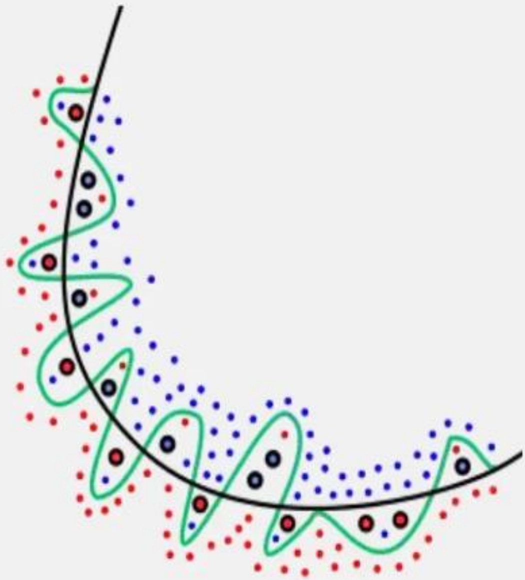
Overfitting in ML
& How to Solve it

With code examples



Swipe
to see in
detail

What is Overfitting

Overfitting occurs when a machine learning model learns the training data too well, capturing noise or random fluctuations in the data that don't represent the underlying patterns. As a result, an overfitted model performs well on the training set but fails to generalize to new, unseen data.

signs that your model might be overfitting:

1. **High Training Accuracy, Low Validation Accuracy:** The model achieves high accuracy on the training data but performs poorly on a separate validation set.
2. **Model Complexity:** Overfit models tend to be excessively complex, capturing noise in the training data instead of the underlying patterns.
3. **Large Differences Between Training and Validation Performance**: If there is a significant gap between the training and validation performance metrics, it may indicate overfitting.

Lert's see how we can solve it

Train-Test Split

Split your dataset into two parts: a training set used to train the model and a test set used to evaluate its performance.

```
from sklearn.model_selection import
train_test_split

X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2,
random_state=42)
```

Cross-Validation

Divide the dataset into multiple subsets (folds) and train the model on different combinations of these subsets to assess its performance more robustly.

```
from sklearn.model_selection import cross_val_score, KFold
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators=100)

# Use cross-validation to evaluate the model
cv = KFold(n_splits=5, shuffle=True, random_state=42)
cv_scores = cross_val_score(model, X, y, cv=cv)

print(f'Cross-Validation Mean Accuracy: {cv_scores.mean()}')
```

Lasso Regression (L1 regularization)

L1 regularization encourages sparsity

```
from sklearn.linear_model import Ridge
ridge_model = Ridge(alpha=1.0)
ridge_model.fit(X_train_scaled, y_train)
```

Ridge Regression (L2 regularization)

L1 regularization encourages sparsity

```
from sklearn.linear_model import Lasso
lasso_model = Lasso(alpha=1.0)
lasso_model.fit(X_train_scaled, y_train)
```

Dropout for Neural Networks

Randomly deactivate some neurons during training to prevent overreliance on specific ones and enhance generalization.

```
from tensorflow.keras.models import
Sequential
from tensorflow.keras.layers import
Dense, Dropout

model = Sequential([
    Dense(64, activation='relu',
input_dim=X_train.shape[1]),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam',
loss='binary_crossentropy', metrics=
['accuracy'])
model.fit(X_train, y_train, epochs=100,
validation_data=(X_val, y_val))
```

Early Stopping

Monitor the validation performance during training and stop when it starts degrading.

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import EarlyStopping

model = Sequential([
    Dense(64, activation='relu', input_dim=X_train.shape[1]),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

# Use early stopping to monitor validation loss
early_stopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)

# Fit the model with early stopping
model.fit(X_train, y_train, epochs=100, validation_data=(X_val,
y_val), callbacks=[early_stopping])
```

Ensemble Methods

Combine predictions from multiple models to improve overall performance.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier

model1 = RandomForestClassifier(n_estimators=100)
model2 = SomeOtherClassifier()

ensemble_model = VotingClassifier(estimators=[('rf', model1),
('other', model2)], voting='hard')
ensemble_model.fit(X_train, y_train)
```

Feature Selection

Select a subset of the most important features to reduce complexity and enhance generalization.

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators=100)
model.fit(X_train, y_train)

feature_importances = model.feature_importances_
selected_features = X_train.columns[feature_importances > threshold]
```

Data Augmentation

Generate new training samples by applying transformations to existing ones, especially useful for image data.

```
from
tensorflow.keras.preprocessing.image
import ImageDataGenerator

datagen =
ImageDataGenerator(rotation_range=20,
width_shift_range=0.2,
height_shift_range=0.2,
shear_range=0.2, zoom_range=0.2,
horizontal_flip=True)

# Fit the generator on your training
data
datagen.fit(X_train)

# Use the generator for training
model.fit(datagen.flow(X_train,
y_train, batch_size=32), epochs=100,
validation_data=(X_val, y_val))
```