

[REPORT] Davide Piu - Lab Big Data

Obiettivi e contesto

L'obiettivo del progetto era quello di creare un'infrastruttura completa che, tramite l'utilizzo di un set di tecnologie mi permettesse di mandare in produzione un modello di Machine Learning, pur consapevole che difficilmente nella messa in produzione di un modello di ML sia una sola persona ad occuparsi delle varie fasi del processo e che probabilmente la soluzione trovata non fa uso delle migliori best practice. Il mio obiettivo di fondo era imparare ad avvicinarmi ad un problema del genere e iniziare ad approfondire alcune tecnologie utilizzate in questi casi. Per la realizzazione del progetto ho utilizzato un dataset di recensioni dal sito [Kaggle.com](https://www.kaggle.com/ryati131457/web-data-amazon-movie-reviews-processed). Il dataset contiene informazioni relative a recensioni di film presenti nella piattaforma Amazon Prime Video. Le recensioni dovranno essere classificate in positive o negative. Ho deciso di utilizzare Pyspark prevalentemente come strumento per il preprocessing dei dati. Ipotizzando un'implementazione reale con in input dataset di grandi dimensioni (o grandi flussi di dati in tempo reale) si potrebbe sfruttare la potenza di calcolo di un cluster che sfrutta gli RDD (e i dataframe) e le varie funzionalità messe a disposizione da Spark. Il risultato delle trasformazioni sui dati verrebbe utilizzato in un'altra fase per realizzare applicazioni e modelli di ML. Il mio progetto ha seguito questa logica.

*Nel codice in allegato è comunque presente un Modello di albero decisionale utilizzato all'interno dell'ambiente di Pyspark

Link dataset: <https://www.kaggle.com/ryati131457/web-data-amazon-movie-reviews-processed>

Fasi del progetto

Il progetto è stato implementato in tre fasi principali:

1. Pre-processing dei dati all'interno di Pyspark (tramite cluster fornito da Databricks CE), l'output finale è un file csv;
2. Realizzazione di una web app con il framework Flask;
3. Creato un container Docker contenente l'applicazione.

1. Prima Fase: Preprocessing dei dati all'interno di Pyspark

Contesto

- In questa fase sono state eseguite delle manipolazioni dei dati iniziali. Ho effettuato una serie di trasformazioni e creato una pipeline, il risultato della pipeline è un dataframe, successivamente esportato tramite csv ed utilizzato all'interno di Flask;
- Il setup utilizzato in questa fase consisteva in un cluster creato tramite Databricks Community Edition;

- In questa fase ho utilizzato una delle strutture dati più usate all'interno di Pyspark, i dataframe. Hanno un comportamento simile ai dataframe di Pandas ma possono essere utilizzati per il calcolo distribuito. Sono costruiti infatti sulla base degli RDD, che sono delle collection distribuite in un'insieme di nodi all'interno di un cluster che possono essere eseguiti in parallelo con un recupero automatico degli errori.

PySpark

Il dataset aveva una struttura iniziale composta da undici colonne:

- **productId**: string --> codice identificativo del film;
- **userId**: string --> utente che ha lasciato una recensione;
- **profileName**: string --> nome utente;
- **helpfulness**: string --> utilità della recensione secondo gli altri utenti;
- **score**: string--> score della recensione lasciato dall'utente (va da 1.0 a 5.0);
- **time**: string ---> non c'era una descrizione della variabile ma molto probabilmente contiene informazioni riguardanti l'orario in cui è stata rilasciata la recensione o esprime l'ordine numerale delle recensioni;
- **text**: string --> testo della recensione;
- ****summary**: **string--> breve descrizione della recensione.

L'assunzione fatta è che in questo caso non esistano recensioni neutrali, proprio per questo motivo ho deciso di assegnare, creando una nuova colonna **"new_score"**, il valore **"negativa"** alle recensioni con uno score inferiore a 3.0, a tutte le recensioni con uno score maggiore di 3.0 è stata assegnata l'etichetta **"positiva"**.

