**ANSWERS**

**ANSWER 1**

Natural Language Processing (NLP) is an area of growing attention due to increasing number of applications like chatbots, machine translation etc. In some ways, the entire revolution of intelligent machines in based on the ability to understand and interact with humans.

TextBlob is built on the shoulders of NLTK and Pattern. A big advantage of this is, it is easy to learn and offers a lot of features like sentiment analysis, pos-tagging, noun phrase extraction, etc. On a side note, there is [spacy](https://www.analyticsvidhya.com/blog/2017/04/natural-language-processing-made-easy-using-spacy-%E2%80%8Bin-python/), which is widely recognized as one of the powerful and advanced library used to implement NLP tasks. But having encountered both spacy and TextBlob, TextBlob is suggested due to its simple interface.

We’ve all seen tweets with a plethora of spelling mistakes. Our timelines are often filled with hastily sent tweets that are barely legible at times. In that regard, spelling correction is a useful pre-processing step because this also will help us in reducing multiple copies of words. For example, “Analytics” and “analytcs” will be treated as different words even if they are used in the same sense.

To achieve this, we will use the textblob library.

Spelling correction is a cool feature which TextBlob offers, we can be accessed using the **correct** function as shown below.

We can also check the list of suggested word and its confidence using the **spellcheck** function.

Note that it will actually take a lot of time to make these corrections. Moreover, we cannot always expect it to be accurate so some care should be taken before applying it. We should also keep in mind that words are often used in their abbreviated form. For instance, ‘your’ is used as ‘ur’. We should treat this before the spelling correction step, otherwise these words might be transformed into any other word. We can use **replace** to replace these shortcut words with the actual word. Else we can remove these words by predefining them as stop words as it does not carry any information.

**ANSWER 3**

Introduction

Given a Hinglish text, for sentiment analysis generally the techniques applicable for the English text are used. Hence, we might lose out on the important sentiments that might be conveyed by the part written in Hindi. Thus, it is highly important to take into account the sentiment of both the languages.

for e.g. "That restraurant is not good. Itna ghatiya khaana to kabhi nahi khaya"

which means "That restaurant is not good. I haven't had such a bad food ever in my life"

The problem with these texts is that the Hindi written is in an informal manner, also it is not in the script in which the language is originally written. Hence different people might have different versions of spellings and the rule with which they write such texts. In the subsequent sections we have given a brief explanation on how these challenges were handled.

Methodology

* **Pre-processing** One of the most initial steps where tasks like removing hashtags, mentions and links in the tweet were completed. We also applied spelling normalisation and found out the stem words using the stemmer package.
* **Clustering** In this task we clustered out the Hindi and the English portions of the tweet. One of the main properties of such texts is that the English and the Hindi parts generally exist in groups. Hence, we first try to isolate them. We use the corpus generated from a dictionary. For e.g. if we have to classify the word 'reccommend', which has been wrongly spelt, and the actual spelling is 'recommend'. So we first of all consider this word and compute it's [Levenshtein distance](https://en.wikipedia.org/wiki/Levenshtein_distance) with words in our corpus starting from 'r' and having a length in range (l-2,l+2) where l is the length of the word we are considering. For the example we have considered the Levenshtein distance will be less. But for a word in Hindi like 'ghatiya', which means 'bad' or 'cheap' depending upon context, will have a large value of the Levenshtein distance with any word beginning with g in the dictionary. Hence, we a lot a distance to every word and then finally apply the k-means algorithm to get two clusters of Hindi and English. In certain cases, like the Hindi word 'main' means 'me'. But this is also an English word, however classifying this word as Hindi won't have any effect on our results since the words like these do not have any overall effect on the sentiment of the tweet. Most of the Hindi words which can affect the overall sentiment have a high levenshtein distance with a word of similar length in the English corpus.
* **Processing** Using the googletrans library which has been [licensed by MIT](https://github.com/vipul-khatana/Hinglish-Sentiment-Analysis/blob/master/LICENSE) we translate the Hindi written in Latin script into Hindi written in Devanagari script. Then we use the ESWN and the HSWN to interact with our text and assign senti scores to all the words. For emojis we have used the python regular expression for assigning score to the emojis.
* **Feature set** We then construct a feature set consisting of 7 features for every tweet:
  + Whether it has a positive score or not
  + Whether it has a negative score or not
  + Word count greater than 8
  + Contains adjectives
  + Contains emojis
  + Contains hashtag
  + Contains mentions
* **Classification** We use SVM classifier.

