



DIPARTIMENTO DI INGEGNERIA INFORMATICA, MODELLISTICA,
ELETTRONICA E SISTEMISTICA

Corso di Laurea Magistrale in Ingegneria Informatica

PROGETTO DI SISTEMI DISTRIBUITI E CLOUD COMPUTING



Studente:

Gallicchio Vittorio

Mat. 263726

Docenti:

Prof. Domenico Talia

Prof. Loris Belcastro

ANNO ACCADEMICO 24/25

Indice

Indice	1
1 Introduzione	2
2 Analisi dei requisiti	5
2.1 Requisiti funzionali	5
2.2 Requisiti non funzionali	7
3 Implementazione del sistema	8
3.1 Servizi cloud utilizzati	8
3.1.1 Architettura del sistema	10
3.2 Backend	11
3.2.1 API esposte tramite HTTP Trigger	11
3.2.2 Pipeline MLOps con Event Grid Trigger	15
3.3 Frontend	17
3.3.1 Struttura	19
3.3.2 Componenti	19
3.3.3 Servizi	20
3.3.4 Hosting del frontend	20
3.4 Flusso di esecuzione	21
3.5 Simulazione esecuzione	22
4 Conclusioni	26
4.1 Risultati raggiunti	26
4.2 Considerazioni personali	27

Capitolo 1

Introduzione

L'evoluzione del **cloud computing** e la diffusione delle architetture **serverless** hanno trasformato radicalmente il modo in cui le applicazioni vengono progettate, sviluppate e distribuite. In particolare, il paradigma **Function-as-a-Service (FaaS)** consente di eseguire codice in risposta a eventi, eliminando la necessità di gestire manualmente server o macchine virtuali, con un modello di scalabilità automatica e pagamento a consumo.

Il presente progetto ha come obiettivo la realizzazione di una **pipeline di Machine Learning completamente automatizzata**, sfruttando le potenzialità offerte dalla piattaforma **Microsoft Azure**. Il sistema utilizza un approccio **event-driven**, in cui il caricamento di un dataset sullo storage cloud funge da evento di partenza per l'intera sequenza di elaborazione.

Il dataset utilizzato è stato il **Car Evaluation Dataset**, il quale è un dataset pubblico largamente impiegato in letteratura come benchmark per la classificazione supervisionata. Tale dataset descrive differenti caratteristiche di autovetture (come prezzo, manutenzione, dimensioni e sicurezza) e ha lo scopo di predire una valutazione qualitativa complessiva. La scelta di questo dataset risulta funzionale a dimostrare le potenzialità di un approccio MLOps applicato a scenari concreti.

Dal sito del dataset, sono state rese note le metriche di **accuracy** e **precision** per i vari modelli utilizzati. La loro visione mi ha indirizzato ad usare come modello da addestrare una **rete neurale**

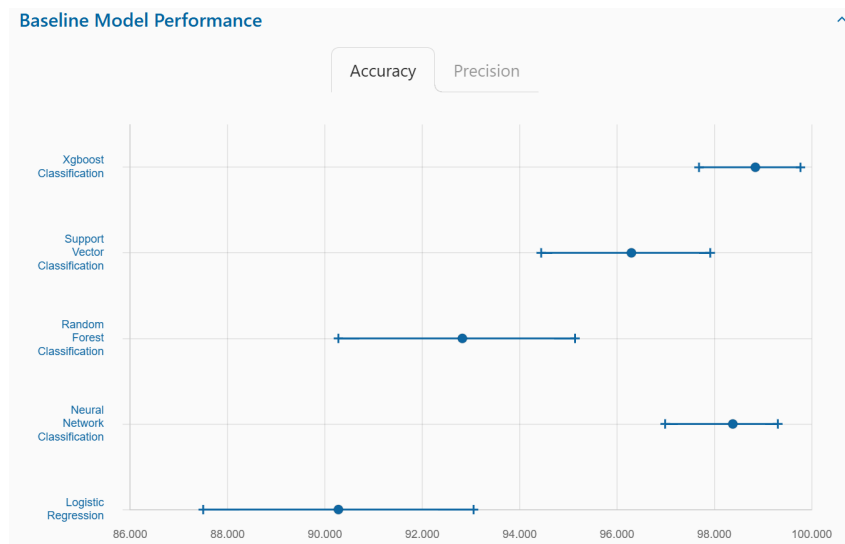


Figura 1.1: Metriche di *accuracy*

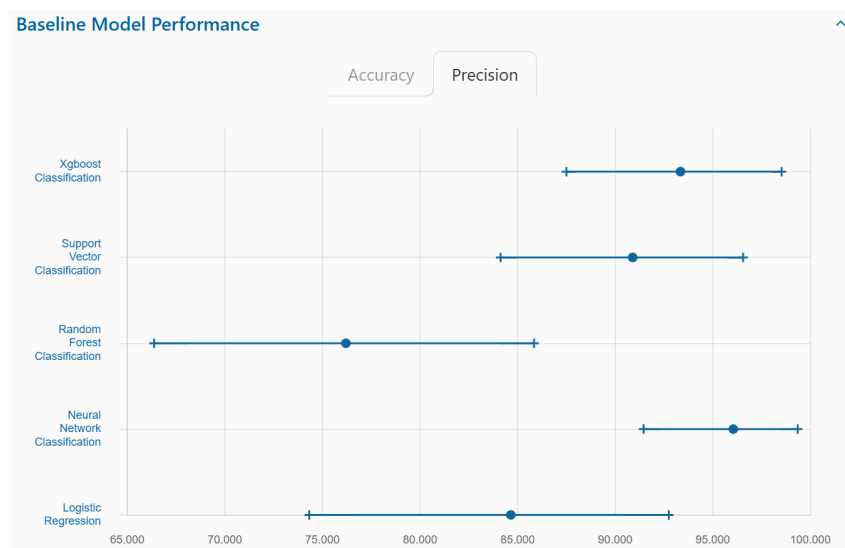


Figura 1.2: Metriche di *precision*

Il flusso di elaborazione è stato suddiviso in quattro fasi principali:

- **Caricamento del dataset:** l'utente carica il dataset in Azure Blob Storage, attivando automaticamente la pipeline;
- **Pre-elaborazione dei dati:** pulizia, normalizzazione e codifica delle *feature*;
- **Addestramento del modello di Machine Learning:** utilizzo di una **rete neurale** come modello, il quale è stato addestrato con l'aiuto della libreria **scikit-learn**;
- **Inferenza in tempo reale su VM:** utilizzo di una macchina virtuale Azure per ospitare un **Frontend web** che consenta all'utente di interagire con il sistema e

visualizzare i risultati delle predizioni, e permetta agli utenti amministratori di caricare il dataset e dare il via alla *pipeline*.

L'obiettivo è fornire una soluzione che garantisca automazione, affidabilità e facilità di manutenzione, sfruttando al meglio le potenzialità offerte dall'ecosistema Azure in termini di calcolo serverless, storage distribuito e scalabilità.

L'adozione di un'architettura serverless permette di ottenere vantaggi significativi in termini di scalabilità, affidabilità e costi, riducendo al minimo il lavoro di gestione dell'infrastruttura sottostante. Ciò ha reso possibile concentrarsi sugli aspetti logici del flusso applicativo. Allo stesso tempo, il progetto si inserisce nel più ampio contesto delle pratiche di MLOps, con l'obiettivo di integrare metodologie di sviluppo software e machine learning in un unico framework coerente e ripetibile.

