EXP NO. 10 IMPLEMENTATION OF BLOCK WORLD PROGRAM

AIM:

ALGORITHM:

PROGRAM:

```
world = ['A', 'B', 'C']
print("Initial state:")
print(world)
while True:
    command = input("Enter a command ('help' for a list of commands): ")
    if command == 'help':
        print("Commands: move [block] [destination], add [block], remove [block], exit")
    elif command.startswith('move'):
        _, block, dest = command.split()
        world.remove(block)
        world.insert(world.index(dest), block)
        print("New state:")
        print(world)
    elif command.startswith('add'):
        _, block = command.split()
        world.append(block)
        print("New state:")
        print(world)
    elif command.startswith('remove'):
        _, block = command.split()
        world.remove(block)
        print("New state:")
        print(world)
    elif command == 'exit':
        print("Goodbye!")
        break
    else:
        print("Invalid command. Type 'help' for a list of commands.")
```

EXP NO. 11 IMPLEMENTATION OF LEARNING ALGORITHM FOR APPLICATION

AIM:

ALGORITHM:

PROGRAM:

```
import numpy as np
import matplotlib.pyplot as plt
data = np.loadtxt('dataset.txt', delimiter=',')
X_train, y_train = data[:80, 0], data[:80, 1]
X_test, y_test = data[80:, 0], data[80:, 1]
def hypothesis(theta0, theta1, x):
    return theta0 + theta1 * x
def cost_function(theta0, theta1, X, y):
    m = len(y)
    return np.sum((hypothesis(theta0, theta1, X) - y)**2) / (2 * m)
def gradient_descent(theta0, theta1, X, y, alpha, num_iterations):
    m = len(y)
    for i in range(num_iterations):
        temp0 = theta0 - alpha * np.sum(hypothesis(theta0, theta1, X) - y) / m
        temp1 = theta1 - alpha * np.sum((hypothesis(theta0, theta1, X) - y) * X) / m
        theta0 = temp0
        theta1 = temp1
    return theta0, theta1
theta0, theta1 = gradient_descent(0, 0, X_train, y_train, 0.01, 1000)
y_pred = hypothesis(theta0, theta1, X_test)
print("Model parameters: theta0 = {}, theta1 = {}".format(theta0, theta1))
plt.scatter(X_test, y_test)
plt.plot(X_test, y_pred)
plt.show()
```

EXP NO. 12 DEVELOPMENT OF ENSEMBLE MODEL FOR APPLICATION

AIM:

ALGORITHM:

PROGRAM:

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.metrics import accuracy_score

data = np.loadtxt('dataset.txt', delimiter=',')
X_train, X_test, y_train, y_test = train_test_split(data[:, :-1], data[:, -1], test_size=0.2, random_state=42)
model1 = DecisionTreeClassifier(max_depth=3)
model2 = KNeighborsClassifier(n_neighbors=3)
model1.fit(X_train, y_train)
model2.fit(X_train, y_train)
ensemble = VotingClassifier(estimators=[('dt', model1), ('knn', model2)], voting='hard')
ensemble.fit(X_train, y_train)
y_pred = ensemble.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy of the ensemble model:", accuracy)
```

EXP NO. 13 NATURAL LANGUAGE PROCESSING – LEVELS OF NLP

AIM:

ALGORITHM:

PROGRAM:

```
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.tag import pos_tag, map_tag
from nltk.chunk import ne_chunk
from nltk.sentiment import SentimentIntensityAnalyzer
from nltk.parse import DependencyGraph, DependencyEvaluator
from nltk.translate import Translator
from nltk import download
download('vader_lexicon')
download('maxent_ne_chunker')
download('words')
text = "The quick brown fox jumps over the lazy dog. John Smith works at IBM in New York. I love this movie!"
words = word_tokenize(text)
sentences = sent_tokenize(text)
pos_tags = pos_tag(words)
pos_tags = map_tag('en-ptb', 'universal', pos_tags)
ner_tags = ne_chunk(pos_tag(words))
analyzer = SentimentIntensityAnalyzer()
sentiment = analyzer.polarity_scores(text)
parse = DependencyGraph("(ROOT (S (NP (DT The) (JJ quick) (JJ brown) (NN fox)) (VP (VBZ jumps) (PP (IN over) (NP (D
the) (JJ lazy) (NN dog)))) (. .)))")
dep_graph = DependencyEvaluator(parse)
translator = Translator()
translation = translator.translate(text, src='en', dest='es')
print("Words:", words)
print("Sentences:", sentences)
print("POS Tags:", pos_tags)
print("NER Tags:", ner_tags)
print("Sentiment Analysis:", sentiment)
print("Dependency Parsing:", dep_graph.parse_result)
```

```
print("Translation:", translation.text)
```

EXP NO. 14 IMPLEMENTATION OF NLP PROBLEMS

AIM:

ALGORITHM:

PROGRAM:

```
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.tag import pos_tag, map_tag
from nltk.chunk import ne_chunk
from nltk.sentiment import SentimentIntensityAnalyzer
from nltk.parse import DependencyGraph, DependencyEvaluator
from nltk.translate import Translator
from nltk.corpus import stopwords
nltk.download('vader_lexicon')
nltk.download('maxent_ne_chunker')
nltk.download('words')
nltk.download('stopwords')
text = "The quick brown fox jumps over the lazy dog. John Smith works at IBM in New York. I love this movie!"
words = word_tokenize(text)
sentences = sent_tokenize(text)
pos_tags = pos_tag(words)
pos_tags = map_tag('en-ptb', 'universal', pos_tags)
ner_tags = ne_chunk(pos_tag(words))
analyzer = SentimentIntensityAnalyzer()
sentiment = analyzer.polarity_scores(text)
parse = DependencyGraph("(ROOT (S (NP (DT The) (JJ quick) (JJ brown) (NN fox)) (VP (VBZ jumps) (PP (IN over) (NP (D
the) (JJ lazy) (NN dog)))) (. .)))")
dep_graph = DependencyEvaluator(parse)
translator = Translator()
translation = translator.translate(text, src='en', dest='es')
stop_words = set(stopwords.words('english'))
filtered_words = [word for word in words if word.lower() not in stop_words]
print("Words:", words)
print("Filtered Words:", filtered_words)
print("Sentences:", sentences)
```

```
print("POS Tags:", pos_tags)
print("NER Tags:", ner_tags)
print("Sentiment Analysis:", sentiment)
print("Dependency Parsing:", dep_graph.parse_result)
print("Translation:", translation.text)
```

EXP NO. 15 APPLYING DEEP LEARNING METHODS FOR AN APPLICATION

AIM:

ALGORITHM:

PROGRAM:

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
x_train = x_train.reshape(-1, 28, 28, 1).astype("float32") / 255.0
x_test = x_test.reshape(-1, 28, 28, 1).astype("float32") / 255.0
y_train = keras.utils.to_categorical(y_train)
y_test = keras.utils.to_categorical(y_test)
model = keras.Sequential( keras.Input(shape=(28, 28, 1)),
    layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Flatten(),
    layers.Dropout(0.5),
    layers.Dense(10, activation="softmax"),])
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
history = model.fit(x_train, y_train, batch_size=128, epochs=15, validation_split=0.1)
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)
print("Test accuracy:", test_acc)
predictions = model.predict(np.array([x_test[0]]))
print("Predictions:", predictions)
model.save("mnist_model.h5")
```