Installation

The libraries required are: numpy, pandas, xlrd, XlsxWriter, scikit-learn, regex, pyparsing, nltk, googletrans, sklearn

These libraries can be installed by using the pip installer

How To Run

In command line run as python main.py <InputFileName.xlsx> where InputFileName.xlsx consists of the tweets you want to classify.

This will give output as the file Output.csv which consists of the scores of the tweets

* 1 for positive
* -1 for negative
* 0 for neutral

**ANSWER 5**

There are different approaches to putting models into productions, with benefits that can vary dependent on the specific use case. Take for example the use case of churn prediction, there is value in having a static value already that can easily be looked up when someone call a customer service, but there is some extra value that could be gained if for specific events, the model could be re-run with the newly acquired information.

There is generally different ways to both train and server models into production:

* **Train**: one off, batch and real-time/online training
* **Serve:**Batch, Realtime (Database Trigger, Pub/Sub, web-service, inApp)

Each approach having its own set of benefits and tradeoffs that need to be considered.

**1.One off Training**

Models don’t necessarily need to be continuously trained in order to be pushed to production. Quite often a model can be just trained ad-hoc by a data-scientist, and pushed to production until its performance deteriorates enough that they are called upon to refresh it.

**From Jupyter to Production:**

Data Scientists prototyping and doing machine learning tend to operate in their environment of choice [Jupyter](https://jupyter.org/) Notebooks. Essentially an advanced GUI on a [repl](https://en.wikipedia.org/wiki/Read%E2%80%93eval%E2%80%93print_loop" \t "_blank), that allows you to save both code and command outputs.

Using that approach it is more than feasible to push an ad-hoc trained model from some piece of code in Jupyter to production. Different types of libraries and other notebook providers help further tie the link between the data-scientist workbench and production.

**Model Format**

[Pickle](https://docs.python.org/3/library/pickle.html) converts a python object to to a bitstream and allows it to be stored to disk and reloaded at a later time. It is providing a good format to store machine learning models provided that their intended applications are also built in python.

[ONNX](https://github.com/onnx) the Open Neural Network Exchange format, is an open format that supports the storing and porting of predictive model across libraries and languages. Most deep learning libraries support it and sklearn also has a library extension to convert their model to [ONNX’s format](https://github.com/onnx/sklearn-onnx/blob/master/docs/tutorial.rst).

[PMML](https://en.wikipedia.org/wiki/Predictive_Model_Markup_Language) or Predictive model mark-up language, is another interchange format for predictive models. Like for ONNX sklearn also has another library extension for converting the models to [PMML format](https://github.com/jpmml/sklearn2pmml). It has the drawback however of only supporting certain type of prediction models has been around since 1997 and so has a large footprint of applications leveraging the format. Applications such as [SAP](https://archive.sap.com/kmuuid2/a07faefd-61d7-2c10-bba6-89ac5ffc302c/Integrating%20Real-time%20Predictive%20Analytics%20into%20SAP%20Applications.pdf) for instance is able to leverage certain versions of the PMML standard, likewise for CRM applications such as [PEGA](https://community.pega.com/knowledgebase/supported-pmml-model-types).

[POJO and MOJO](http://docs.h2o.ai/h2o/latest-stable/h2o-docs/productionizing.html#about-pojos-and-mojos)are [H2O.ai](https://www.h2o.ai/)’s export format, that intendeds to offers an easily embeddable model into java application. They are however very specific to using the H2O’s platform.

Training

For one off training of models, the model can either be trained and fine tune ad hoc by a data-scientists or training through AutoML libraries. Having an easily reproducible setup, however helps pushing into the next stage of productionalization, i.e.: batch training.

**2.Batch Training**

While not fully necessary to implement a model in production, batch training allows to have a constantly refreshed version of your model based on the latest train.

Batch training can benefit a-lot from AutoML type of frameworks, AutoML enables you to perform/automate activities such as feature processing, feature selection, model selections and parameter optimization. Their recent performance has been on par or bested the most diligent data-scientists.

Different technologies exists that are made to support this continuous batch training, these could for instance be setup through a mix of [airflow](https://medium.com/analytics-and-data/airflow-the-easy-way-f1c26859ee21) to manage the different workflow and an AutoML library such as [tpot](https://epistasislab.github.io/tpot/" \t "_blank), Different cloud providers offer their solutions for AutoML that can be put in a data workflow. Azure for instance integrates machine learning prediction and model training with their [data factory offering](https://azure.microsoft.com/es-es/blog/retraining-and-updating-azure-machine-learning-models-with-azure-data-factory/).

