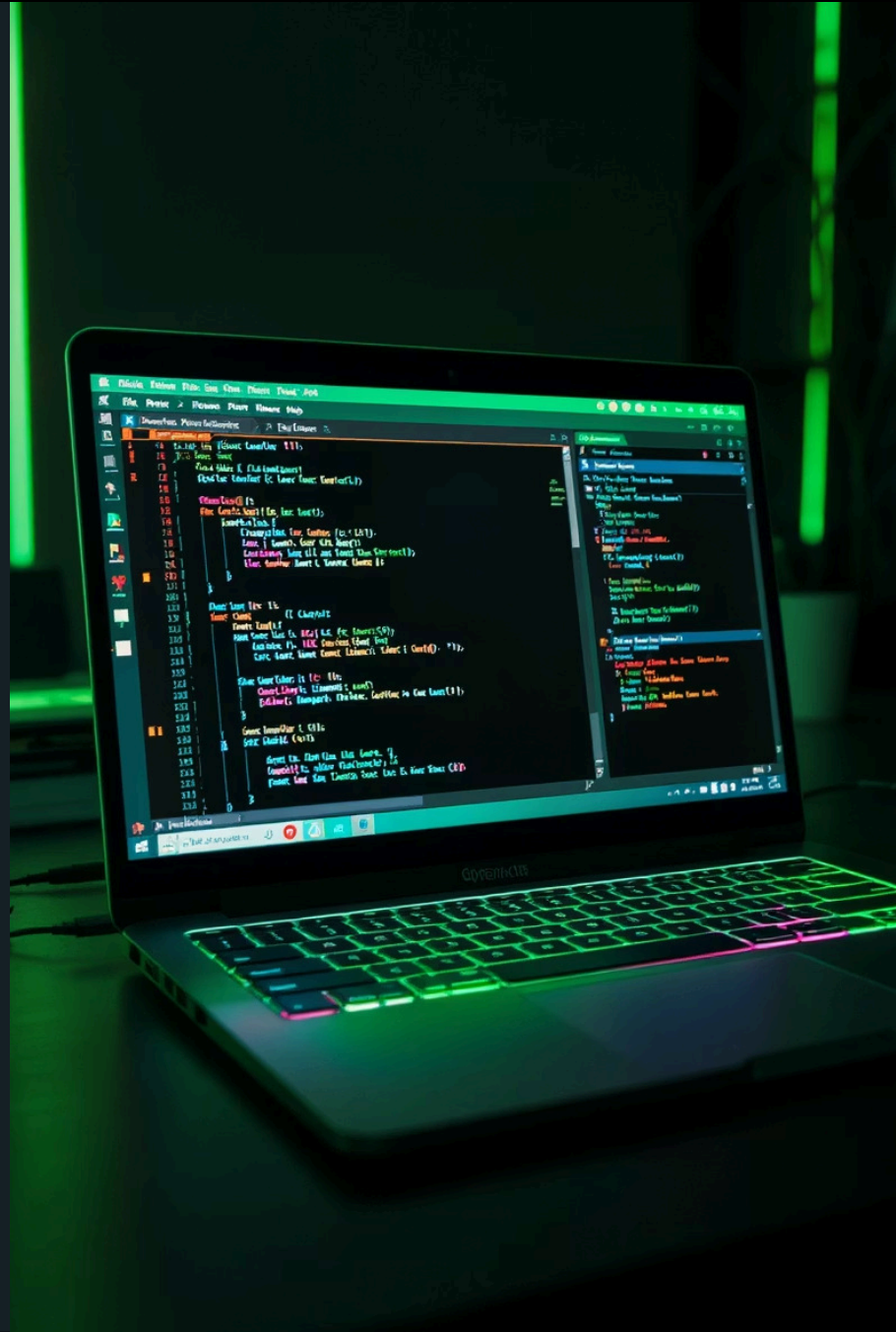
Fine-tuning con OpenAI: Guida Pratica per Principianti

Benvenuti a questo corso introduttivo sul fine-tuning dei modelli OpenAI utilizzando Python. In questa presentazione, esploreremo passo dopo passo il processo di creazione, caricamento e implementazione di un dataset personalizzato per addestrare modelli linguistici avanzati come GPT-4.1-nano.

Il fine-tuning permette di specializzare modelli pre-addestrati per compiti specifici, migliorando significativamente le loro prestazioni in domini particolari. Impareremo insieme come gestire l'intero processo utilizzando la versione 1.87 della libreria OpenAI.