Capitolo 2

Analisi dei requisiti

2.1 Requisiti funzionali

I requisiti funzionali definiscono le specifiche e le funzionalità che il sistema deve offrire. Per il progetto in questione, a seguito della lettura della traccia, sono state identificate le seguenti aree funzionali:

- **Caricamento del dataset**
- **Pre-elaborazione dei dati**
- **Addestramento del modello**
- **Servizio di inferenza**

Inoltre, per avere una migliore *user experience* per quanto riguarda le predizioni, è stata introdotta un'ulteriore area funzionale legata alla **Gestione delle predizioni**, la quale implementa alcune funzionalità accessorie.

Per ogni area troviamo i seguenti requisiti funzionali:

1. Caricamento del Dataset

- (a) Il sistema deve consentire l'**upload** di un dataset da parte dell'utente amministratore;
- (b) Il sistema deve invocare automaticamente la **funzione FaaS** di pre-elaborazione dei dati al completamento dell'upload.

2. Pre-elaborazione dei dati

- (a) Il sistema deve essere in grado di **leggere** il dataset caricato.
- (b) Il sistema deve eseguire la **pulizia** del dataset, comprensiva di:
 - rimozione di valori nulli;
 - gestione dei dati mancanti secondo regole predefinite.
- (c) Il sistema deve effettuare la **trasformazione** del dataset, comprendente:
 - codifica delle feature categoriche.
- (d) Il sistema deve **salvare** il dataset pre-elaborato in una nuova posizione nello storage, distinta da quella del dataset originale.

3. Addestramento del modello

- (a) Il sistema deve attivare la **funzione di training** automaticamente al caricamento del dataset pulito sullo storage;
- (b) La funzione di training deve permettere di addestrare un modello sulla base del dataset;
- (c) Il sistema deve **salvare il modello addestrato** in formato serializzato in una posizione specifica dello storage.

4. Servizio di inferenza

- (a) Il sistema deve implementare un servizio di login per distinguere i normali utenti dagli utenti amministratori, mostrando solo le rispettive funzionalità;
- (b) Il sistema deve offrire un **frontend** che permetta all'utente di:
 - Effettuare nuove predizioni;
 - Caricare il dataset dataset, previa login e verifica del ruolo dell'utente.
- (c) Il sistema deve esporre un **endpoint HTTP POST** che consenta di inviare nuove istanze di dati e ricevere come risposta una predizione in tempo reale;
- (d) Il sistema deve esporre un **endpoint HTTP POST** per consenta l'upload del dataset e lo salvi all'interno dello storage.

5. Gestione delle predizioni

- (a) Il sistema deve permettere agli utenti, previa login, di:
 - Salvare una predizione fatta

- Visualizzare le predizioni salvata
- Filtrare le predizioni salvate in base all'attributo di *classe*
- Eliminare una predizione salvata

2.2 Requisiti non funzionali

I requisiti non funzionali descrivono le caratteristiche e i vincoli del sistema che influenzano la sua qualità e le sue prestazioni. Essi non riguardano direttamente le funzionalità del sistema, ma sono altrettanto importanti per garantire che il sistema sia efficiente, sicuro e facile da usare. Di seguito sono elencati i requisiti non funzionali per il progetto:

- **Scalabilità:** L'architettura deve poter gestire variazioni di carico in maniera automatica, scalando il numero di funzioni serverless in base al volume delle richieste;
- **Affidabilità e Disponibilità:** Il sistema deve garantire un'elevata disponibilità dei servizi, attraverso l'uso dei servizi di Azure, riducendo al minimo i tempi di inattività. Inoltre, eventuali errori devono essere gestiti in modo tale da garantire la continuità del flusso di lavoro;
- **Performance:** Le operazioni di preprocessing e addestramento devono essere eseguite in tempi ragionevoli, compatibilmente con le risorse allocate dalle funzioni di Azure e, le richieste di inferenza devono restituire una predizione in tempo reale (latenza ridotta);
- **Manutenibilità:** Il codice deve essere organizzato in moduli chiari, leggibili e ben documentato, separando il codice per funzione e per componenti, favorendo la possibilità di estendere o modificare singole componenti;
- **Sicurezza:** L'accesso al blob storage ed al database deve essere protetto e deve essere poter effettuato solo da servizi stessi di Azure (il backend serverless). I dati salvati devono essere trattati in conformità alle buone pratiche di sicurezza fornite direttamente da Azure;

In più, il sistema deve implementare una fase di autenticazione per consentire solo agli utenti autorizzati di accedere alla funzionalità di caricamento del dataset;

Capitolo 3

Implementazione del sistema

L'architettura dell'applicazione è stata suddivisa in due macro-componenti principali:

- **Backend:** realizzato attraverso le Azure function e deployato attraverso una Azure Function App
- **Frontend:** realizzato in Angular ed hostato su una Azure Virtual Machine

3.1 Servizi cloud utilizzati

Il sistema è stato implementato sfruttando i seguenti **servizi di Microsoft Azure**:

- **Azure Functions**

Il sistema ha implementato una **Azure Function App** che ha preso il **ruolo di backend** dell'applicazione. Dunque, al suo interno sono state definite le *azure functions* sviluppate per eseguire tutte le funzionalità richieste.

Dunque, utilizzando le *Azure function* il backend è stato implementato in modo **serverless**, permettendo di esporre le API per la Fase 4, ma anche di intercettare gli eventi sul *blob storage*, così da eseguire automaticamente le funzioni associate alle Fasi 2 e 3.

I **trigger** usati sono stati:

- **HTTP Trigger:** Definendo delle *route* di attivazione, ha permesso di esporre degli endpoint HTTP da richiamare lato frontend, attivando ed eseguendo la funzione associata alla *route*

- **Event Grid Trigger:** Hanno reso possibile intercettare gli eventi sul *blob storage*, così eseguire le funzioni associate alla Fase 2 ed alla Fase 3.

- **Azure Blob Storage**

Il sistema ha utilizzato **Azure Blob Storage** per memorizzare, sotto forma di **blob**, il dataset originale, il dataset trasformato ed il modello addestrato. In particolare, sono stati creati 3 container:

1. **Raw:** memorizza il dataset "*grezzo*", ovvero così come viene caricato dall'utente amministratore;
2. **Clean:** memorizza il dataset pulito e trasformato a seguito della funzione di *pre-processing*;
3. **Model:** memorizza gli artefatti prodotto a seguito della pipeline di Machine Learning, ovvero il modello addestrato ed il preprocessor, entrambi memorizzati attraverso serializzazione.

- **Event Grid**

Il sistema ha usufruito del servizio di **Event grid** per intercettare sul *blob storage* sul gli eventi di:

- Caricamento del dataset "*grezzo*"
- Caricamento del dataset "*pulito*"

Associando ad ogni evento la relativa *azure function* da eseguire. In particolare:

1. Il primo *event grid* è servito a **rilevare l'upload del dataset** sul blob storage (Fase 1) ed attivare la **funzione di preprocessing** (Fase 2);
2. Il secondo *event grid* è servito a **rilevare l'inserimento del dataset pulito** sul blob storage ed attivare la **funzione di training** (Fase 3).