**Real time training**

Real-time training is possible with ‘Online Machine Learning’ models, algorithms supporting this method of training includes K-means (through mini-batch), Linear and Logistic Regression (through Stochastic Gradient Descent) as well as Naive Bayes classifier.

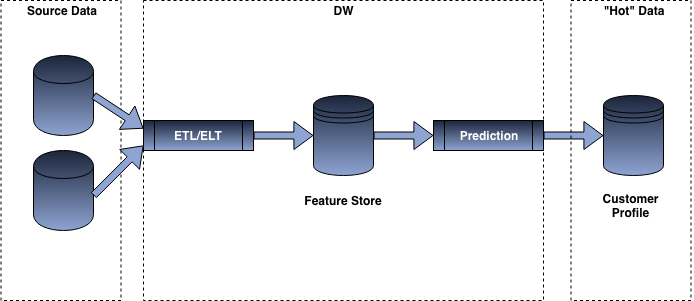
Spark has StreamingLinearAlgorithm/StreamingLinearRegressionWithSGD to perform these operations, sklearn has SGDRegressor and SGDClassifier that can be incrementally trained.

When deploying this type of models there needs to be serious operational support and monitoring as the model can be sensitive to new data and noise, and model performance needs to be monitored on the fly. In offline training, you can filter points of [high leverage](https://en.wikipedia.org/wiki/Leverage_%28statistics%29) and correct for this type of incoming data. This is much harder to do when you are constantly updating your model training based on a stream of new data points.

Another challenge that occurs with training online model is that they don’t decay historical information. This means that, on case there are structural changes in your datasets, the model will need to be anyway re-trained and that there will be a big onus in model lifecycle management.

1. **Batch Prediction Integration**

Batch predictions rely on two different set of information, one is the predictive model and the other one is the features that we will feed the model. In most type of batch prediction architecture, ETL is performed to either fetch pre-calculated features from a specific datastore (feature-store) or performing some type of transformation across multiple datasets to provide the input to the prediction model. The prediction model then iterates over all the rows in the datasets providing the different score.

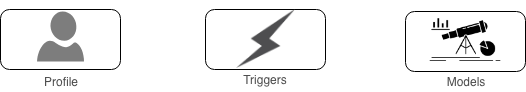


example flow to model serving for batch prediction

Once all the predictions have been computed, we can then “serve” the score to the different systems wanting to consume the information. This can be done in different manner depending on thee use case for which we want to consume the score, for instance if we wanted to consume the score on a front-end application, we would most likely push the data to a “cache” or NoSQL database such as Redis so that we can offer milliseconds responses, while for certain use cases such as the creation of an email journey, we might just be relying on a CSV SFTP export or a data load to a more traditional RDBMS.

1. **Real-time Prediction integration**

Being able to push model into production for real-time applications require 3 base components. A customer/user profile, a set of triggers and predictive models.



**Profile:**The customer profile contains all the related attribute to the customer as well as the different attributes (e.g.: counters) necessary in order to make a given prediction. This is required for customer level prediction in order to reduce the latency of pulling the information from multiple places as well as to simplify the integration of machine learning models in productions. In most cases a similar type of data store would be needed in order to effectively fetch the data needed to power the prediction model.

**Triggers:** Triggers are events causing the initiation of process, they can be for churn for instance, call to a customer service center, checking information within your order history, etc …

**Models:**models need to have been pre-trained and typically exported to one of the 3 formats previously mentioned (pickle, ONNX or PMML) to be something that we could easily port to production.

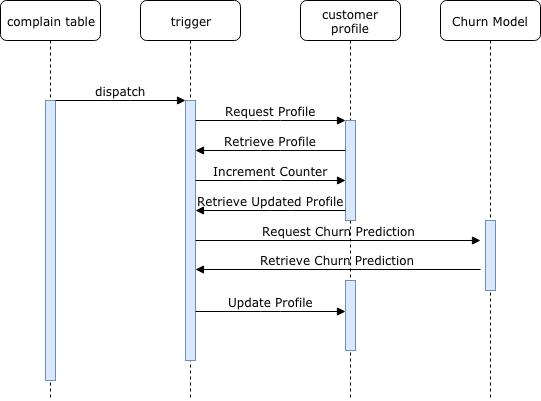
There are quite a few different approaches to putting models for scoring purpose in production:

* Relying on in Database integration: a lot of database vendors have made a significant effort to tie up advanced analytics use cases within the database. Be it by direct integration of Python or R code, to the import of PMML model.
* Exploiting a Pub/Sub model: The prediction model is essentially an application feeding of a data-stream and performing certain operations, such as pulling customer profile information.
* Webservice: Setting up an API wrapper around the model prediction and deploying it as a web-service. Depending on the way the web-service is setup it might or might not do the pull or data needed to power the model.
* inApp: it is also possible to deploy the model directly into a native or web application and have the model be run on local or external data sources.