Il **dataframe** come si presenta il dataframe in questa fase:

productId	userId	profileName	helpfulness	score	time	summary	text	text_length	new_score
B003AI2VGA	A141HP4LYPWSR	Brian E. Erland '...	7/7	3.0	1182729600	'There Is So Much...	[Synopsis: On the ...	1842	positive
B003AI2VGA	A328S9RN3U5M68	Grady Harp	4/4	3.0	1181952000	Worthwhile and Im...	THE VIRGIN OF JUA...	2556	positive
B003AI2VGA	A1I7QGUDP043DG	Chrissy K. McVay ...	8/10	5.0	1164844800	This movie needed...	The scenes in thi...	854	positive
B003AI2VGA	A1M5405JH9THP9	golgotha.gov	1/1	3.0	1197158400	distantly based o...	THE VIRGIN OF JUA...	2856	positive
B003AI2VGA	ATXL536YX71TR	KerrLines '"M...	1/1	3.0	1188345600	'What's going on ...	Informationally, ...	762	positive
B003AI2VGA	A3QYDL5CDNYN66	abra 'a devoted r...	0/0	2.0	1229040000	Pretty pointless ...	The murders in Ju...	190	negative
B003AI2VGA	AQJVNDW6YZFQS	Charles R. Williams	3/11	1.0	1164153600	This is junk, sta...	Mexican men are m...	312	negative
B00006HAXW	AD4CDZK7D31XP	Anthony Accordino	64/65	5.0	1060473600	A Rock N Roll Hi...	Over the past few...	1709	positive
B00006HAXW	A3Q4S5DFVPB70D	Joseph P. Aiello	26/26	5.0	1041292800	A MUST-HAVE vid...	I recvd this vide...	1072	positive
B00006HAXW	A2P7UB02HAVEPB	'bruce_from_la'	24/24	5.0	1061164800	If You Like DooWo...	Wow! When I saw t...	725	positive
B00006HAXW	A2TX99AZKDK0V7	Henrique Peirano	22/23	4.0	1039564800	I expected more...	I have the Doo Wo...	1447	positive
B00006HAXW	AFC8IKR407HSK	Richard Albero	14/14	5.0	1045526400	Professional Exce...	Having worked in ...	675	positive
B00006HAXW	A1FRPGQYQTAOR1	Les	9/9	5.0	1062979200	Marvelous, just M...	The people who ha...	309	positive
B00006HAXW	A1RSDE90N6RSZF	Joseph M. Kotow	9/9	5.0	1042502400	Pittsburgh - Home...	I have all of the...	259	positive
B00006HAXW	A1OUB0GB5970AO	'fellafromnyc'	7/7	4.0	1049846400	They sang in the ...	The performance o...	187	positive
B00006HAXW	A3NPHQVIY59Y0Y	S. Dorman	7/7	5.0	1047945600	DOO WOP RECORDED ...	[Get it, also get ...	144	positive
B00006HAXW	AFKMBAY28XOSA	RFP	7/7	5.0	1038787200	ROCK RYTHM AND DO...	Excellent, excell...	174	positive
B00006HAXW	A66KMXH9V70GU	C. Thomas	4/4	5.0	1177804800	Unbelievable Best...	This video is awe...	524	positive

Nel problema in questione per classificare una recensione è necessario farlo sulla base del valore di score, *score* è stato però trasformato in una nuova variabile **"new_score"** che può avere solo due valori. È necessario però che questa nuova colonna venga indicizzata, alcuni algoritmi infatti non possono lavorare direttamente con dati categorici e sono necessarie delle trasformazioni, richiedono infatti che sia le variabili di input che quelle di output siano numeriche. Devo quindi convertire la colonna da stringa in un formato numerico. Per questo motivo ho deciso di utilizzare lo StringIndexer sulla colonna *new_score*, in modo da avere il valore 0 in caso di recensioni "positive" e il valore 1 in caso di

recensioni "negative". L'output di questa trasformazione è la colonna **score_indexed**.

```
# indicizzo la colonna dello score dei film
from pyspark.ml.feature import StringIndexer
indexer = StringIndexer(inputCol="new_score", outputCol="score_indexed")
df = indexer.fit(df2).transform(df2)
```

Per effettuare la classificazione ho scelto solo due colonne: **text** e **score_indexed**. Ogni riga della colonna text contiene una recensione. Ovviamente si tratta solamente di una lunga stringa di caratteri a cui è difficile assegnare un valore numerico. In questa fase ci viene in aiuto il Natural Language Processing. Il NLP ci permette infatti di risolvere l'ambiguità del linguaggio scritto e aggiunge una struttura numerica alle informazioni contenute nei testi. Spark tramite la libreria MLlib mette a disposizione una serie di metodi per il NLP. Ho effettuato una serie di trasformazioni sulla colonna **text**. In sequenza ho:

1. Rimosso dei pattern di caratteri tramite regex (regex_replace);
2. Convertito le stringhe di testo in liste di stringhe tramite Tokenizer
3. Rimosso le **Stop Words** dai tokens

Le fasi 2 e 3 sono state implementate tramite un'altro metodo molto utilizzato messo a disposizione da Pyspark: le **Pipeline**. Tramite esse è possibile passare un oggetto a cui verranno effettuate una serie di trasformazioni in sequenza e l'ordine delle operazioni può essere definito tramite il parametro **stages=[]**.

```
# processing del dataframe che andrò ad utilizzare in flask
tokenizer = Tokenizer(inputCol="new_text", outputCol="token_text")
stopremove =
StopWordsRemover(inputCol='token_text',outputCol='stop_tokens')
data_pipe = Pipeline(stages=[tokenizer,stopremove])
model = data_pipe.fit(df)
df = model.transform(df)
```

Ecco come si mostrava il dataset una volta applicata la Pipeline

```

1 # Display mostra delle statistiche e le informazioni principali del dataset
2 display(movie_reviews_df)

```

► (2) Spark Jobs

	productid ▲	stop_tokens ▲	score_indexed ▲
1	B003AI2VGA	► ["synopsis", "daily", "trek", "juarez", "mexico", "el", "paso", "texas", "ever", "increasing", "number", "female", "workers", "found", "raped", "murdered", "surrounding", "desert", "investigative", "reporter", "karina", "danes", "minnie", "driver", "arrives", "los", "angeles", "pursue", "story", "angers", "local", "police", "factory", "owners", "employee", "undocumented", "aliens", "pointed", "questions", "relentless", "quest", "truth", "br", "br", "story", "goes", "nationwide", "young", "girl", "named", "mariela", "ana", "claudia", "talancon", "survives", "vicious", "attack", "walks", "desert", "crediting", "blessed", "virgin", "rescue", "story", "enhanced", "wounds", "christ", "stigmata", "appear", "palms", "also", "claims", "received", "message", "hope", "virgin", "mary", "soon", "fanatical", "movement", "forms", "around", "fight", "evil", "holds", "stranglehold", "area", "br", "br", "critique", "possessing", "lifelong", "fascination", "esoteric", "matters", "catholic", "mysticism", "miracles", "mysterious", "appearance", "stigmata", "immediately", "attracted", "05", "dvd", "release", "virgin", "juarez", "film", "offers", "rather", "unique", "storyline", "blending", "current", "socio", "political", "concerns", "constant", "flow", "mexican", "migrant", "workers", "back", "forth", "across", "u", "mexican", "border", "traditional", "catholic", "beliefs", "hispanic", "population", "must", "say", "quite", "surprised", "unexpected", "route", "taken", "plot", "means", "methods", "heavenly", "message", "unfolds", "br", "br", "virgin", "juarez", "film", "care", "watch", "interesting", "enough", "merit", "least", "one", "viewing", "minnie", "driver", "delivers", "solid", "performance", "ana", "claudia", "talancon", "perfect", "fragile", "innocent", "visionary", "mariela", "also", "starring", "esai", "morales", "angus", "macfadyen", "braveheart"]	0
	B003AI2VGA	► ["virgin", "juarez", "based", "true", "events", "surrounding", "crime", "problems", "juarez", "mexico", "reflected", "gringo", "exploitation", "businesses", "neighboring", "el", "paso", "texas", "story", "contains", "many", "important", "facts", "desperately", "need", "brought", "light", "impact", "film", "falters", "choices", "made", "writer", "director", "br", "br", "karina", "danes", "minnie", "driver", "journalist", "los", "angeles", "newspaper", "flown", "juarez", "investigate", "multiple", "hundreds", "killings", "young", "women", "targets", "murders", "seem", "young", "women", "working", "us", "sponsored", "sweatshops", "juarez", "picked", "night", "work", "raped", "beaten", "killed", "danes", "convinced", "juarez", "police", "force", "nothing", "takes", "mission", "exposing",	0

Showing the first 1000 rows.



