Machine Learning Models, Optimisation and Evaluation

Abstract

This document provides a summary of machine learning techniques, including an overview of different model types, optimisation methodologies and evaluation metrics.

Table of Contents

1. Introduction

Machine learning (ML) is a subfield of artificial intelligence focused on building systems that learn patterns from data and improve performance on a task without being explicitly programmed. This document provides a structured overview of the main types of machine learning models, model optimisation techniques, and metrics used to assess model performance.

2. Exploratory data analysis

3. Types of Machine Learning Models

3.1. Supervised Learning

Supervised learning models are trained on labelled data, where both inputs and target outputs are known.

3.1.1. Regression Models

Regression models are used when the target variable is continuous. They aim to model the relationship between input features and a numerical output.

- Linear Regression :

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create a linear regression model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)

# Predict using the model
y_pred = model.predict(X_test)
```

- Polynomial Regression

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

# Generate polynomial features
poly = PolynomialFeatures(degree=2)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)

# Create a linear regression model
model = LinearRegression()

# Train the model
model.fit(X_train_poly, y_train)

# Predict using the model
y_pred = model.predict(X_test_poly)
```

```
# Calculate MSE and R-squared
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

- Ridge Regression

```
# Standardize features (optional but recommended for polynomial features)
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Apply second-order polynomial transformation
poly = PolynomialFeatures(degree=2, include_bias=False)
X_train_poly = poly.fit_transform(X_train_scaled)
X_test_poly = poly.transform(X_test_scaled)

# Create Ridge regression model with regularization alpha=0.1
ridge_model = Ridge(alpha=0.1)

# Fit model on polynomial-transformed training data
ridge_model.fit(X_train_poly, y_train)

# Predict on polynomial-transformed test data
y_pred = ridge_model.predict(X_test_poly)
```

- Lasso Regression
- Support Vector Regression (SVR)
- Decision Tree Regressors

```
from sklearn.tree import DecisionTreeRegressor
model = DecisionTreeRegressor(max_depth=5)
```

- Random Forest Regressors

```
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor(n_estimators=100, max_depth=5)
```

- XGBoost regressor

```
import xgboost as xgb
model = xgb.XGBRegressor(n_estimators=100, learning_rate=0.1, max_depth=5)
```

- Gradient Boosting Regressors

3.1.2. Classification Models

Classification models are used when the target variable is categorical. They assign input data to one of several predefined classes.

- Logistic Regression

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X, y)
```

- Linear SVM classifier : 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.

```
from sklearn.svm import SVC
model = SVC(kernel='linear', C=1.0)
```

- k-Nearest Neighbours (k-NN)

```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=5, weights='uniform')
```

- Naive Bayes
- Support Vector Machines (SVM)
- Decision Trees

```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier(max_depth=5)
```

- Random Forests
- Gradient Boosting (XGBoost, LightGBM, CatBoost)
- Neural Networks

3.2. Unsupervised Learning

Unsupervised learning models operate on unlabelled data to discover hidden structure or patterns.

3.2.1. Clustering

Clustering algorithms group similar data points together based on a distance or similarity measure.
- k-Means
- Hierarchical Clustering
- DBSCAN
- HDBSCAN
- Gaussian Mixture Models (GMM)

3.2.2. Dimensionality Reduction

Dimensionality reduction techniques reduce the number of input features while preserving as much information as possible.
- Principal Component Analysis (PCA)
- t-SNE
- UMAP (Uniform Manifold Approximation and Projection) is used for dimensionality reduction.
  Pros: High performance, preserves global structure.

Cons: Sensitive to parameters.
Applications: Data visualization, feature extraction.
Key hyperparameters:
- n_neighbors: Controls the local neighborhood size (default = 15).
- min_dist: Controls the minimum distance between points in the embedded space (default = 0.1).
- n_components: The dimensionality of the embedding (default = 2).