1. **Database integrations**

If the overall size of your database is fairly small (< 1M user profile) and the update frequency is occasional it can make sense to integrate some of the real-time update process directly within the database.

Postgres possess an integration that allows to run Python code as functions or stored procedure called [PL/Python](http://pl/Python). This implementation has access to all the libraries that are part of the **PYTHONPATH**, and as such are able to use libraries such as Pandas and SKlearn to run some operations.  
This can be coupled with Postgres’ [Triggers](https://www.tutorialspoint.com/postgresql/postgresql_triggers.htm) Mechanism to perform a run of the database and update the churn score. For instance, if a new entry is made to a complaint table, it would be valuable to have the model be re-run in real-time.



**Sequence flow**

The flow could be setup in the following way:

New Event: When a new row is inserted in the complain table, an event trigger is generated.

Trigger: The trigger function would update the number of complaints made by this customer in the customer profile table and fetch the updated record for the customer.

Prediction Request: Based on that it would re-run the churn model through PL/Python and retrieve the prediction.

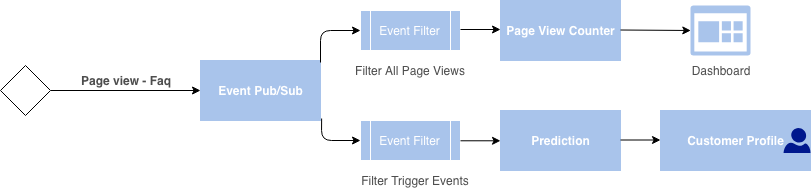
Customer Profile Update: It can then re-update the customer profile with the updated prediction. Downstream flows can then happen upon checking if the customer profile has been updated with new churn prediction value.

**Technologies**

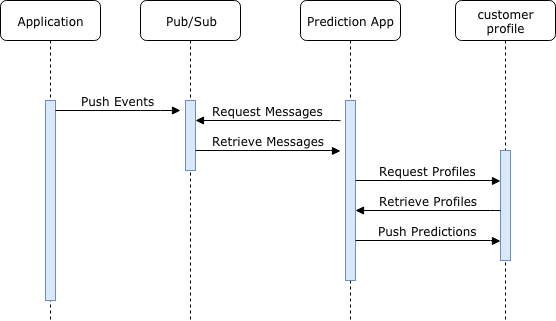
Different databases are able to support the running of Python script, this is the case of PostGres which has a native Python integration as previously mentioned, but also of Ms SQL Server through its’ [Machine Learning Service (in Database)](https://www.sqlshack.com/how-to-use-python-in-sql-server-2017-to-obtain-advanced-data-analytics/), other databases such as Teradata, are able to run R/Python script through an external script command. While Oracle supports [PMML model](https://docs.oracle.com/database/121/DMPRG/GUID-55C6ADBF-DA64-48B6-A424-5F0A59CD406D.htm#DMPRG701) through its data mining extension.

**Pub/Sub**

Implementing real-time prediction through a pub/sub model allows to be able to properly handle the load through throttling. For engineers, it also means that they can just feed the event data through a single “logging” feed, to which different application can subscribe.  
An example, of how this could be setup is shown below:



The page view event is fired to a specific event topic, on which two application subscribe a page view counter, and a prediction. Both of these applications filter out specific relevant event from the topic for their purpose and consume the different messages in the topics. The page view counter app, provides data to power a dashboard, while the prediction app, updates the customer profile.



**Sequence flow:**

Event messages are pushed to the pub/sub topic as they occur, the prediction app poll the topic for new messages. When a new message is retrieved by the prediction app, it will request and retrieve the customer profile and use the message and the profile information to make a prediction. which it will ultimately push back to the customer profile for further use.

A slightly different flow can be setup where the data is first consumed by an “enrichment app” that adds the profile information to the message and then pushes it back to a new topic to finally be consumed by the prediction app and pushed onto the customer profile.