Preparazione del Dataset in Formato JSONL

Il primo passo fondamentale nel processo di fine-tuning è la preparazione di un dataset strutturato correttamente. OpenAI richiede che i dati siano formattati in JSONL (JSON Lines), dove ogni riga rappresenta un esempio di addestramento come oggetto JSON indipendente.

Ogni esempio deve contenere coppie di prompt e completion, dove il prompt è l'input fornito al modello e la completion è la risposta desiderata. È essenziale mantenere una struttura coerente attraverso tutti gli esempi per garantire un addestramento efficace.

1 **Struttura Base**

Ogni riga deve essere un oggetto JSON valido contenente almeno i campi "prompt" e "completion".

```
{"prompt": "Traduci in italiano: Hello world", "completion": "Ciao mondo"}
```

2 **Dimensione del Dataset**

Per risultati ottimali, è consigliabile preparare almeno 100-200 esempi di alta qualità, rappresentativi del compito specifico.

3 **Varietà**

Includere una gamma diversificata di esempi che coprano le varie casistiche che il modello dovrà gestire nella pratica.

Validazione del Dataset con OpenAI Tools

Prima di procedere con il caricamento, è fondamentale validare il dataset per identificare e correggere potenziali problemi. OpenAI fornisce uno strumento specifico per questo scopo che analizza la struttura e il contenuto del file JSONL.

La validazione aiuta a identificare errori di formattazione, problemi di struttura e potenziali incoerenze nei dati che potrebbero compromettere il processo di fine-tuning.

Esecuzione dello Strumento di Validazione

```
python -m openai tools fine_tunes.prepare_data \
  --file training_data.jsonl \
  --quiet
```

Lo strumento analizzerà il file e produrrà un report dettagliato con suggerimenti per migliorare la qualità del dataset.

Problemi Comuni Rilevati

- JSON non valido o malformato
- Campi mancanti (prompt o completion)
- Lunghezza eccessiva dei testi
- Token troncati o incompleti
- Distribuzione sbilanciata degli esempi

Correzione degli Errori nel Dataset

Dopo la validazione, è probabile che lo strumento di OpenAI segnali alcuni errori o avvisi che richiedono attenzione. Questa fase è cruciale per garantire che il processo di fine-tuning proceda senza intoppi.

Le correzioni più comuni includono la risoluzione di problemi di sintassi JSON, l'adeguamento della lunghezza dei testi e il bilanciamento degli esempi per garantire una rappresentazione equa di tutti i casi d'uso.

Analisi degli Errori

Esamina attentamente il report di validazione per identificare tutti i problemi segnalati, ordinandoli per gravità.

Correzione Sistematica

Correggi prima i problemi strutturali (JSON malformato), poi quelli relativi al contenuto (lunghezza, formato).

Rivalidazione

Esegui nuovamente lo strumento di validazione per confermare che tutte le correzioni siano state applicate correttamente.

Finalizzazione

Una volta eliminati tutti gli errori, il file è pronto per essere caricato sulla piattaforma OpenAI.

Upload del Dataset con client.files.create()

Una volta che il dataset è stato correttamente validato e ottimizzato, è il momento di caricarlo sui server OpenAI. Questo processo viene gestito attraverso la API client.files.create() disponibile nella libreria OpenAI v1.87.

Il caricamento è un passaggio necessario prima di poter avviare il processo di fine-tuning, poiché il modello deve avere accesso ai dati di addestramento.

Codice di Esempio per l'Upload

```
from openai import OpenAI

client = OpenAI()

file_response = client.files.create(
    file=open("training_data.jsonl", "rb"),
    purpose="fine-tune"
)

file_id = file_response.id
print(f"File caricato con ID: {file_id}")
```

Parametri Importanti

- **file**: il file JSONL aperto in modalità binaria
- **purpose**: deve essere impostato su "fine-tune" per indicare lo scopo del file

Conserva sempre l'ID del file restituito dalla risposta, sarà necessario per riferirsi a questo dataset nelle fasi successive del processo di fine-tuning.

Verifica dell'ID File Caricato

Dopo aver caricato il file, è fondamentale verificare che l'operazione sia andata a buon fine e conservare correttamente l'identificativo univoco (ID) assegnato al file. Questo ID sarà essenziale per tutte le operazioni successive relative a questo dataset.

OpenAI fornisce metodi per verificare lo stato dei file caricati e confermare che siano pronti per essere utilizzati nel processo di fine-tuning.