- **Azure Virtual Machines**

Il sistema ha disposto dell'utilizzo di una *macchina virtuale* per effettuare l'hosting del frontend. L'uso di *Azure Virtual Machine* ha inoltre permesso di associare alla macchina virtuale un **indirizzo IP pubblico** ed un **dominio pubblico**, rendendo l'applicazione fruibile da tutti su internet.

- **Server flessibile di Database di Azure per MySQL**

Per memorizzare i dati legati alle funzionalità di *gestione delle predizioni* il sistema ha utilizzato un **database MySQL**.

Dunque, si sono create le tabelle:

1. **Users:** (email, name)

La tabella serve a memorizzare nome ed email degli utenti dell'applicazione, in sincronizzazione con il servizio di autenticazione **Auth0**. Infatti, la gestione del login è affidata al servizio **Auth0**. In questo modo è stato possibile legare agli utenti le predizioni che essi salvano;

2. **Prediction:** (id, user_email, buying, maint, doors, persons, lug_boot, safety, class)

La tabella serve a memorizzare le predizioni salvate, dove ogni predizione è associata al proprio utente che l'ha memorizzata.

Infine, per motivi di sicurezza si è reso accessibile il database solo da altri servizi di Azure. Di fatto, solo le azure function del backend hanno la necessità di dover accedere al database.

3.1.1 Architettura del sistema

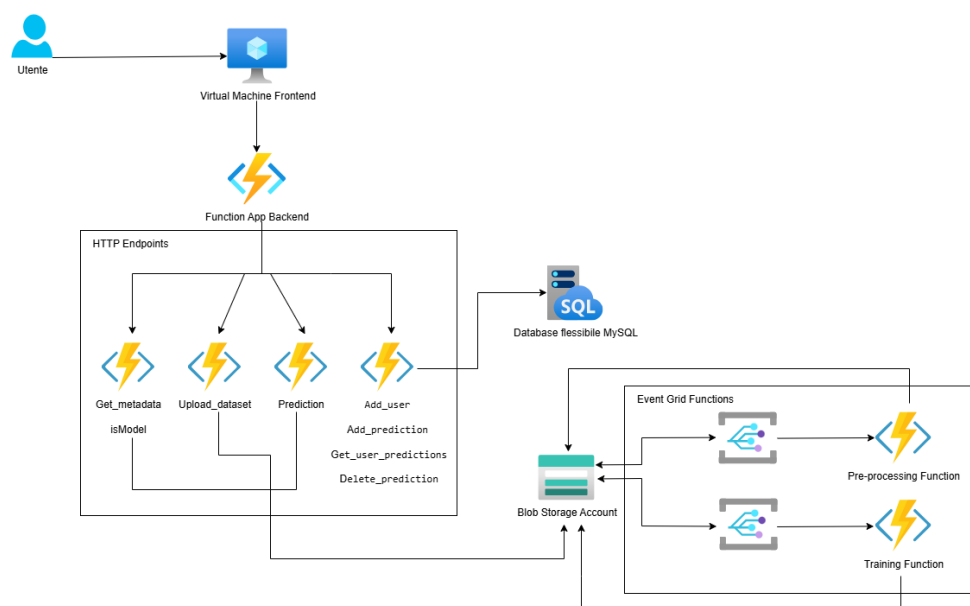


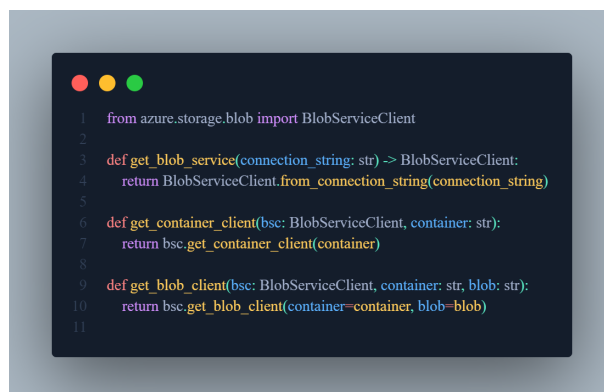
Figura 3.1: Immagine riassuntiva dell'architettura del sistema

3.2 Backend

Il backend, realizzato attraverso le Azure functions, quindi, con un'architettura serverless, è stato pensato e strutturato in maniera modulare, gestendo al meglio la logica applicativa, l'elaborazione dei dati e la comunicazione con i servizi cloud.

La realizzazione del backend ha previsto:

- L'utilizzo di **python** come linguaggio di programmazione
- L'utilizzo delle librerie:
 - Machine Learning: `pandas`, `numpy`, `scikit-learn`, `joblib`;
 - Azure SDK: `azure-functions`, `azure-storage-blob`
 - Database: `mysql-connector-python`
 - Altro (parsing dell'upload del file): `requests-toolbelt`
- L'implementazione di tutte le funzioni necessarie a seguito dell'analisi dei requisiti
- L'implementazione di alcuni script python di utilità, tra cui alcune funzioni di utilità per accedere al blob storage tramite SDK



```
1 from azure.storage.blob import BlobServiceClient
2
3 def get_blob_service(connection_string: str) -> BlobServiceClient:
4     return BlobServiceClient.from_connection_string(connection_string)
5
6 def get_container_client(bsc: BlobServiceClient, container: str):
7     return bsc.get_container_client(container)
8
9 def get_blob_client(bsc: BlobServiceClient, container: str, blob: str):
10    return bsc.get_blob_client(container=container, blob=blob)
11
```

3.2.1 API esposte tramite HTTP Trigger

Di seguito la lista con descrizione delle API esposte tramite HTTP trigger. Per semplicità si riporta l'immagine del codice solo delle funzioni più importanti.

- `/prediction` (POST) [Figura 3.2]
 - **Input:** Un JSON {buying, maint, doors, persons, lug_boot, safety};

