AI/ML Classifier API:

The main idea here is to build a classifier that takes in a document or a contract, gathers the content of the contract and classifies it as:

1. Inbound contracts
2. Outbound contracts
3. Rental contracts

Prerequisites:

1. Understanding of the basic Machine Learning concepts
2. Python 3.0
3. Git, Github
4. TCI/Flogo subscription

This can be seen as a **text classification** problem. Text classification is one of the widely used natural language processing (NLP) applications in different business problems. The classifier can be built with a supervised machine learning **classification model** that is able to predict the category of a given contract.This means we need a labeled dataset so the algorithms can learn the patterns and correlations in the data.

Step 1: Reading the data and including the labels

Python comes with a whole lot of libraries to read the files from directory and store it into a dict object and convert it to a data frame.

```
import os
import glob
```

```
files='./data/contract/'
```

```
os.listdir(files)
```

```
['rental', 'inbound', 'outbound']
```

```
from collections import defaultdict
dicts=defaultdict(list)

for dir_name,_,file_names in os.walk(files):
    for file in file_names:
        dicts['categories'].append(os.path.basename(dir_name))
        name=os.path.splitext(file)[0]
        dicts['doc_id'].append(name)
        path=os.path.join(dir_name,file)

        with open(path,'r',encoding='latin-1') as file:
            dicts['text'].append(file.read())
```

```
df=pd.DataFrame.from_dict(dicts)
```

Step 2: Preprocessing into the format that can be used by any ML classifier

Data is cleaned, preprocessed by using Natural Language Processing libraries like nltk. TF-IDF (**term frequency-inverse document frequency**) is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. This is done with the help of the TfidfVectorizer model imported from the scikit-learn library to convert a collection of raw documents to a matrix of TF-IDF features. This trained model is saved for later during prediction.

```
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/sahanaramesh/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
True
```

```
def Clean(text):
    text=text.split()
    text=[i.lower() for i in text if i.lower() not in stopwords.words('english')]
    text=' '.join(text)
    text=re.sub('[^A-Za-z0-9]+',' ',text)
    text=text.lower()
    return text
```

```
df['article']=df['article'].apply(lambda x: Clean(x))
```

```
from sklearn.preprocessing import LabelEncoder
l_enc=LabelEncoder()
```

```
df['labels']=l_enc.fit_transform(df['categories'])
```

```
df_save = df[["categories", "labels"]]
```

```
df_save.to_csv("labels.csv")
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf=TfidfVectorizer()
```

```
X=tfidf.fit_transform(df['article'])
y=df['categories']
```

```
import joblib
filename = 'tfidf_model_contract'
joblib.dump(tfidf, filename)
```

```
['tfidf_model_contract']
```

Step 3: Split the data for training and testing purposes and train different ML models.

The data is usually split in the train test ratio of 1:4 or sometimes even slightly lower for the test part depending on how much data is available.

```
: from sklearn.model_selection import train_test_split
  X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=.3)
```

Step 4: Training and testing the model

There are different supervised classification models that can be used in Machine Learning such as Logistic Regression, Naive Bayes classifier, Decision Tree Classifier, Random Forest Classifier etc. The train data is fed into different ML models and each of

their performance is compared by the various metrics of accuracy, recall, precision and F-1 score.

```
models=[LogisticRegression(),RandomForestClassifier(),DecisionTreeClassifier(),
       MultinomialNB(),SVC(),PassiveAggressiveClassifier()]
scores=pd.DataFrame({'Model':[],'Train_Score':[],'Test_Score':[]})
for j,i in enumerate(models):
    i.fit(X_train,y_train)
    scores.loc[j,:]=[i,i.score(X_train,y_train),i.score(X_test,y_test)]
scores
```

Step 5: Model Selection

The best performing model is selected and saved for prediction using the joblib library. Logistic Regression usually performs very well in a classification problem.

Logistic Regression

```
clf = LogisticRegression(random_state=0).fit(X_train, y_train)
```

```
clf.predict(X_train)
```

```
array(['outbound', 'inbound', 'rental', 'inbound', 'rental', 'rental',
       'rental', 'inbound', 'inbound', 'rental', 'outbound', 'inbound',
       'rental', 'inbound', 'outbound', 'inbound', 'inbound', 'outbound',
       'outbound'], dtype=object)
```

```
clf.predict(X_test)
```

```
array(['outbound', 'rental', 'outbound', 'rental', 'outbound', 'rental',
       'inbound', 'outbound', 'outbound'], dtype=object)
```

```
import joblib
filename = 'LR-model-Contract'
joblib.dump(clf, filename)
```

```
['LR-model-Contract']
```

Step 6: Ml classifier model as a REST API

Once the trained ML classifier model is in place, the classifier can be leveraged as a REST API call by using the flask framework. Import all the necessary libraries used for the model.

```
import json
from flask import Flask, request, jsonify
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

import re
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
import joblib
```

The incoming data to the REST API is raw and needs to be in a proper format before it can be loaded into the saved classifier model.

```
def Clean(text):
    text=text.split()
    text=[i.lower() for i in text if i.lower() not in stopwords.words('english')]
    text=' '.join(text)
    text=re.sub('[^A-Za-z0-9]+',' ',text)
    text=text.lower()
    return text

def Load_Predict(text):
    tfidf = joblib.load("tfidf_model-contract")
    X_pred = tfidf.transform([text]).todense()
    clf = joblib.load("LR-model-Contract")
    prediction = clf.predict(X_pred)
    return prediction[0]
```

Flask instance is created and a POST method is defined for the API.

```
# create an instance of Flask
app = Flask(__name__)



# Define a post method for our API.
@app.route('/classify', methods=['POST'])
def classify():
    text=request.form['text']
    text = Clean(text)
    prediction = Load_Predict(text)
    payload = {
    'prediction': prediction
    }
    response = jsonify(payload)
    return response


if __name__=="__main__":
    app.run(debug=True)
```

This python application file can be run locally to generate a localhost root url which can be accessed and tested by using API testing tools like Postman.

Step 7: Deployment on Heroku
Deployment on heroku or any other cloud application platform makes it essential for the service to be accessible universally. The classifier is deployed on heroku by following the steps in Deploying the app on Heroku.

Step 8: The deployed url is now used in Flogo in the application flow to classify the contracts as inbound, outbound or rental.