3.3. Semi-Supervised Learning

Semi-supervised learning combines a small amount of labelled data with a large amount of unlabelled data to improve learning performance.
- Label Propagation
- Self-training methods

3.4. Reinforcement Learning

Reinforcement learning models learn optimal behaviour through interaction with an environment using reward signals.

3.5. Ensemble Methods

Ensemble methods combine multiple models to improve predictive performance and robustness.
- Bagging (e.g. Random Forests)
- Boosting (AdaBoost, Gradient Boosting)
- Stacking

4. Model Optimisation Techniques

4.1. Feature Engineering

Feature engineering focuses on improving input representations to enhance model performance.
- Feature scaling
  - standardisation

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

  - normalisation
- Encoding categorical variables
- Feature selection
- Feature extraction

4.2. Optimisation Algorithms

Optimisation algorithms minimise a loss function during training.
- Gradient Descent
- Stochastic Gradient Descent (SGD)
- Mini-batch Gradient Descent
- Momentum
- RMSProp
- Adam

4.3. Hyperparameter Optimisation

Hyperparameter optimisation searches for the best model configuration.

- Grid Search

```
# 1. Split into train+validation (80%) and test (20%)
X_temp, X_test, y_temp, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 2. Split train+validation into training (60%) and validation (20%)
X_train, X_val, y_train, y_val = train_test_split(
    X_temp, y_temp, test_size=0.25, random_state=42
)
# Explanation: 0.25 * 0.8 = 0.2 → validation is 20% of total

# 3. Create a pipeline: scaling -> polynomial -> Ridge
pipeline = Pipeline([
    ("scaler", StandardScaler()),
    ("poly", PolynomialFeatures(include_bias=False)),
    ("ridge", Ridge())
])

# 4. Define parameter grid: Ridge alpha and polynomial degree
param_grid = {
    "poly__degree": [1, 2, 3],
    "ridge__alpha": [0.01, 0.1, 1, 10, 100]
}

# 5. Grid Search with 4-fold CV
grid_search = GridSearchCV(
    estimator=pipeline,
    param_grid=param_grid,
    cv=4,
    scoring="r2",  # you can use 'neg_mean_squared_error' instead
    n_jobs=-1
)

# Fit Grid Search on training data
grid_search.fit(X_train, y_train)

# 6. Best parameters based on CV
best_params = grid_search.best_params_
print("Best parameters from Grid Search:", best_params)

# 7. Evaluate best model on validation set
best_model = grid_search.best_estimator_
y_val_pred = best_model.predict(X_val)

r2_val = r2_score(y_val, y_val_pred)
mse_val = mean_squared_error(y_val, y_val_pred)

# 8. Evaluate best model on test set
y_test_pred = best_model.predict(X_test)

r2_test = r2_score(y_test, y_test_pred)
mse_test = mean_squared_error(y_val, y_test_pred)
```

- Random Search
- Bayesian Optimisation

- Hyperband

4.4. Regularisation Techniques

Regularisation techniques reduce overfitting by constraining model complexity.
- L1 regularisation (Lasso)
- L2 regularisation (Ridge)
- Elastic Net
- Dropout (neural networks)
- Early stopping

4.5. Cross-Validation

Cross-validation estimates a model's ability to generalise to unseen data.
- k-Fold Cross-Validation
- Stratified k-Fold

5. Model Evaluation Metrics

5.1. Regression Metrics

Regression metrics evaluate performance on continuous targets:

- Mean Absolute Error :

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

```
from sklearn.metrics import mean_absolute_error
mae = mean_absolute_error(y_test, y_pred)
```

- Mean Squared Error :

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, y_pred)
print(f"Test MSE: {mse:.2e}")
```

- Root Mean Squared Error :

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

```
import numpy as np
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
```

- R^2 score : useful to assess the proportion of variance explained by the model, to be used alongside above-mentioned error-based metrics as it does not measure prediction error magnitude.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2}$$

