**On the capabilities of Google's prediction API for building practical applications**

by

Data Analytics Group

Object Computing, Inc. (OCI)

**Introduction**

Application development has continued to evolve over the last several decades. We have come so far from building applications on a single machine in a single location to a stage where we are building applications on infrastructure which might be very remote to us. Even more today, we are able to leverage pre-built applications which are offered as a service to make Machine Learning work for us. Enterprises or individuals trying to solve a problem with machine learning, do not have to worry about fault tolerance of the machines, performance limitations, or tuning the machine learning algorithms. Instead, they can invest time on data preparation, incorporating the domain knowledge in the machine learning system and acting on the offerings (prediction results) from these API's to create business value.

Many advanced machine learning capabilities are made available to everyone by Prediction API's through a fast, reliable and cost-effective infrastructure. There are several prediction API's available currently, such as Amazon Machine Learning, Big ML, Google prediction API and many others. While every prediction API differs in terms of their services, there is no single standard metric to evaluate the performance of these API's [a comparison study of some of the prediction API's can be found at

<https://www.programmableweb.com/news/comparing-four-machine-learning-apis-performance/elsewhere-web/2015/09/19>

However, it is possible to identify an API or a combination of them that works better for the context of a specific application which needs expert knowledge and deeper look into the context of the problem, details of which is out of the scope of the current topic, and will be the topic of a future article.  For the purposes of this article, we chose to describe the capabilities of the Google Prediction API. In this article, you will learn about the possibilities the Google's prediction API offers, with respect to a set of representative use cases.

**What is Google Prediction API?**

Google’s Prediction API offers machine learning as a service. It learns from a user’s given training data and provides pattern matching and machine learning capabilities. The Prediction API can predict a numeric value or a categorical value derived from the data provided in the training set. Using these capabilities there is a possibility of building applications ranging from spam detection to recommendation engines without actually worrying about building a model.

The following are the range of use-cases that can be built leveraging the capabilities of Google’s Prediction API:

* Predict future trends from a given historical series of data.
* Detect if a given email is a spam
* Recommend a product/movie to the user based on the interests of a similar user.
* Identify whether a given user will default based on the credit usage history of the user.
* Activity detection from smartphone sensor data using labeled activity datasets.

On a given labeled dataset, prediction API performs the following specific tasks:

* Given a new item, predict a numeric value for that item, based on similarly valued examples in its training data (regression).
* Given a new item, choose a category that describes it best, given a set of similarly categorized items in its training data (classification).

The bottom line of all the above-discussed applications is to predict a world state parameter (target label value) for an unknown example based on the past labeled data examples. The prediction API will take care of building best suitable models using Google’s fast and reliable computing resources. Most prediction queries take less than 200ms.

**How does the prediction API work?**

The implementation of the Google's prediction API can be pretty much defined as a black-box approach. In other words, there is no way to control the model-selection, model-tuning and things that happen in the background during training. The model configuration is restricted to specifying the class of problem whether it is "Classification" vs. "Regression" during the data preparation for the training process. On a very high level, the information flow in the prediction API looks as shown in the following:

Information flow of Prediction API

Input features can contain any type of data, and the API does not impose any constraints on input data types or any configuration process. The API will take care of value normalization, feature selection and even missing values in the dataset. The important step is data preparation, as per the specified format which that Prediction API will accept for training. The data simply looks like a big table, where each record is an input data example and the labels (target value) are specified in the very first column in the training set. The only difference between the training and testing sets is that the first column will not be present in the testing set. The training data looks as shown in the following figure.

The biggest disadvantage with this type of implementation, however, is you cannot predict two different world state parameters (labels) at the same time, which otherwise can be done with your own model implementation.

For the project, you will be working, you need to enable the Prediction API. To use the Prediction API, the only cloud service required is “Cloud Storage” which is enabled by default in your Google Cloud Project. This is required to create a bucket, the location where your training set is uploaded. Prediction API offers a simple way to train machine learning models through a RESTful interface. To authorize the requests, your application must use “OAuth 2.0” protocol and the API does not support any other types of protocols. The detailed information on the authorization is given at:

<https://developers.google.com/identity/protocols/OAuth2>

The training phase is initialized by calling the "trainedmodels.insert" method. The training phase is asynchronous, where you can poll the API using "prediction.trainedmodels.get" method to check the status of the training. In the response, when the "trainingStatus" property changes to "Done" from "Running", then the model training is completed. Only after this, can you start making predictions for new data examples.