Questo è il dataset finale che ho utilizzato per fare il training del modello che ho utilizzato per la realizzazione all'interno della web app realizzata con Flask. Ovviamente consapevole della possibilità di applicare migliorie, tra le quali: utilizzare le colonne *summary*, *time* e **text_lenght*. *Il metodo `display()` mette a disposizione un bottom che permette di scaricare il dataset in formato `csv` e fornisce, inoltre, una serie di grafici sulle variabili presenti nel dataset.

2. Realizzazione di una web app con il framework Flask e Scikit-learn

L'obiettivo in questa fase era quello di creare un sistema che utilizzasse il Machine Learning per classificare le recensioni. Quindi il workflow in questa fase è il seguente:

1. Viene fatto il training offline di un classificatore utilizzando recensioni positive e recensioni negative;
2. Il modello **trained* *viene messo a disposizione agli utenti (*as a service*);
3. Utenti possono fare predizioni.

Costruzione del modello

Il dataset risultante dal pre-processing realizzato all'interno di Pyspark contiene le colonne:

- **stop_tokens**: ogni campo contiene una lista di token a cui sono già state rimosse le stop words
- **score_indexed**: è l'etichetta, 0: recensione positiva, 1: recensione negativa;
- **prouctionId**: codice identificativo del film, non è stata utilizzata in questa fase.

In questa fase ho utilizzato la libreria Scikit-learn e il classificatore utilizzato è il Naive Bayes Classifier. È un modello che viene utilizzato per dati di grandi dimensioni e ha dimostrato di avere un discreto successo in applicazioni come le diagnosi mediche e la classificazione di documenti testuali. Con Naive Bayes non si fa riferimento ad un classificatore specifico ma a tutta una famiglia di algoritmi. Tutti questi

algoritmi sono basati sul principio che un valore di un attributo è indipendente dal valore di qualsiasi altro attributo. Questi classificatori etichettano i dati in classi distinte (sentiment positivo o negativo). Per capirne il funzionamento è necessario accennare il teorema di Bayes:

$$P(C|D)$$

=

$$\frac{P(D|C)P(C)}{P(D)}$$

Dove:

$$P(C)$$

è la probabilità a priori

$$P(C|D)$$

è quello che si vuole trovare, la probabilità a posteriori.

$$P(D)$$

è la probabilità che un dato possa appartenere ad una qualsiasi ipotesi

In questo caso D è la recensione, con

$$D$$

=

$$(d_1, d_2, \dots, d_n)$$

dove

$$d_i$$

è l'

$$i$$

-esimo attributo della lista di parole di cui è costituito il testo della recensione. Mentre

$$C$$

rappresenta la classificazione del testo, che può essere quindi nella classe

positiva

o

negativa

. Si cerca di trovare la probabilità che una recensione, date le parole al suo interno, appartenga alla classe negativa o alla classe positiva. Questo algoritmo viene chiamato *Naive* (ingenuo) perché si presuppone che il sentiment di una parola non sia influenzata dalle altre parole presenti nella recensione. Il classificatore bayesiano fa parte della famiglia dell'apprendimento *supervisionato* quindi, per addestrare il modello è necessario passare una serie di recensioni già classificate. Ed è proprio quello che è stato fatto e lo scopo del pre-processing all'interno di Spark era indirizzato a questo.

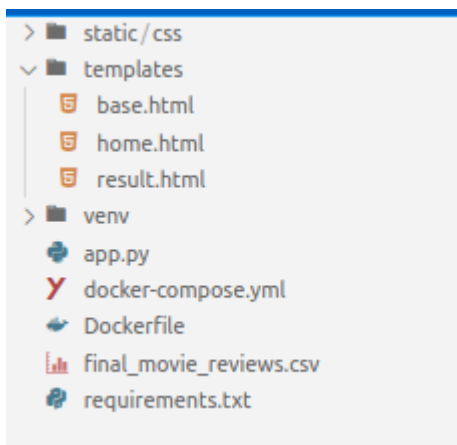
Ecco i risultati del modello:

	precision	recall	f1-score	support
0	0.86	0.85	0.86	303
1	0.88	0.89	0.88	357
accuracy			0.87	660
macro avg	0.87	0.87	0.87	660
weighted avg	0.87	0.87	0.87	660

Creazione dell'applicazione Web

Una volta preparato il modello per la classificazione delle recensioni ho sviluppato un'applicazione che consiste in una serie di semplici pagine web che permettono di inserire una recensione all'interno di un form che, una volta inviato il submit del form contenente la recensione, rimanderanno ad una pagina con il risultato o che mostrano direttamente il risultato nel caso dell'API realizzata con Flassger.