- **Output:** Un JSON con classe predetta (`acc`, `good`, `vgood`, `unacc`);
 - **Descrizione:** Esegue l'inferenza sul modello generando la predizioni sulla base dei dati in input. Dunque, la funzione carica il modello ed il preprocessor, prendendoli dal container `model/` sul *blob storage* ed effettua la predizione tramite la funzione `.predict()` della libreria *scikit-learn* e ne restituisce la classe di output della predizione.
- **/upload_dataset** (POST, multipart/form-data) [Figura 3.3]
 - **Input:** Un file in formato (.csv o .data)
 - **Output:** Status code 200 / 400 / 500
 - **Descrizione:** Carica un nuovo dataset in `raw/car.data`.
 - **/get_metadata** (GET)
 - **Input:** —
 - **Output:** Un JSON {name, size, last_modified, n_row, n_col}
 - **Descrizione:** Restituisce metadati associati al dataset per la visualizzazione lato frontend.
 - **/add_user** (POST)
 - **Input:** Un JSON {email, name}
 - **Output:** Status code 200 / 400 / 500
 - **Descrizione:** Verifica la presenza dell'utente all'interno del sistema; in caso contrario effettua l'inserimento dell'utente nel database. Il metodo serve a sincronizzare gli utenti registrati con il servizio Auth0.
 - **/add_prediction** (POST)
 - **Input:** Un JSON {user_email, buying, maint, doors, persons, lug_boot, safety, class}
 - **Output:** Status code 200 / 500
 - **Descrizione:** Permette di salvare in modo persistente all'interno del database una predizione effettuata dall'utente.

```

1  @prediction.route(route="prediction", methods=["POST"])
2  def prediction(req: func.HttpRequest) -> func.HttpResponse:
3      logging.info("Python HTTP trigger. Esecuzione della predizione")
4
5      logging.info("Connessione ad Azure Blob Storage...")
6
7      # Lettura del modello e del preprocessing dei dati
8      connection_string = os.environ["AzureWebJobsStorage"]
9      bsc = get_blob_service(connection_string)
10
11     model_blob = bsc.get_blob_client(container=MODEL_CONTAINER, blob=MODEL_NAME).download_blob().readall()
12     preprocessor_blob = bsc.get_blob_client(container=MODEL_CONTAINER, blob=PREPROCESSOR_NAME).download_blob().readall()
13     model = joblib.load(io.BytesIO(model_blob))
14     preprocessor = joblib.load(io.BytesIO(preprocessor_blob))
15
16     if model is None or preprocessor is None:
17         return func.HttpResponse(
18             json.dumps({"error": "Modello non presente"}),
19             status_code=400,
20             mimetype="application/json"
21         )
22     logging.info("Modello e preprocessor caricati in memoria.")
23
24     try:
25         # Lettura dei dati nella richiesta. Dati provenienti come JSON
26         try:
27             data = req.get_json()
28         except ValueError:
29             return func.HttpResponse(
30                 json.dumps({"error": "Body non in formato JSON"}),
31                 status_code=400,
32                 mimetype="application/json"
33             )
34         if not data:
35             return func.HttpResponse(
36                 json.dumps({"error": "Nessun dato ricevuto"}),
37                 status_code=400,
38                 mimetype="application/json"
39             )
40         logging.info("Dati letti correttamente.")
41         logging.info(str(data))
42
43         # Esecuzione della predizione
44         df = pd.DataFrame([data])
45         df_transformed = preprocessor.transform(df)
46         if hasattr(preprocessor, "get_feature_names_out"):
47             feature_names = preprocessor.get_feature_names_out()
48             df_transformed = pd.DataFrame(df_transformed.toarray(), columns=feature_names)
49
50         pred = model.predict(df_transformed)
51
52         logging.info("Predizione avvenuta correttamente")
53         return func.HttpResponse(
54             json.dumps({"prediction": pred[0]}),
55             status_code=200,
56             mimetype="application/json"
57         )
58
59     except Exception as e:
60         return func.HttpResponse(
61             json.dumps({"error": str(e)}),
62             status_code=500,
63             mimetype="application/json"
64         )
65

```

Figura 3.2: Function di predizione

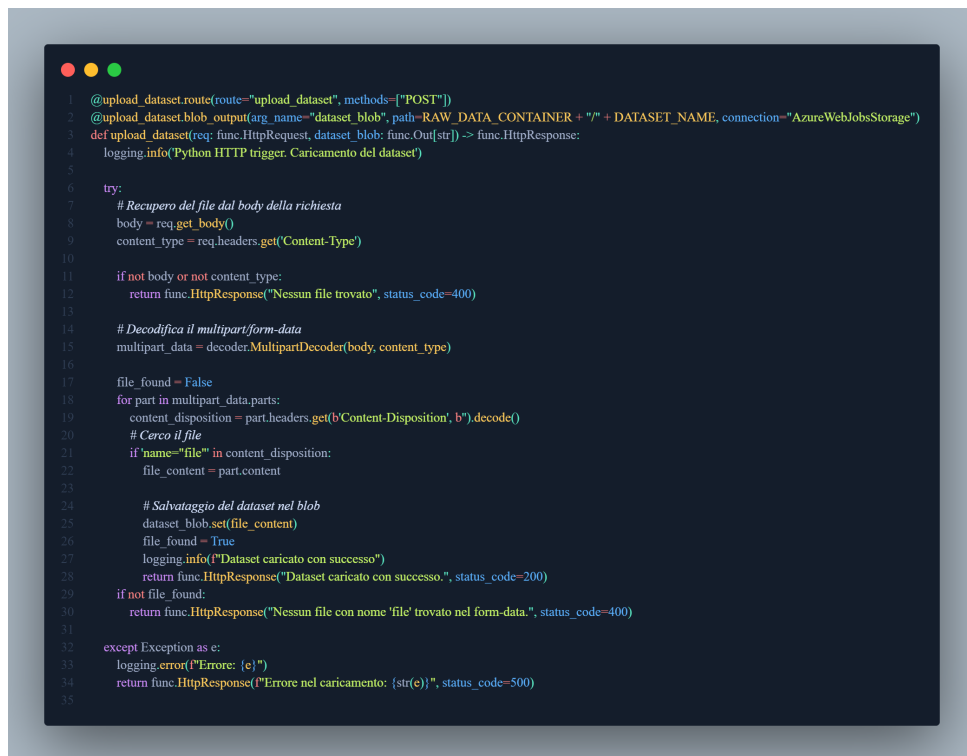


Figura 3.3: Function di upload del dataset

- **/get_user_predictions** (GET)
 - **Input:** Query string {email, page, opz. class}
 - **Output:** Un JSON {page, total_pages, predictions: [lista]}
 - **Descrizione:** Restituisce la lista delle predizioni salvate da un utente, con paginazione (page size = 6) e filtro opzionale per classe.
- **/delete_prediction** (DELETE)
 - **Input:** Query string {id}
 - **Output:** 200 testo / 404
 - **Descrizione:** Permette di eliminare dal database una predizione precedentemente salvata dall'utente ed identificata dal proprio id.
- **/isModel** (GET)
 - **Input:** —
 - **Output:** Status code 200 / 404 / 500
 - **Descrizione:** Verifica l'esistenza e la disponibilità del modello addestrato, per visualizzarne lo stato lato frontend.

3.2.2 Pipeline MLOps con Event Grid Trigger

- **Preprocessing Function** [Figura 3.4]

- **Trigger: Upload del dataset**, ovvero: BlobCreated sul container `raw/` con estensione `.csv` o `.data`
- **Descrizione:** Il caricamento del dataset funge da trigger ed attiva la funzione, la quale legge il dataset ed inizia il processo di pulizia e trasformazione. Dunque, vengono aggiunti i valori delle colonne al dataset, vengono rimosse le righe con valori nulli e le feature vengono separate dal *target*.

Le feature sono state gestite attraverso l'utilizzo del `ColumnTransformer` e dell'`OneHotEncoder`. Quindi, si ha che il dataset viene trasformato e finalizzato. Infine, si ha la serializzazione ed il salvataggio del preprocessor (`preprocessor.joblib`) e, l'upload del dataset pulito

Nota: Si salva anche il *preprocessor*, separatamente dal modello, per garantire coerenza delle feature in fase di inferenza.