Verifica del Caricamento

Controlla che il file sia stato caricato correttamente esaminando l'oggetto di risposta restituito da client.files.create() e verificando che contenga un ID valido.

Elenco dei File

Puoi utilizzare client.files.list() per visualizzare tutti i file caricati e verificare che il tuo dataset sia presente nell'elenco.

```
files = client.files.list()
```

Recupero Dettagli

Per ottenere informazioni dettagliate su un file specifico, usa client.files.retrieve(file_id) inserendo l'ID ottenuto nella fase di caricamento.

Creazione del Job di Fine-tuning

Dopo aver caricato con successo il dataset, il passaggio successivo è creare un job di fine-tuning che addestrerà il modello base (in questo caso gpt-4.1-nano) utilizzando i dati forniti. Questo processo viene gestito attraverso l'API di OpenAI.

La creazione di un job richiede di specificare diversi parametri che influenzeranno il processo di addestramento e la qualità del modello risultante.

Codice per Avviare il Fine-tuning

```
from openai import OpenAI

client = OpenAI()

job = client.fine_tuning.jobs.create(
    training_file=file_id,
    model="gpt-4.1-nano",
    hyperparameters={
        "n_epochs": 4
    }
)

print(f"Job creato con ID: {job.id}")
```

Parametri Essenziali

- **training_file**: l'ID del file JSONL caricato
- **model**: il modello base da utilizzare (gpt-4.1-nano)
- **hyperparameters**: impostazioni che controllano il processo di addestramento

Il parametro n_epochs determina quante volte il modello passerà attraverso l'intero dataset durante l'addestramento. Un valore troppo basso può portare a un addestramento insufficiente, mentre un valore troppo alto può causare overfitting.

Monitoraggio dello Stato del Job

Una volta avviato il job di fine-tuning, è importante monitorarne lo stato per verificare che proceda correttamente e per essere pronti a intervenire in caso di problemi. Il processo può richiedere da pochi minuti a diverse ore, a seconda della dimensione del dataset e della complessità del modello.

OpenAI fornisce API specifiche per controllare lo stato di avanzamento e ottenere informazioni dettagliate sul processo in corso.

1 Controllo Periodico

```
job_status = client.fine_tuning.jobs.retrieve(job.id)
print(f"Stato: {job_status.status}")
```

Questa chiamata restituisce lo stato attuale del job, che può essere uno tra: "queued", "running", "succeeded", "failed", o "cancelled".

2 Recupero degli Eventi

```
events = client.fine_tuning.jobs.list_events(
    fine_tuning_job_id=job.id
)
for event in events.data:
    print(f"{event.created_at}: {event.message}")
```

Gli eventi forniscono informazioni più dettagliate sull'avanzamento, inclusi eventuali problemi o warning.

3 Attesa del Completamento

È possibile implementare un ciclo di polling che controlli periodicamente lo stato fino al completamento, con pause appropriate per non sovraccaricare l'API.

Gestione di Job Falliti o con Warning

Durante il processo di fine-tuning, possono verificarsi diversi tipi di problemi che generano warning o, nei casi più gravi, causano il fallimento completo del job. Saper interpretare e gestire queste situazioni è fondamentale per ottimizzare il processo.

I problemi più comuni includono dataset mal formattati, hyperparameters non ottimali, o limitazioni delle risorse disponibili.

Warning Comuni

- Distribuzione sbilanciata degli esempi nel dataset
- Lunghezza eccessiva di alcuni prompt o completion
- Numero insufficiente di esempi per un addestramento efficace
- Token troncati o non correttamente tokenizzati

Cause di Fallimento

- Dataset corrotto o non conforme alle specifiche
- Errori di sintassi nel file JSONL
- Problemi di autorizzazione o limiti di quota API
- Interruzioni del servizio o timeout durante l'elaborazione

Strategie di Risoluzione

- Rianalizzare e correggere il dataset seguendo i suggerimenti degli errori
- Modificare gli hyperparameters in base ai warning ricevuti
- Suddividere dataset molto grandi in parti più gestibili
- Verificare lo stato dell'account e delle quote disponibili

Verifica del Modello Generato

Una volta completato con successo il processo di fine-tuning, è fondamentale verificare che il modello generato sia disponibile e funzionante. OpenAI assegna al modello un identificativo univoco che combina il nome del modello base con un suffisso personalizzato.

Questa fase di verifica permette di confermare che il modello sia stato correttamente creato e sia pronto per essere utilizzato nelle applicazioni.

