```
import os
import logging
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
import joblib

# Configure logging
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(name)s - %(levelname)s -
%(message)s')
logger = logging.getLogger(__name__)

class DocumentClassifierTrainer:
    def __init__(self, data_dir="training_data"):
        self.data_dir = data_dir
        self.model_path = os.path.join(data_dir, "document_classifier.pkl")
        self.vectorizer_path = os.path.join(data_dir, "tfidf_vectorizer.pkl")

        # Create data dir if it doesn't exist
        os.makedirs(data_dir, exist_ok=True)

    def load_training_data(self):
        """
        Load training data from files

        In a real implementation, this would load text samples from
        a structured dataset. For this example, we'll create synthetic examples.
        """
        logger.info("Loading training data...")

        # This is a placeholder for real training data
        # In a real implementation, you'd load actual document text and labels

        texts = []
        labels = []

        # Examples for Final Inspection Card - Using real examples from provided document
s
        fic_examples = [
            "City of Selma BUILDING DIVISION Location of Work: 2252 BERRY ST, SELMA, CA 9
3662 SOLAR INSPECTIONS SOLAR- PANEL HOLD DOWNS SOLAR- UFER GROUND SOLAR - FINAL",
            "CITY OF CREST HILL, ILLINOIS BUILDING DEPARTMENT APPROVED Type of Inspection
 SOLAR FINAL Date of Inspection 2-28-25",
            "NOTICE THIS IS A SERVICE ONLY CERTIFICATE This is to certify that I have thi
s day inspected and approved service CITY ELECTRICAL INSPECTOR CITY OF GALESBURG STATE OF
 ILLINOIS",
            "MCCARTHY SOLAR PV ROOF MOUNTED SOLAR PV SYSTEM Building Final Pass 02/11/202
5 Electrical Final Pass 02/11/2025",
            "CITY OF RIVERSIDE BUILDING & SAFETY DIVISION PERMIT NUMBER BP-2024-18468 INS
TALL PV SOLAR SYSTEM PHOTOVOLTAIC PANELS ON RESIDENTIAL ROOF Date of Final 2/28/25",
            "FINAL INSPECTION CARD Building Department Approval for solar installation PA
SSED POST THIS CARD SO IT IS VISIBLE FROM STREET",
            "City of Yuba City Development Services Department BUILDING PERMIT Type: Resi
dential Permit Sub-Type: SOLAR (SOLAR/R) Category: WITHOUT STORAGE BATTERY",
            "I, Shelby Guenther, attest that this property is located within a jurisdicti
on in which permits are not required for the installation of solar panels.",
            "No permits no inspections moving to interconnection"
        ]
        for text in fic_examples:
            texts.append(text)
            labels.append(1)  # Category 1: Final Inspection Card

        # Examples for Interconnection Agreement - Based on real examples
        ia_examples = [
            "AGREEMENT FOR INTERCONNECTION AND PARALLEL OPERATION OF DISTRIBUTED GENERATI
ON Oncor Electric Delivery Company LLC",
            "Interconnection Agreement is made and entered into this day by Oncor Electri
c Delivery Company LLC and Customer",
```

```
            "Certificate of Completion Interconnection Customer Information Name Filomena
 Hernandez Street Address 682 Elsie Ave",
            "SOUTHERN CALIFORNIA EDISON COMPANY NET BILLING TARIFF OR NET ENERGY METERING
 AND RENEWABLE ELECTRICAL GENERATING FACILITY INTERCONNECTION AGREEMENT",
            "This Net Billing Tariff (NBT) or Net Energy Metering (NEM) and Renewable Ele
ctrical Generating Facility Interconnection Agreement",
            "AGREEMENT AND CUSTOMER AUTHORIZATION Net Billing Tariff (NBT) Interconnectio