```python
from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred)
print(f"R² score (test data): {r2:.4f}")
```

5.2. Classification Metrics

Classification metrics assess predictive performance on categorical targets.

- Accuracy : proportion of correct predictions (true positives and true negatives) out of all predictions.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

- Precision :

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- Recall (true positive rate) :

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- F1-score : The F1-score is the harmonic mean of precision and recall, providing a single metric that balances both measures. It is particularly useful when you need to find an optimal blend of both. If either precision or recall is 0, then $F_1 = 0$.

$$F_1 = 2 \cdot \left(\frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \right)$$

- Confusion matrix

5.2.1. Threshold-Independent Metrics

- Receiver Operating Characteristic (ROC) curve and AUC
- Precision–Recall curve

5.3. Clustering Metrics

Clustering metrics evaluate the quality of unsupervised groupings.
- Silhouette score
- Davies–Bouldin index

5.4. Probabilistic and Ranking Metrics

These metrics are used when models output probabilities or rankings.
- Log loss (cross-entropy) :

$$L = -\sum_{i=1}^{C}(y_i \log(\hat{y}_i))$$

```python
from sklearn.metrics import log_loss
loss = log_loss(y_true, y_pred_proba)
```

- Brier score

- Mean Average Precision (MAP)
- Normalised Discounted Cumulative Gain (NDCG)

6. Bias–Variance Trade-off

As model complexity increases, bias generally decreases while variance increases. A well-performing model balances both to minimise generalisation error.

7. Vizualisation Techniques

7.1. Matplotlib

- Line plot

```
import matplotlib.pyplot as plt
plt.plot(x, y, color='red', linewidth=2)
plt.title("My Title")
plt.xlabel("Feature_name")
plt.ylabel("Target_name")
plt.ylim(min,max)
plt.grid(True)
plt.show()
```

- Scatter plot

```
plt.scatter(x, y, color='purple', marker='o', s=50)
```

- Hist plot

```
plt.hist(df["Feature"], bins=25, color="orange", edgecolor="black")
```

- Bar chart

```
plt.bar(x, height, color='green', width=0.5)
```

- Pie chart

```
plt.pie(sizes, labels=labels, colors=colors, explode=explode)
```

- Subplotting

```
fig, axes = plt.subplots(nrows=2, ncols=2)
```

7.2. Seaborne

- Scatter plot

```
sns.scatterplot(x="Feature1", y="Feature2", hue="Target", data=df)
```

- Regplot

```
sns.regplot(x="Feature", y="Target", data=df)
```

- Box plot

```
sns.boxplot(df["Feature"])
```

- Pair plots

```
sns.set_context('talk')
sns.pairplot(df, hue='Target')
```

```
#Plot 5 scatter plots per line, for features, and one hist plot for the target
for i in range(0, len(df.columns), 5):
    sns.pairplot(data=df,
                x_vars=df.columns[i:i+5],
                y_vars=['Target'])
```

- Heatmap
- Distplot

```
tg_plot = sns.distplot(df['Target'])

#If the distribution is skewed, use log transform
log_transformed = np.log(df['Target'])
tg_transformed = sns.distplot(log_transformed)
```

7.3. Pandas

```
#Line plot
df.plot(x=’Date’, y=’Target’, kind=’line’)

#Area plot
df.plot(kind='area')

#Hist plot
df[‘Feature’].plot(kind='hist', bins=10)

#Bar chart
df.plot(kind='bar')

#Pie chart
df.plot(x='Category',y='Percentage',kind='pie')

#Scatter plot
df.plot(x='Height', y='Weight', kind='scatter')
```

- Box plot

```python
df.boxplot(by='target') # Assuming target is categorical
```

8. Conclusion

Machine learning comprises a broad set of models, optimisation strategies, and evaluation metrics. Selecting appropriate techniques depends on the task, data properties, and performance requirements. A structured understanding of these components is essential for developing reliable and effective machine learning systems.