- **Output:** Salvataggio del dataset pulito sotto forma di blob all'interno del container `clean/` e salvataggio del *preprocessor* sotto forma di blob all'interno del container `model/`

- **Training Function** [Figura 3.5]

- **Trigger: Upload del dataset trasformato e pulito**, ovvero: BlobCreated sul container `model/` con estensione `.csv`
- **Descrizione:** Il caricamento del dataset "pulito" funge da trigger ed attiva la funzione, la quale legge il nuovo dataset ed inizia la il processo di training.

In particolare il modello, scelto a seguito della visione delle metriche sul sito del dataset, ed utilizzato per l'addestramento è stato quello di **rete neurale**, utilizzando la classe `MLPClassifier` di *scikit-learn*, con configurazione:

- * **Architettura:** 20 neuroni + 10 neuroni

La rete neurale è composta da più livelli:

- Input layer: riceve le variabili del dataset (es. `buying`, `maint`, `doors`...)
- Hidden layers: 20 neuroni + 10 neuroni


```

1  @preprocessfunction.function_name(name="preprocessingFunctionEventGrid")
2  @preprocessfunction.event_grid_trigger(arg_name="event", auth_level=func.AuthLevel.ANONYMOUS)
3  def preprocessfunction(event: func.EventGridEvent):
4      logging.info(f"Evento ricevuto: {event.event_type}")
5
6      # Recupero i dati del blob dall'evento e lo leggo
7      event_data = event.get_json()
8      blob_url = event_data.get("url")
9      if not blob_url:
10         logging.warning("Nessun URL blob trovato nell'evento")
11         return
12     logging.info(f"Blob URL: {blob_url}")
13
14     path_parts = blob_url.split("/")
15     container_name = path_parts[-2]
16     blob_name = path_parts[-1]
17
18     connect_str = os.environ["AzureWebJobsStorage"]
19     bsc = get_blob_service(connect_str)
20     container_client = bsc.get_container_client(container_name)
21     blob_client = container_client.get_blob_client(blob_name)
22
23     data_bytes = blob_client.download_blob().readall()
24     data_str = data_bytes.decode('utf-8')
25
26     # Carico i dati in un DataFrame pandas aggiungendo l'header alle colonne
27     columns = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'class']
28     df = pd.read_csv(io.StringIO(data_str), names=columns)
29     logging.info(f"Dataset originale: {df.shape[0]} righe, {df.shape[1]} colonne")
30
31     # Rimozione delle righe con valori nulli, se ci sono
32     df = df.dropna()
33     logging.info(f"Dopo rimozione valori nulli: {df.shape[0]} righe")
34
35     # Separazione delle features dal target
36     X = df.drop('class', axis=1)
37     y = df['class']
38
39     # Gestione delle feature
40     categorical_cols = X.columns.tolist()
41     preprocessor = ColumnTransformer(transformers=[('cat', OneHotEncoder(handle_unknown='ignore'), categorical_cols)])
42     X_transformed = preprocessor.fit_transform(X)
43
44     # Riassegno i nomi delle colonne
45     ohe_cols = preprocessor.get_feature_names_out()
46     df_transformed = pd.DataFrame(X_transformed.toarray(), columns=ohe_cols)
47     df_transformed['class'] = y.values
48     logging.info(f"Dataset trasformato: {df_transformed.shape[0]} righe, {df_transformed.shape[1]} colonne")
49
50     # Salvataggio dataset pronto in un nuovo blob
51     output_csv = df_transformed.to_csv(index=False, encode="utf-8")
52     clean_container_client = bsc.get_container_client(CLEAN_DATA_CONTAINER)
53     clean_container_client.upload_blob(name=NAME_PROCESSED_DATASET, data=output_csv, overwrite=True)
54     logging.info(f"Blob con il dataset pulito salvato in {CLEAN_DATA_CONTAINER}/{NAME_PROCESSED_DATASET}")
55
56     # Salvataggio anche del preprocessor
57     model_bytes = io.BytesIO()
58     joblib.dump(preprocessor, model_bytes)
59     model_bytes.seek(0)
60     model_container_client = bsc.get_container_client(MODEL_CONTAINER)
61     model_container_client.upload_blob(name=PREPROCESSOR_NAME, data=model_bytes, overwrite=True)
62     logging.info(f"Blob con il preprocessing salvato in {MODEL_CONTAINER}/{PREPROCESSOR_NAME}")
63

```

Figura 3.4: Funzione di elaborazione del dataset grezzo per prepararlo all'addestramento

- **Output layer:** restituisce la classe predetta

- * **Funzione di attivazione:** tanh

Fa sì che ogni neurone della rete applichi una funzione non lineare, in questo caso la tangente iperbolica, e ciò rende la rete capace di catturare relazioni non lineari

- * **Ottimizzatore:** LBFGS

LBFGS (Limited-memory Broyden–Fletcher–Goldfarb–Shanno) è l'algoritmo che la rete usa per aggiornare i pesi durante l'addestramento. Esso è molto efficace su dataset piccoli/medi, poiché converge velocemente e in modo stabile

- * **Iterazioni massime:** 1000

Serve per evitare che il modello resti bloccato in un ciclo infinito di calcolo per trovare i pesi ottimali

- * **Random state:** 42

È un seme casuale usato per inizializzare i pesi e per eventuali operazioni randomiche

A seguito dell'addestramento, il modello addestrato viene serializzato, via `joblib` e salvato sul blob storage

- **Output:** Salvataggio del modello addestrato sotto forma di blob all'interno del container `model/`

Successivamente, tutte le funzioni sono state registrate all'interno del file principale di `function_app.py`

3.3 Frontend

Il frontend, sviluppato con **Angular 16** con l'utilizzo di **bootstrap 5** per la grafica, fornisce un'interfaccia moderna e *responsive* per la realizzazione del servizio di inferenza in tempo reale.

Esso è strutturato in:

- **Model:** definisce le strutture dei dati utilizzati nel frontend

```

1  @trainingfunction.function_name(name="trainingFunctionEventGrid")
2  @trainingfunction.event_grid_trigger(arg_name="event", auth_level=func.AuthLevel.ANONYMOUS)
3  def trainingfunction(event: func.EventGridEvent):
4      logging.info(f"Evento ricevuto: {event.event_type}")
5
6      # Recupero i dati del blob dall'evento e lo leggo
7      event_data = event.get_json()
8      blob_url = event_data.get("url")
9      if not blob_url:
10         logging.warning("Nessun URL blob trovato nell'evento")
11         return
12     logging.info(f"Blob URL: {blob_url}")
13
14     path_parts = blob_url.split("/")
15     container_name = path_parts[-2]
16     blob_name = path_parts[-1]
17
18     connect_str = os.environ["AzureWebJobsStorage"]
19     bsc = get_blob_service(connect_str)
20     container_client = bsc.get_container_client(container_name)
21     blob_client = container_client.get_blob_client(blob_name)
22
23     # Scarico il dataset dal blob e lo inserisco in un DataFrame
24     data_bytes = blob_client.download_blob().readall()
25     data_str = data_bytes.decode('utf-8')
26     df = pd.read_csv(io.StringIO(data_str))
27     logging.info(f"Dataset caricato: {df.shape[0]} righe, {df.shape[1]} colonne")
28
29     # Separazione delle features dal target
30     X = df.drop('class', axis=1)
31     y = df['class']
32
33     # Creazione e training del modello con Rete Neurale
34     # Pipeline: preprocessing → scaling → rete neurale
35     categorical_cols = X.columns.tolist()
36     preprocessor = ColumnTransformer(
37         transformers=[('cat', OneHotEncoder(handle_unknown='ignore'), categorical_cols)]
38     )
39
40     model = Pipeline(steps=[
41         ('preprocessor', preprocessor),
42         ('scaler', StandardScaler(with_mean=False)),
43         ('model', MLPClassifier(
44             hidden_layer_sizes=(20,10),
45             activation="tanh",
46             solver="lbfgs",
47             max_iter=1000,
48             random_state=42
49         ))
50     ])
51
52     model.fit(X, y)
53     logging.info("Training completato con successo.")
54
55     # Serializzazione modello
56     model_bytes = io.BytesIO()
57     joblib.dump(model, model_bytes)
58     model_bytes.seek(0)
59
60     # Salvataggio del modello nel container
61     model_container_client = bsc.get_container_client(MODEL_CONTAINER)
62     model_container_client.upload_blob(name=MODEL_NAME, data=model_bytes, overwrite=True)
63     logging.info(f"Modello salvato in {MODEL_CONTAINER}/{MODEL_NAME}")
64