n for Solar and/or Wind Electric Generating Facilities",
            "Application for Net Metering Services Ameren Illinois",
            "Acceptance and Final Approval for Interconnection The interconnection agreem
ent is approved",
            "Interconnection Agreement for Customer-Owned Renewable Generation Tier 1 - 1
0 kW or Less",
            "The interconnection customer acknowledges that it shall not operate the dist
ributed generation facility until receipt of the final acceptance"
        ]
        for text in ia_examples:
            texts.append(text)
            labels.append(2)  # Category 2: Interconnection Agreement

        # Examples for PTO (Permission-To-Operate)
        pto_examples = [
            "PERMISSION TO OPERATE Notice Date: 06/01/2023 Customer Address: 123 Main St"
,
            "Electric Utility Company PERMISSION TO OPERATE NOTIFICATION Solar Generation
 System",
            "PTO APPROVAL Your solar system at 456 Oak Avenue has been authorized to oper
ate",
            "Permission to Operate (PTO) Date Received: 07/15/2023 Solar Generator Addres
s: 789 Pine Rd",
            "NOTICE OF AUTHORIZATION TO OPERATE RENEWABLE GENERATING FACILITY PTO Date: 0
8/20/2023"
        ]
        for text in pto_examples:
            texts.append(text)
            labels.append(3)  # Category 3: PTO

        # Examples for Warranty Extension
        we_examples = [
            "Solar Panel Warranty Extension Certificate SolarEdge Inverter Serial Number:
 SE12345678",
            "EXTENDED WARRANTY DOCUMENT SolarEdge Technologies Inc. Product Warranty Exte
nsion",
            "SolarEdge Warranty Extension Confirmation Serial Number: SE98765432",
            "Product Warranty Extension Agreement SolarEdge Monitoring Portal Verificatio
n",
            "Solar System Extended Warranty Certificate for SolarEdge Components Serial #
: SE11223344"
        ]
        for text in we_examples:
            texts.append(text)
            labels.append(4)  # Category 4: Warranty Extension

        # Examples for Interconnection / NEM Agreement
        nem_examples = [
            "NET ENERGY METERING (NEM) INTERCONNECTION AGREEMENT Customer Signature: John
 Doe Utility Signature: Jane Smith",
            "SOLAR NEM AGREEMENT Application for Net Energy Metering Solar Facility",
            "Net Energy Metering (NEM) and Interconnection Agreement for Solar Generating
 Facility",
            "UTILITY COMPANY NEM INTERCONNECTION AGREEMENT Renewable Energy Credits (SREC
)",
            "Standard Net Energy Metering Agreement for Customer-Generator Facility"
        ]
        for text in nem_examples:
            texts.append(text)
            labels.append(5)  # Category 5: Interconnection / NEM Agreement

        logger.info(f"Loaded {len(texts)} training examples with {len(set(labels))} categ
ories")
```

```
        return texts, labels

    def train_model(self):
        """
        Train a document classifier model
        """
        try:
            # Load training data
            texts, labels = self.load_training_data()

            # Data augmentation - generate more training samples for each category
            # by creating slight variations of existing examples
            augmented_texts = []
            augmented_labels = []

            # Keep track of samples per class for better balancing
            sample_counts = {}
            for label in labels:
                sample_counts[label] = sample_counts.get(label, 0) + 1

            # Add the original samples
            augmented_texts.extend(texts)
            augmented_labels.extend(labels)