Recupero delle Informazioni sul Modello

```
# Recupera i dettagli del job completato
job_info = client.fine_tuning.jobs.retrieve(job.id)

# Ottieni l'ID del modello fine-tuned
fine_tuned_model = job_info.fine_tuned_model

print(f"Modello fine-tuned: {fine_tuned_model}")
```

Il campo fine_tuned_model sarà disponibile solo quando il job è stato completato con successo.

Elenco dei Modelli Disponibili

```
# Ottieni l'elenco completo dei modelli disponibili
models = client.models.list()

# Filtra per trovare i tuoi modelli fine-tuned
your_models = [m for m in models.data
               if m.id.startswith("ft:")]

for model in your_models:
    print(f"- {model.id}")
```

Questo codice permette di visualizzare tutti i modelli fine-tuned disponibili per il tuo account, confermando che il nuovo modello sia accessibile.

Creazione di Chiamate Test sul Modello Fine-tuned

Dopo aver verificato che il modello fine-tuned sia disponibile, è essenziale testarlo con una serie di prompt di esempio per valutare le sue prestazioni. Questa fase permette di capire se il modello ha effettivamente appreso i pattern desiderati dal dataset di addestramento.

I test dovrebbero includere sia casi presenti nel dataset di addestramento, sia nuovi casi che il modello non ha mai visto prima, per verificare la sua capacità di generalizzazione.

Implementazione dei Test

```
response = client.chat.completions.create(
    model=fine_tuned_model,
    messages=[
        {"role": "system", "content": "Sei un assistente utile."},
        {"role": "user", "content": "Prompt di test..."}
    ],
    temperature=0.7
)
print(response.choices[0].message.content)
```

Criteri di Valutazione

- Accuratezza delle risposte rispetto alle aspettative
- Coerenza con lo stile e il formato del dataset
- Capacità di gestire variazioni nei prompt
- Robustezza a input imprevisti o edge case

Parametri da Sperimentare

- **temperature**: controlla la creatività (0.2-0.8)
- **max_tokens**: limita la lunghezza della risposta
- **top_p**: alternativa alla temperatura per il controllo della diversità

Confronto Output: Modello Base vs Fine-tuned

Un aspetto cruciale della valutazione è confrontare direttamente le prestazioni del modello fine-tuned con quelle del modello base (gpt-4.1-nano). Questo confronto permette di quantificare il miglioramento ottenuto attraverso il processo di personalizzazione.

Il confronto dovrebbe essere metodico, utilizzando gli stessi prompt per entrambi i modelli e valutando le differenze nelle risposte generate.

Codice per il Confronto

```
def compare_models(prompt, models):
    results = {}
    for model_name in models:
        response = client.chat.completions.create(
            model=model_name,
            messages=[
                {"role": "user", "content": prompt}
            ],
            temperature=0.5
        )
        results[model_name] = response.choices[0].message.content
    return results

test_prompt = "Prompt di test..."
models_to_compare = ["gpt-4.1-nano", fine_tuned_model]
comparison = compare_models(test_prompt, models_to_compare)

for model, output in comparison.items():
    print(f"\n--- {model} ---\n{output}")
```

Aspetti da Valutare

- **Rilevanza**: quanto la risposta è pertinente al prompt
- **Specificità**: capacità di fornire dettagli rilevanti al dominio
- **Formato**: aderenza al formato desiderato nella risposta
- **Tono**: adeguatezza dello stile e del tono utilizzati
- **Accuratezza**: correttezza delle informazioni fornite
- **Coerenza**: logica interna e fluidità della risposta

Considera di creare una griglia di valutazione numerica per quantificare il miglioramento su ciascun aspetto.

Logging degli Output di Test

Registrare sistematicamente i risultati dei test è fondamentale per documentare le prestazioni del modello fine-tuned e tracciare i miglioramenti nel tempo. Un buon sistema di logging permette di analizzare i pattern di successo e fallimento, facilitando l'iterazione e l'ottimizzazione continua.

Il logging dovrebbe includere non solo gli output generati, ma anche metadati come timestamp, parametri utilizzati e metriche di valutazione.

