Author

Jeff Grossman
Abhishek Gagneja