            # For each original text, create variations by removing random words,
            # changing capitalization, etc.
            import random
            for i, (text, label) in enumerate(zip(texts, labels)):
                # Only augment if this class has fewer than average samples
                avg_samples = len(texts) / len(set(labels))
                if sample_counts[label] < avg_samples * 1.2:
                    # Create 1-2 variations for underrepresented classes
                    num_variations = min(2, int(avg_samples - sample_counts[label]) + 1)

                    words = text.split()
                    for _ in range(num_variations):
                        if len(words) > 5:  # Only modify if there are enough words
                            # Remove 1-2 random words (except first and last 2 words)
                            variant_words = words.copy()
                            for _ in range(min(2, len(words) - 4)):
                                idx_to_remove = random.randint(2, len(variant_words) - 3)
                                variant_words.pop(idx_to_remove)

                            # Change capitalization of 1-2 words
                            for _ in range(min(2, len(variant_words))):
                                idx_to_modify = random.randint(0, len(variant_words) - 1)
                                if random.choice([True, False]):
                                    variant_words[idx_to_modify] = variant_words[idx_to_m
odify].upper()
                                else:
                                    variant_words[idx_to_modify] = variant_words[idx_to_m
odify].lower()

                            augmented_text = ' '.join(variant_words)
                            augmented_texts.append(augmented_text)
                            augmented_labels.append(label)
                            sample_counts[label] = sample_counts[label] + 1

            logger.info(f"Original samples: {len(texts)}, Augmented samples: {len(augment
ed_texts)}")

            # Split into train and test sets
            X_train, X_test, y_train, y_test = train_test_split(
                augmented_texts, augmented_labels, test_size=0.2, random_state=42
            )

            # Create and fit the vectorizer with enhanced parameters
            logger.info("Creating TF-IDF features...")
            vectorizer = TfidfVectorizer(
```

```
                max_features=5000,
                ngram_range=(1, 3),
                min_df=2,
                use_idf=True,
                sublinear_tf=True
            )
            X_train_tfidf = vectorizer.fit_transform(X_train)
            X_test_tfidf = vectorizer.transform(X_test)

            # Train the classifier with better parameters for small datasets
            logger.info("Training document classifier...")
            # Use higher C value to reduce regularization for small datasets
            classifier = LogisticRegression(
                C=5.0,
                random_state=42,
                max_iter=2000,
                class_weight='balanced',
                solver='liblinear',
                multi_class='ovr'
            )
            classifier.fit(X_train_tfidf, y_train)

            # Evaluate the model
            y_pred = classifier.predict(X_test_tfidf)
            accuracy = accuracy_score(y_test, y_pred)
            logger.info(f"Model accuracy: {accuracy:.4f}")
            logger.info("Classification report:\n" + classification_report(y_test, y_pred
))

            # Save the model and vectorizer
            logger.info(f"Saving model to {self.model_path}")
            joblib.dump(classifier, self.model_path)

            logger.info(f"Saving vectorizer to {self.vectorizer_path}")
            joblib.dump(vectorizer, self.vectorizer_path)

            logger.info("Training completed successfully")
            return True

        except Exception as e:
            logger.error(f"Error training model: {str(e)}", exc_info=True)
            return False

    def test_document_classification(self, test_text):
        """
        Test the trained model on a sample document
        """
        try:
            # Load the model and vectorizer
            classifier = joblib.load(self.model_path)
            vectorizer = joblib.load(self.vectorizer_path)

            # Transform the test text
            test_tfidf = vectorizer.transform([test_text])

            # Predict the category
            category_id = classifier.predict(test_tfidf)[0]
            probabilities = classifier.predict_proba(test_tfidf)[0]
            confidence = np.max(probabilities)

            # Map category ID to name
            category_names = {
                1: "Final Inspection Card",
                2: "Interconnection Agreement",
                3: "PTO",
                4: "Warranty Extension",
                5: "Interconnection / NEM Agreement"
            }

            category_name = category_names.get(category_id, "Unknown")
```

```
            return {
                "category": category_name,
                "confidence": confidence,
                "category_id": int(category_id)
            }

        except Exception as e:
            logger.error(f"Error testing document classification: {str(e)}", exc_info=Tru
e)
            return {
                "category": "Error",
                "confidence": 0.0,
                "error": str(e)
            }

if __name__ == "__main__":
    trainer = DocumentClassifierTrainer()
    trainer.train_model()

    # Test with a sample document
    test_sample = "PERMISSION TO OPERATE Notice Date: 10/01/2023 Your solar system at 123
 Example St has been approved"
    result = trainer.test_document_classification(test_sample)
    print(f"Test Sample Classification: {result}")
```