Implementazione di un Logger

```
import json
import datetime

def log_test_result(prompt, response, model_name):
    timestamp = datetime.datetime.now().isoformat()
    log_entry = {
        "timestamp": timestamp,
        "model": model_name,
        "prompt": prompt,
        "response": response,
        "parameters": {
            "temperature": 0.7,
            "max_tokens": 500
        }
    }

    with open("test_logs.jsonl", "a") as f:
        f.write(json.dumps(log_entry) + "\n")
```

Analisi dei Log

I log possono essere analizzati per:

- Identificare pattern nei casi in cui il modello ha prestazioni inferiori
- Misurare il miglioramento delle prestazioni nel tempo
- Raccogliere esempi per futuri cicli di fine-tuning
- Identificare bias o comportamenti problematici

Considera l'utilizzo di strumenti come Pandas per analizzare i log in formato JSON Lines, trasformandoli in dataframe per facilitare l'analisi statistica e la visualizzazione.

Documentazione del Processo

Documentare accuratamente l'intero processo di fine-tuning è essenziale per la riproducibilità, la condivisione con il team e il miglioramento continuo. Una buona documentazione permette di comprendere le scelte fatte, i risultati ottenuti e le lezioni apprese.

La documentazione dovrebbe essere strutturata in modo da essere utile sia ai membri tecnici del team che agli stakeholder non tecnici interessati ai risultati del progetto.

Elementi Chiave da Documentare

- Struttura e fonte del dataset originale
- Processo di preparazione e pulizia dei dati
- Parametri di fine-tuning scelti e motivazione
- ID di file e modelli per riferimento futuro
- Risultati dei test e confronti con il modello base
- Problemi incontrati e soluzioni adottate

Formati di Documentazione

- **README**: per una panoramica generale del progetto
- **Jupyter Notebook**: per documentare codice e output con spiegazioni
- **Diagrammi di flusso**: per visualizzare il processo end-to-end
- **Wiki interna**: per documentazione collaborativa e in evoluzione

Best Practice

- Utilizzare il controllo versione per la documentazione
- Includere snippet di codice commentati
- Aggiungere esempi concreti di input/output
- Documentare anche i tentativi falliti e le lezioni apprese

Gestione e Pulizia dei File Caricati

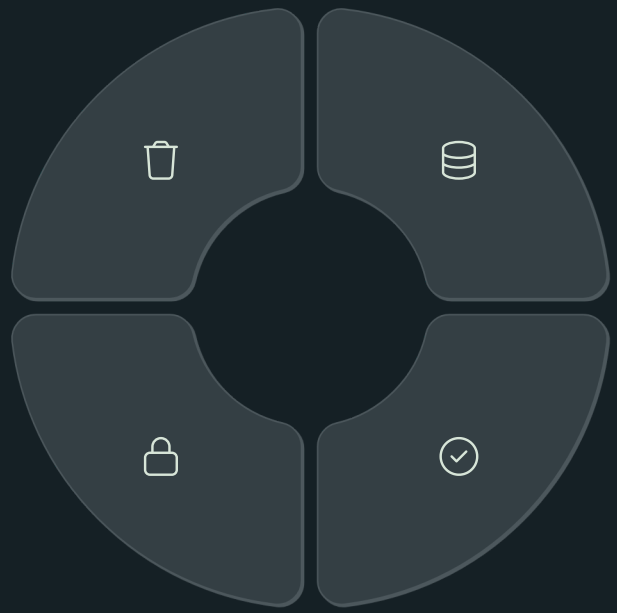
Una gestione efficiente dei file caricati e dei modelli fine-tuned è essenziale per ottimizzare i costi, mantenere l'ordine e rispettare le best practice di sicurezza dei dati. OpenAI addebita costi per lo storage dei file e per l'uso dei modelli fine-tuned.

È importante implementare procedure regolari per rivedere, archiviare o eliminare file non più necessari, specialmente in ambienti di produzione o con risorse limitate.

Eliminazione File

```
# Elimina un file specifico
client.files.delete(file_id)

# Verifica che sia stato eliminato
try:
client.files.retrieve(file_id)
print("File ancora presente")
except:
print("File eliminato con successo")
```



Archiviazione Dataset

Mantieni copie locali dei dataset in un repository versionato come Git, etichettate con le informazioni sui modelli che hanno generato.

- Utilizza tag e branch per versioni diverse del dataset
- Documenta le modifiche tra versioni successive

Sicurezza Dati

Assicurati che i dataset non contengano dati sensibili o personali prima del caricamento, e implementa politiche di conservazione conformi alle normative sulla privacy.

Monitoraggio Risorse

Implementa script per monitorare regolarmente i file e i modelli attivi, segnalando quelli inutilizzati da lungo tempo.

- Automatizza report settimanali o mensili
- Imposta soglie di età per revisione