```

Figura 3.5: Funzione per eseguire il training del modello

- **Service:** definisce i servizi di comunicazione con il backend attraverso chiamate HTTP
- **Component:** definisce e struttura le componenti visive del frontend

3.3.1 Struttura

La struttura è stata gestita attraverso il **routing** fornito da Angular, dove troviamo:

- **/car-evaluation:** è la pagina dove gli utenti possono valutare auto e ottenere predizioni;
- **/dataset:** è la pagina dedicata alla gestione del dataset, protetta da **AuthGuard** ed accessibile solo ad utenti con ruolo *Admin*;
- **/error:** pagina di errore o accesso negato, utilizzata per il redirect in caso di mancanza di autorizzazioni.

Inoltre, poiché a seguito di alcune *policy* legate all'account student di Azure non è stato possibile utilizzare *Microsoft Entra ID* per gestire l'autenticazione, il frontend utilizza per l'autenticazione il servizio esterno **Auth0** per realizzare un login semplice ed intuitivo. Necessario a sbloccare le funzionalità aggiuntive dell'applicazione ed a differenziare i normali utenti dagli utenti amministratori. (Lo stesso problema si è avuto nel deploy del frontend, poiché per policy non è stato possibile utilizzare *Azure Static Web Apps*, ma ho dovuto hostare il frontend su una macchina virtuale)

3.3.2 Componenti

Per semplicità le immagini relative alla visualizzazione del frontend sono inserite successivamente nella sezione di "simulazione d'esecuzione"

- **CarEvaluationPageComponent**

Essa mostra lo stato del modello, contiene il **CarFormComponent** per la compilazione del form ed il **PredictionResultComponent** per la visualizzazione del risultato.

Inoltre, a seguito di autenticazione permette di salvare le predizioni effettuate e mostra la lista delle predizioni salvate dell'utente attraverso il **ListPredictionsComponent**.

- **CarFormComponent**

Definisce il form per effettuare la predizione, quindi, fornisce selezioni tramite menù a tendina per selezionare i valori di: `buying`, `maint`, `doors`, `persons`, `lug_boot`, `safety`. Compilato il form permette di richiamare il servizio di predizione a seguito del click del bottone.

- **PredictionResultComponent**

Permette di mostrare il risultato della predizione, mostrandone il corrispettivo valore con un colore diverso a seconda della classe di valutazione (`vgood`, `good`, `acc`, `unacc`). Inoltre, se l'utente è loggato mostra anche un bottone per effettuare il salvataggio della predizione.

- **ListPredictionsComponent**

Mostra le predizioni salvate di un utente loggato, tramite paginazione e possibilità di eseguire filtri sulla base della *classe*. Dunque, vengono mostrati i dati riassuntivi di ogni predizione sotto forma di *card*, all'interno della quale si trova anche un bottone che permette di eliminare la predizione, a seguito di conferma dell'utente

- **DatasetManagementPageComponent (Admin)**

La pagina integra l'`UploadAreaComponent` per consentire l'upload del file di dataset ed in più, mostra i metadati del dataset

3.3.3 Servizi

Troviamo 2 servizi principali:

- **PredictionService**: per richiamare le API relative alle predizioni
- **DatasetService**: per richiamare le API relative al dataset

3.3.4 Hosting del frontend

Come detto in precedenza, per effettuare l'hosting del frontend si è utilizzata una Azure Virtual Machine, con dimensione **Standard B1s (1 cpu virtuale, 1 GiB memoria)** ed usando come sistema operativo Ubuntu 22.04 LTS server.

La macchina virtuale è stata configurata usando **Nginx** come server web per la fruizione delle pagine web e, in più, grazie all'utilizzo di **Certbot + Let's encrypt** si sono abilitate le comunicazioni HTTPS.

Dunque, attraverso il comando `npm run build --configuration production` si è *buildato* il frontend, pronto per essere caricato nella cartella `/var/www/app/`, attraverso il comando `scp -r dist/frontend/* azureuser@74.161.160.201:/var/www/app/`

Configurato appositamente il web server *nginx* l'applicazione è stata resa fruibile da chiunque attraverso internet al dominio `gallicchiovittorioproject.switzerlandnorth.cloudapp.azure.com`

3.4 Flusso di esecuzione

- **Fase 1**

L'utente amministratore, *previa login*, accede alla sezione di gestione del dataset e carica il dataset *car.data* dall'interfaccia web (oppure ciò può essere fatto direttamente dal portale di Azure). Il caricamento del dataset fa sì che si richiami l'API `/upload_dataset` attivando la corrispondente funzione, la quale si occupa di caricare il dataset come blob nell'apposito container dello storage. A questo punto, Event Grid rileva l'evento di caricamento e invoca la funzione associata di *pre-processing* del dataset.

- **Fase 2**

Attivata la funzione di *pre-processing*, essa legge dal container nel blob storage il dataset grezzo ed applica al dataset le operazioni di: pulizia, trasformazione dei valori, normalizzazione delle feature e gestione delle variabili categoriche. Fatto ciò, la funzione salva il dataset pulito nell'apposito container dello storage e, salva anche il *preprocessor* delle variabili categoriche nel container del modello, necessario successivamente per effettuare le predizioni.

- **Fase 3**

Il caricamento del dataset pulito e trasformato nel blob storage attiva la funzione di *training*. Essa, in primis, addestra il modello di ML e, successivamente, terminato l'addestramento serializza il modello, il quale viene salvato come blob all'interno

dell'apposito container del blob storage. Ciò permette di poter accedere al modello in qualsiasi momento ed utilizzarlo per effettuare le predizioni in tempo reale.

- **Fase 4**

Il *servizio di inferenza*, fornito tramite interfaccia web, permette all'utente di compilare il *form* per ottenere il risultato della predizione in tempo reale. Ciò che accade al momento di effettuare la predizione è che il frontend richiama la relativa API del backend, la quale attiva la funzione di predizione; quindi, vengono letti i parametri dalla richiesta e vengono recuperati e de-serializzati il *preprocessor* ed il *modello* dal container nel blob storage e, viene effettuata la predizione sul modello. Il risultato è poi restituito inviandolo nella risposta, così da essere visualizzato nel frontend.