Questa è la struttura della directory della web app:



```

# inizializza flaskj
app = Flask(__name__)
Swagger(app)# crea l'api per l'app Flask

# importazione dataset
df = pd.read_csv("final_movie_reviews.csv")

# features and label
X, y = df.stop_tokens, df.score_indexed

# extract featuresf with count vectorizer
cv = CountVectorizer()

X = cv.fit_transform(X) # Fit the Data
X_train, X_test, y_train, y_test = train_test_split( # divisione training e test
    X, y, test_size=0.33, random_state=42)

clf = MultinomialNB()
clf.fit(X_train, y_train)
clf.score(X_test, y_test)

@app.route('/')
def home():
    return render_template('home.html')# renderizza template home, da qui è possibile inserire la recensione

#mostra risultato della predizione
@app.route('/predict', methods=['POST'])
def predict():
    if request.method == 'POST':
        message = request.form['message']
        data = [message]
        vect = cv.transform(data).toarray()
        my_prediction = clf.predict(vect)
        return render_template('result.html', prediction=my_prediction)

# flasgger api, espone i risultati della predizione tramite un'api
@app.route('/predict_api',methods=["Get"])# per accedere bisogna dare il l'url '\apidocs'
def predict_api():
    """Controlla se la recensione è positiva
    o negativa !
    ---
    parameters:
      - name: message
        in: query
        type: string
        required: true
    responses:
      200:
        description: The output values
    """
    message=request.args.get("message")# prende il messaggio del form
    data = [message] # inserisce in data
    vect = cv.transform(data).toarray() # vettorizza
    prediction=clf.predict(vect) # predizione
    print(prediction)
    return "Ciao, ecco la recensione ----> "+str(prediction)+"\n[0]--> positiva \n[1]-->negativa" # risultato

```

- `templates` contiene i file statici html che verranno renderizzati nel browser
- `app.py` contiene il codice che verrà eseguito dall'interprete Python per avviare l'applicazione e include il codice per classificare le recensioni
- Ho inizializzato l'istanza di Flask con l'argomento **name**
- I decorator `@app.route("")` specificano quale URL verrà utilizzato quando vengono eseguite le funzioni home, predict, ecc,
- il Metodo POST permette di trasportare i dati presenti nel form nel server inserendolo all'interno di **message**.
- la funzione **predict** prende il messaggio all'interno del form e renderizza la pagina 'result.html' con il risultato
- **predict_api** permette di inserire il messaggio ma restituisce il risultato della classificazione all'interno della stessa pagina ed espone il risultato tramite API. (Per questo passo è stato utilizzato

il tool Flasgger)

Risultato Finale:

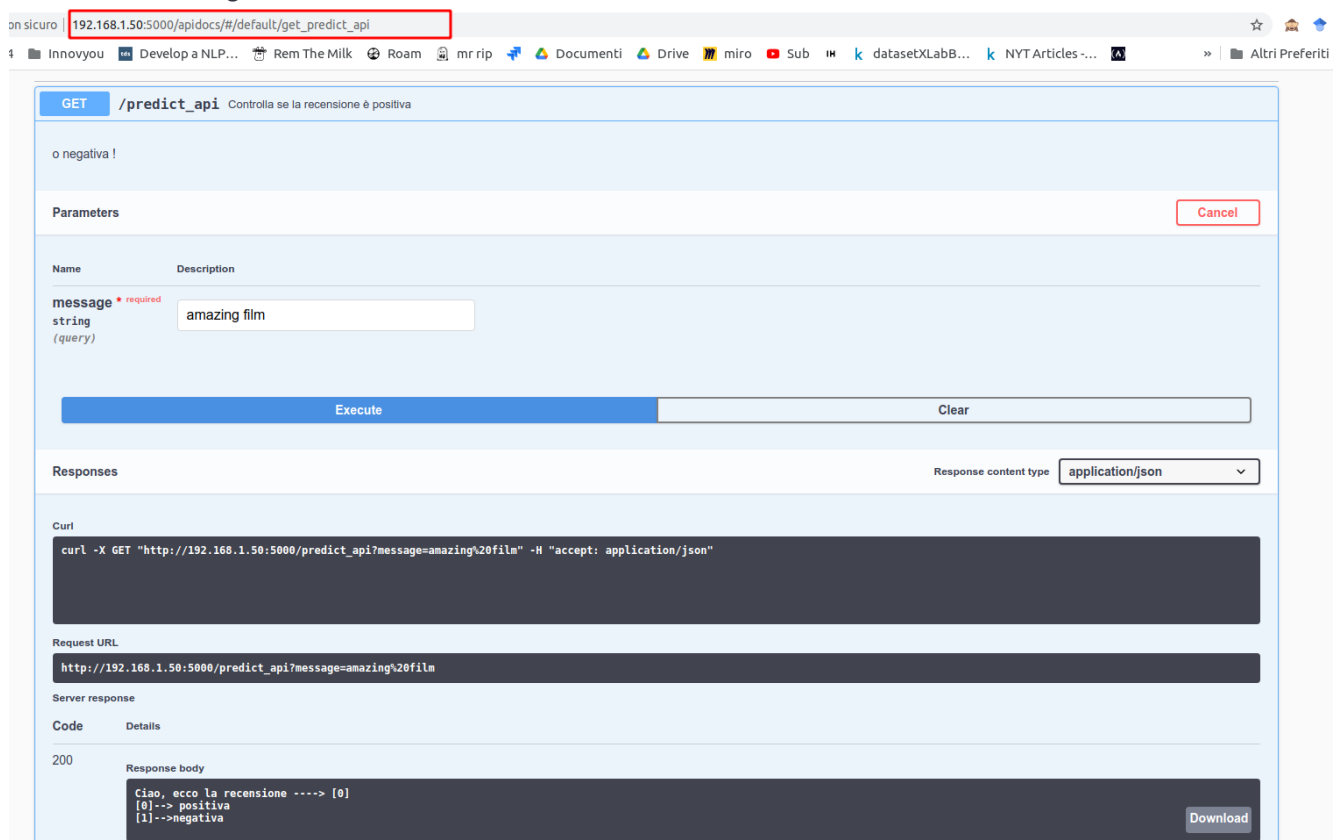
- Pagina con form inserimento recensione



- Pagina risultato Classificazione



- REST API Flassger



3. Docker

Docker si basa sul concetto di virtualizzazione, sopra l'OS creiamo dei container, ogni container avrà una root user folder. I container funzionano un po' come le macchine virtuali. Isolano una singola

applicazione (e le relative dipendenze) sia dal SO che dagli altri container. Tutti i container condividono un SO comune ma sono indipendenti l'uno dall'altro. I principali benefici di Docker sono:

- Migliora utilizzo delle risorse
- Velocizza il flusso del delivery del software
- Application portability

Docker e ML

Ho utilizzato docker perchè volevo provare ad utilizzare uno strumento che permette di incrementare la portabilità di un progetto e teoricamente incrementarne la scalabilità.

Ho prima creato un'immagine tramite un Dockerfile della mia applicazione e successivamente ho costruito il container utilizzando Docker-Compose. Strumento che permette di creare un network di container. Attualmente nel mio *docker-compose* è presente solo l'immagine della web app realizzata ma un possibile miglioramento potrebbe essere quello di inserire l'ambiente Spark all'interno dello stesso docker-compose e utilizzarlo in un cluster su Google Cloud o AWS.

Creazione *Dockerfile*:

```
FROM python:3.8
WORKDIR /app
COPY . .
RUN pip install -r requirements.txt
ENTRYPOINT [ "python" ]
CMD [ "app.py" ]
```

docker-compose.yml:

```
version: "3.7"
services:
  davidepiu_api:
    image: "davidapiubigdata:1.0"
    build:
      context: ./
    ports:
      - 5000:5000
```

Possibili miglioramenti

- Implementare l'algoritmo di Machine Learning all'interno dello stesso Cluster Pyspark e serializzare il modello utilizzando un formato adatto, il modello serializzato poi si potrà usare all'interno di un'applicazione
- Utilizzare la libreria Spark NLP che fornisce strumenti di deep learning per problemi del genere;
- Utilizzare più features e l'OnehotEncoder;

- Inserire sia Flask che Pyspark all'interno del docker-compose;
- Approfondire la parte relativa ai modelli e alle metriche di valutazione.