Folha de Dicas: Construindo Modelos de Aprendizado Supervisionado

Modelos comuns de aprendizado supervisionado

Nome do Processo	Descrição Breve	Sintaxe do Código

Classificador One vs One (usando regressão logística)	**Processo:** Este método treina um classificador para cada par de classes.
	Hiperparâmetros chave:
	- `estimator`: Classificador base (por exemplo, regressão logística)
	Prós: Pode funcionar bem para conjuntos de dados pequenos.
	Contras: Computacionalmente caro para conjuntos de dados grandes.
	Aplicações comuns: Problemas de classificação multiclasse onde o número de classes é relativamente pequeno.
		`from sklearn.multiclass import OneVsOneClassifier`
		`from sklearn.linear_model import LogisticRegression`
		`model = OneVsOneClassifier(LogisticRegression())`

Classificador One vs All (usando regressão logística)	**Processo:** Treina um classificador por classe, onde cada classificador distingue entre uma classe e o restante.
	Hiperparâmetros chave:
	- `estimator`: Classificador base (por exemplo, Regressão Logística)
	- `multi_class`: Estratégia para lidar com classificação multiclasse (`ovr`)
	Prós: Mais simples e escalável do que One vs One.
	Contras: Menos preciso para classes altamente desbalanceadas.
	Aplicações comuns: Comum em problemas de classificação multiclasse, como classificação de imagens.
		`from sklearn.multiclass import OneVsRestClassifier`
		`from sklearn.linear_model import LogisticRegression`
		`model = OneVsRestClassifier(LogisticRegression())`
		or
		`from sklearn.linear_model import LogisticRegression`
		`model_ova = LogisticRegression(multi_class='ovr')`

Classificador de árvore de decisão	**Processo:** Um classificador baseado em árvore que divide os dados em subconjuntos menores com base nos valores das características. **Hiperparâmetros chave:** - `max_depth`: Profundidade máxima da árvore **Prós:** Fácil de interpretar e visualizar. **Contras:** Suscetível ao overfitting se não for podada corretamente. **Aplicações comuns:** Tarefas de classificação, como avaliação de risco de crédito.	`from sklearn.tree import DecisionTreeClassifier` `model = DecisionTreeClassifier(max_depth=5)`
Regressor de árvore de decisão	**Processo:** Semelhante ao classificador de árvore de decisão, mas usado para tarefas de regressão para prever valores contínuos. **Hiperparâmetros chave:** - `max_depth`: Profundidade máxima da árvore **Prós:** Fácil de interpretar, lida com dados não lineares. **Contras:** Pode overfit e ter um desempenho ruim em dados ruidosos. **Aplicações comuns:** Tarefas de regressão, como prever preços de imóveis.	`from sklearn.tree import DecisionTreeRegressor` `model = DecisionTreeRegressor(max_depth=5)`
Classificador SVM linear	**Processo:** Um classificador linear que encontra o hiperplano ótimo separando as classes com uma margem máxima. **Hiperparâmetros chave:** - `C`: Parâmetro de regularização - `kernel`: Tipo de função de kernel (`linear`, `poly`, `rbf`, etc.) - `gamma`: Coeficiente do kernel (apenas para `rbf`, `poly`, etc.) **Prós:** Eficaz para espaços de alta dimensão. **Contras:** Não é ideal para problemas não lineares sem truques de kernel. **Aplicações comuns:** Classificação de texto e reconhecimento de imagens.	`from sklearn.svm import SVC` `model = SVC(kernel='linear', C=1.0)`
Classificador de k-vizinhos mais próximos	**Processo:** Classifica os dados com base na classe majoritária de seus vizinhos mais próximos. **Hiperparâmetros chave:**	`from sklearn.neighbors import KNeighborsClassifier` `model = KNeighborsClassifier(n_neighbors=5, weights='uniform')`

Nome do Processo	Descrição Breve	Sintaxe do Código
	- `n_neighbors`: Número de vizinhos a serem utilizados - `weights`: Função de peso usada na previsão (`uniform` ou `distance`) - `algorithm`: Algoritmo usado para calcular os vizinhos mais próximos (`auto`, `ball_tree`, `kd_tree`, `brute`) **Prós:** Simples e eficaz para conjuntos de dados pequenos. **Contras:** Computacionalmente caro à medida que o conjunto de dados cresce. **Aplicações comuns:** Sistemas de recomendação, reconhecimento de imagens.	
Regressor de Floresta Aleatória	**Processo:** Um método de ensemble que utiliza múltiplas árvores de decisão para melhorar a precisão e reduzir o overfitting. **Hiperparâmetros chave:** - `n_estimators`: Número de árvores na floresta - `max_depth`: Profundidade máxima de cada árvore **Prós:** Menos suscetível ao overfitting do que árvores de decisão individuais. **Contras:** A complexidade do modelo aumenta com o número de árvores. **Aplicações comuns:** Tarefas de regressão, como prever vendas ou preços de ações.	`from sklearn.ensemble import RandomForestRegressor` `model = RandomForestRegressor(n_estimators=100, max_depth=5)`
Regressor XGBoost	**Processo:** Um método de boosting de gradiente que constrói árvores sequencialmente para corrigir erros das árvores anteriores. **Hiperparâmetros chave:** - `n_estimators`: Número de rodadas de boosting - `learning_rate`: Tamanho do passo para melhorar a precisão - `max_depth`: Profundidade máxima de cada árvore **Prós:** Alta precisão e funciona bem com grandes conjuntos de dados. **Contras:** Intensivo em computação, complexo de ajustar. **Aplicações comuns:** Modelagem preditiva, especialmente em competições Kaggle.	`import xgboost as xgb` `model = xgb.XGBRegressor(n_estimators=100, learning_rate=0.1, max_depth=5)`

Funções associadas usadas

Nome do Método	Descrição Breve	Sintaxe do Código

OneHotEncoder	Transforma características categóricas em uma matriz codificada em one-hot.	`from sklearn.preprocessing import OneHotEncoder`
		`encoder = OneHotEncoder(sparse=False)`
		`encoded_data = encoder.fit_transform(categorical_data)`

accuracy_score	Calcula a acurácia de um classificador comparando rótulos previstos e verdadeiros.	`from sklearn.metrics import accuracy_score` `accuracy = accuracy_score(y_true, y_pred)`

LabelEncoder	Codifica rótulos (variável alvo) em formato numérico.	`from sklearn.preprocessing import LabelEncoder`
		`encoder = LabelEncoder()`
		`encoded_labels = encoder.fit_transform(labels)`

plot_tree	Plota um modelo de árvore de decisão para visualização.	`from sklearn.tree import plot_tree` `plot_tree(model, max_depth=3, filled=True)`
normalize	Escala cada característica para ter média zero e variância unitária (padronização).	`from sklearn.preprocessing import normalize` `normalized_data = normalize(data, norm='l2')`
compute_sample_weight	Calcula pesos de amostra para conjuntos de dados desbalanceados.	`from sklearn.utils.class_weight import compute_sample_weight` `weights = compute_sample_weight(class_weight='balanced', y=y)`
roc_auc_score	Calcula a Área Sob a Curva Característica de Operação do Receptor (AUC-ROC) para modelos de classificação binária.	`from sklearn.metrics import roc_auc_score` `auc = roc_auc_score(y_true, y_score)`

Autor

Jeff Grossman
Abhishek Gagneja