In aggiunta al servizio di inferenza, sono rese disponibili dal frontend le funzionalità di *gestione delle predizioni*. In particolare, si sono rese disponibili, *previa login* da parte di un normale utente, le funzionalità descritte precedentemente di: salvataggio delle proprie predizioni; visualizzazione delle predizioni, con possibilità di applicare dei filtri sulla *classe della predizione* ed, infine, eliminazione di predizioni precedentemente salvate.

3.5 Simulazione esecuzione

1. Inizio della pipeline da parte dell'utente amministratore con il caricamento del dataset

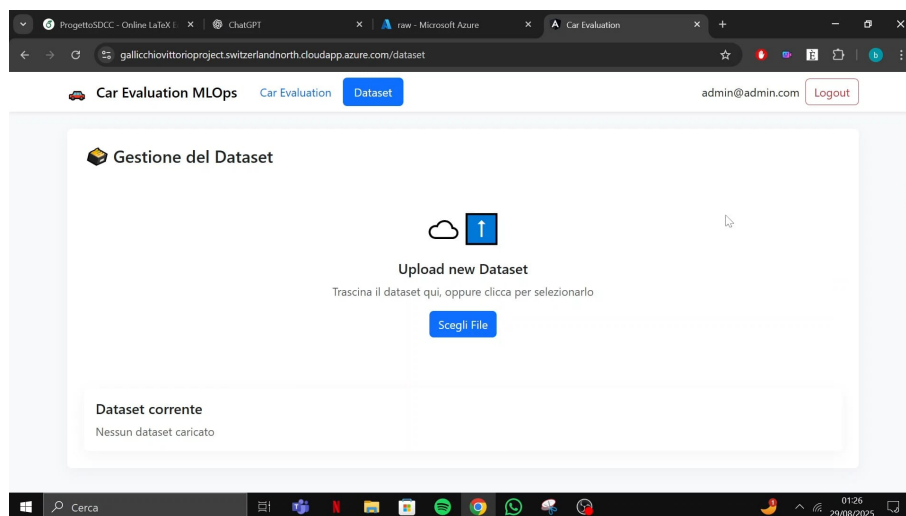


Figura 3.6: Pagina visibile solo dall'utente amministratore per la gestione del dataset

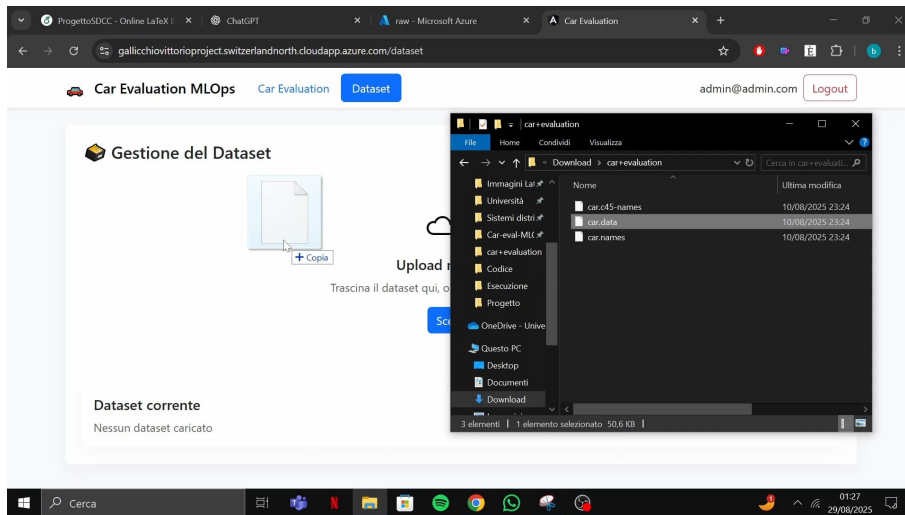


Figura 3.7: Upload del dataset

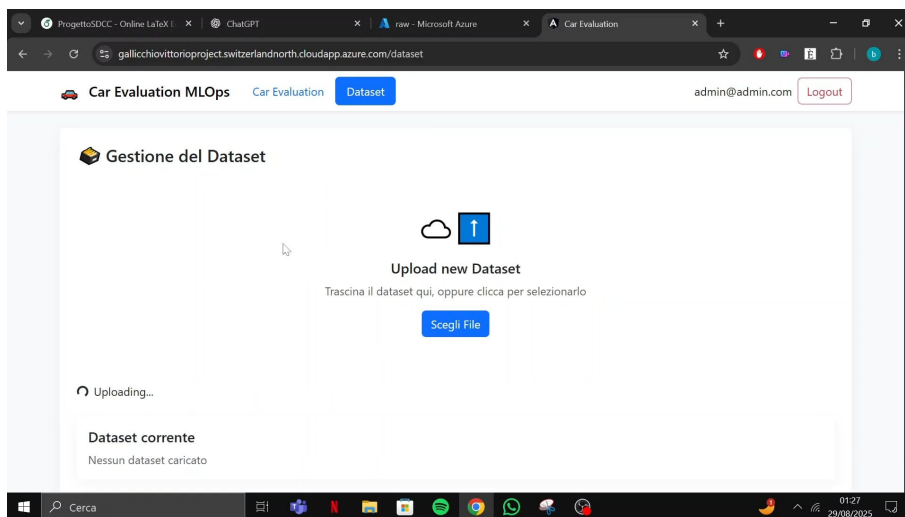


Figura 3.8

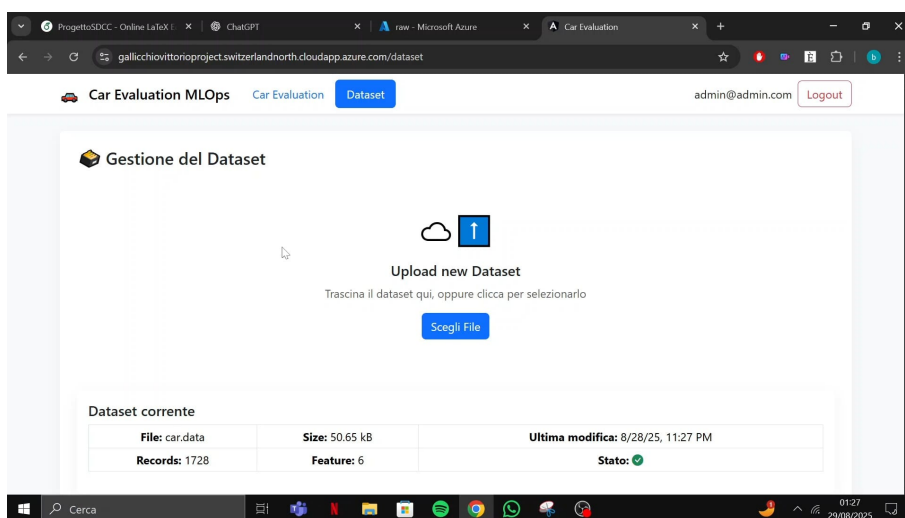


Figura 3.9: Corretto caricamento del dataset

2. Servizio di inferenza per un utente non loggato

Car Evaluation MLOps Car Evaluation Login

Modello:

"Car Evaluation" Form

Prezzo d'acquisto: Medio

Costi di manutenzione: Medio

Numero di porte: 5 o più

Capacità del numero di persone: 4

Grandezza del bagagliaio: Medio

Valutazione sulla sicurezza: Bassa

Effettua "Car Evaluation"

Risultato della predizione

Compila il form e clicca "Effettua Car Evaluation" per vedere il risultato della predizione

Figura 3.10: Pagina per effettuare la predizioni, dove l'utente compila il form e preme il bottone

Car Evaluation MLOps Car Evaluation Login

Modello:

"Car Evaluation" Form

Prezzo d'acquisto: Medio

Costi di manutenzione: Medio

Numero di porte: 5 o più

Capacità del numero di persone: 4

Grandezza del bagagliaio: Medio

Valutazione sulla sicurezza: Bassa

Effettua "Car Evaluation"

Risultato della predizione

INACCETTABILE

Figura 3.11: Visualizzazione della predizione

Car Evaluation MLOps Car Evaluation Login

Modello:

"Car Evaluation" Form

Prezzo d'acquisto: Medio

Costi di manutenzione: Medio

Numero di porte: 5 o più

Capacità del numero di persone: 4

Grandezza del bagagliaio: Medio

Valutazione sulla sicurezza: Alta

Effettua "Car Evaluation"

Risultato della predizione

VERAMENTE BUONO

Figura 3.12: Visualizzazione di una seconda predizione

3. Servizio di inferenza per un utente loggato

The screenshot shows the 'Car Evaluation MLOps' web application. At the top, there's a navigation bar with 'Car Evaluation' and a user profile 'utente@utente.com' with a 'Logout' button. The main content area is divided into two sections. On the left, the 'Modello' section contains the 'Car Evaluation Form' with fields for 'Prezzo d'acquisto' (Alto), 'Costi di manutenzione' (Molto alto), 'Numero di porte' (3), 'Capacità del numero di persone' (4), 'Grandezza del bagagliaio' (Piccolo), and 'Valutazione sulla sicurezza' (Alta). A blue button 'Effettua "Car Evaluation"' is at the bottom of the form. On the right, the 'Risultato della predizione' section shows a grey box with the text 'INACCETTABILE' and a blue button 'Salva predizione'. Below these sections, there's a 'Filtri' dropdown set to 'Tutte' and three colored boxes: 'Accettabile' (yellow), 'Inaccettabile' (red), and 'Veramente buono' (green). Each box contains a list of features and their values, such as 'Prezzo: V. Alto', 'Manti: Basso', 'Porte: 5 o più', 'Persone: Più', 'Bag: Medio' for the 'Accettabile' category.

Figura 3.13: Esecuzione di una predizione con possibilità di salvataggio

The screenshot shows the 'Car Evaluation' web application with the 'Risultato della predizione' section at the top. Below it, there's a 'Filtri' dropdown set to 'Tutte' and six colored boxes arranged in two rows. Each box represents a prediction category: 'Accettabile' (yellow), 'Inaccettabile' (red), 'Veramente buono' (green), 'Buono' (blue), 'Accettabile' (yellow), and 'Inaccettabile' (red). Each box contains a list of features and their values, such as 'Prezzo: V. Alto', 'Manti: Basso', 'Porte: 5 o più', 'Persone: Più', 'Bag: Medio' for the top-left 'Accettabile' category. Each box has a red 'Elimina' button at the bottom. At the bottom of the page, there's a pagination bar with 'Prev', '1', '2', and 'Success' buttons.

Figura 3.14: Visualizzazione della lista di predizioni salvate con possibilità di eliminazione

Capitolo 4

Conclusioni

Il progetto sviluppato mi ha permesso di realizzare una pipeline per il Machine Learning in ambiente cloud, sfruttando i paradigmi serverless e le funzionalità offerte da Microsoft Azure. La pipeline, integra in maniera ottimale servizi serverless, macchine virtuali, gestione dei dati attraverso blob e database e componenti di Machine Learning, fornendo una soluzione completamente automatizzata, scalabile e facilmente manutenibile.

L'utilizzo di Azure ha permesso di sfruttare un'infrastruttura affidabile, e di ridurre i costi operativi grazie al modello a consumo per le funzioni serverless, mentre la Virtual Machine appositamente configurata ha permesso di fornire un'interfaccia web interattiva user-friendly per l'inferenza in tempo reale.

Grazie all'utilizzo di un'architettura FaaS (Function as a Service), il sistema garantisce scalabilità, flessibilità e riduzione dei costi, poiché le risorse vengono allocate solo quando effettivamente necessarie. Inoltre, l'approccio adottato rispetta i principi dell'MLOps, favorendo l'automazione, la riproducibilità e dei processi di Machine Learning.

4.1 Risultati raggiunti

Tutti gli obiettivi richiesti dalla traccia di progetto sono stati raggiunti, come dimostrato dalle sezioni di implementazione e simulazione. In particolare, il sistema realizzato risponde pienamente ai requisiti funzionali e non funzionali, integrando in modo coerente i servizi offerti da Azure.

Oltre agli obiettivi minimi richiesti, sono state introdotte alcune funzionalità aggiuntive che arricchiscono il progetto e lo rendono più vicino a un'applicazione reale:

- Gestione delle predizioni: possibilità di salvare, filtrare e cancellare predizioni associate al singolo utente;
- Sistema di autenticazione basato su Auth0 con differenziazione tra utenti normali e amministratori con funzionalità dedicate;
- Gestione sicura dei dati: database accessibile solo da servizi autorizzati e comunicazioni HTTPS abilitate.

4.2 Considerazioni personali

Lo sviluppo di questo progetto ha rappresentato per me un'importante occasione di crescita, sia dal punto di vista tecnico che metodologico. La realizzazione di un progetto di MLOps in ambiente cloud, attraverso l'uso di servizi serverless, di storage e di macchine virtuali, ha infatti richiesto di affrontare diverse sfide pratiche legate allo studio degli ambienti, all'integrazione e all'utilizzo delle SDK. Un ulteriore aspetto rilevante è stata la necessità di bilanciare le risorse disponibili con i costi dei servizi cloud, imparando a valutare con attenzione le soluzioni più efficienti dal punto di vista economico e prestazionale.

Dal punto di vista delle competenze acquisite, il progetto mi ha permesso di acquisire e/o consolidare conoscenze relative a:

- Uso degli strumenti cloud per la gestione di servizi di: Storage, Funzioni e Macchine Virtuali.
- Principi dell'architettura serverless e dei sistemi distribuiti;
- Tecniche di pre-processing e addestramento di modelli di machine learning;
- Pratiche di MLOps, per l'automazione dei flussi e alla riproducibilità dei processi.

In conclusione, questo progetto mi ha permesso non solo di mettere in pratica le nozioni teoriche affrontate durante il corso, ma anche di acquisire un metodo di lavoro orientato alla scalabilità, alla modularità e alla manutenibilità delle soluzioni. Il risultato ottenuto dimostra come un approccio moderno basato sul cloud e sull'automazione possa semplificare lo sviluppo di sistemi complessi e gettare le basi per applicazioni reali in ambito industriale.