**Technologies:**

The typical open source combination that you would find that support this kind of use case in the data ecosystem is a combination of Kafka and Spark streaming, but a different setup is possible on the cloud. On google notably a google pub-sub/dataflow (Beam) provides a good alternative to that combination, on azure a combination of Azure-Service Bus or EventHub and Azure Functions can serve as a good way to consume the messages and generate these predictions.

1. **Web Service**

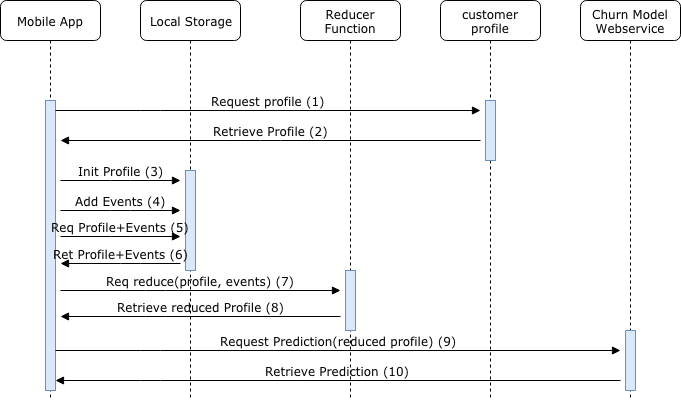
We can implement models into productions as web-services. Implementing predictions model as web-services are particularly useful in engineering teams that are fragmented and that need to handle multiple different interfaces such as web, desktop and mobile.

Interfacing with the web-service could be setup in different way:

* Either providing an identifier and having the web-service pull the required information, compute the prediction and return its’ value
* Or by accepting a payload, converting it to a data-frame, making the prediction and returning its’ value.

The second approach is usually recommended in cases, when there is a lot of interaction happening and a local cache is used to essentially buffer the synchronization with the backend systems, or when needing to make prediction at a different grain than a customer id, for instance when doing session-based predictions.

The systems making use of local storage, tend to have a reducer function, which role is to calculate what would be the customer profile, should the event in local storage be integrated back. As such it provides an approximation of the customer profile based on local data.



**Sequence Flow**

The flow for handling the prediction using a mobile app, with local storage can be described in 4 phases.

Application Initialization (1 to 3)**:**The application initializes, and makes a request to the customer profile, and retrieve its initial value back, and initialize the profile in local storage.

Applications (4): The application stores the different events happening with the application into an array in local storage.

Prediction Preparation (5 to 8)**:**The application wants to retrieve a new churn prediction, and therefore needs to prepare the information it needs to provide to the Churn Web-service. For that, it makes an initial request to local storage to retrieve the values of the profile and the array of events it has stored. Once they are retrieving, it makes a request to a reducer function providing these values as arguments, the reducer function outputs an updated\* profile with the local events incorporated back into this profile.

Web-service Prediction (9 to 10): The application makes a request to the churn prediction web-service, providing the different the updated\*/reduced customer profile from step 8 as part of the payload. The web-service can then use the information provided by the payload to generate the prediction and output its value, back to the application.

**Technologies**

There are quite a few technologies that can be used to power a prediction web-service:

1. **Functions**

AWS Lambda functions, Google Cloud functions and Microsoft Azure Functions (although Python support is currently in Beta) offer an easy to setup interface to easily deploy scalable web-services.  
For instance on Azure a prediction web-service could be implemented through a function looking roughly like this:

1. **Container**

An alternative to functions, is to deploy a flask or Django application through a docker container (Amazon ECS, Azure Container Instance or Google Kubernetes Engine). Azure for instance provides an easy way to setup prediction containers through its’ [Azure Machine Learning service](https://docs.microsoft.com/en-us/azure/machine-learning/service/how-to-deploy-and-where).

Notebooks

Different notebooks providers such as [databricks](https://docs.databricks.com/applications/mlflow/models.html" \t "_blank) and [dataiku](https://www.dataiku.com/dss/features/model-deployment/" \t "_blank) have notably worked on simplifying the model deployment from their environments. These have the feature of setting up a web service to a local environment or deploying to external systems such as Azure ML Service, Kubernetes engine etc…

1. **In App**

In certain situations when there are legal or privacy requirements that do not allow for data to be stored outside of an application, or there exists constraints such as having to upload a large number of files, leveraging a model within the application tend to be the right approach.

Android-ML Kit or the likes of Caffe2 allows to leverage models within native applications, while [Tensorflow.js](https://www.tensorflow.org/js) and [ONNXJS](https://github.com/Microsoft/onnxjs) allow for running models directly in the browser or in apps leveraging javascripts/ Python.