The API provides a "trainedmodels.analyze” method which specifies the "modelDescription" that contains the confusion Matrix in the JSON format. It will not provide any additional statistics like precision, recall, or F-score which you would have to calculate manually.

For the resulting model which the API builds, you can make predictions for new example datasets by calling "trainedmodels.predict" method, this returns the parameters "outputLabel" (numeric or String) and "outputMulti" which provide probability measures for each prediction class. The API assigns final predicted labels based on a voting mechanism where a class with highest probability score is predicted as “outputLabel”. The “outputMulti” is useful in making multiple predictions, for example - “top three predicted labels”.  The trained model remains until it is explicitly deleted. Apart from the training session, it does not continue to learn from the Predict queries. However, the API provides a “trainedmodels.update” method which can be used to make update calls with additional examples. The applications that constantly fetch the user data can take advantage of this method to achieve better performance by improving the models on a continuous basis.

**Who is the audience for Prediction API's?**

Traditionally, to implement a machine learning capability, you normally start with preprocessing your data by performing some steps like dealing with missing data, input normalization, dataset splitting into training and validation sets. Then comes the step of model selection, based on the correlations in your training data, in which you identify a model that fits your scenario.

With the Prediction API, you don’t need to worry about these steps, since the API automatically trains and tests a lot of complex models, tuned with different parameters, choosing the best one for the final evaluation. The model which API finally comes up with would have been the one you end up after so many iterations of tuning the model parameters.

Even, the model evaluation is handled by the API itself. All you need to worry and work about is providing a valid data source in the required format to the API. Google's black-box implementation approach for prediction API is intended to ease the implementation for non-coders. In a way, the mathematical expertise required to build, analyze the machine learning models, is eliminated by the capabilities of the API. One can invest more time in problem formulation, data collection and make a flawless end to end implementation, by leveraging the algorithmic capabilities of prediction API.

**Using a Hosted Model**

If you have a specific problem and do not have time, resources, or expertise to build a model, you can use from the gallery of user-submitted models which are hosted at

<https://cloud.google.com/prediction/docs/gallery>

These models may be a paid service or free to use, depending upon the hosted owner. The gallery currently consists of models built by Prediction API team itself and does not contain any models built other users (as of the time of this article is published). The Prediction API is working to enable customers to expose their model for paid use by other API users in the future. As described in the documentation, the only method call supported on a hosted model is “predict”.

The gallery will list the URL required for a prediction call to a specific model. Note that hosted models are versioned; this means that when a model is retrained, it will get a new path that includes the version number. You will need to periodically check the gallery to ensure that you're using the latest version of a hosted model. The model version number appears in the access URL. Different versions might return different scores for the same input.

**Prediction API: A real use-case for sentiment analysis**

To illustrate the use of the prediction API, a simple binary classification problem is shown to classify positive or negative sentiment from the Twitter sentiment analysis corpus dataset found at

<http://thinknook.com/twitter-sentiment-analysis-training-corpus-dataset-2012-09-22/>

For working with a machine learning problem, one of the most important and time-consuming aspects is defining the problem appropriately and preparing the dataset accordingly. Before all that, the problem should be analyzed and assessed to ensure that the problem is appropriate for applying a machine learning approach in the first place. After all, not every problem is solvable using machine learning.

There are two main things you need to clarify and make sure before starting the problem:

* determine if the problem requires regression or classification, and clearly identify what are you going to predict/classify
* identify all necessary assumptions, ensuring they do not affect the scope of the problem

For our current problem, since we are going to identify a predefined label "Negative" or "Positive", this is clearly a classification problem. The second point above is important as you can have a redundant data which will, in turn, affects the model accuracy. A detailed description and scope of machine learning algorithms were described in our previous article which can be found at

<https://www.ociweb.com/resources/publications/sett/february-2015-welcome-to-the-machine-learning/>

Typically, if the features are too specific to the current dataset, you may end up with a very large generalization error, resulting in overfitting. This is the piece you need to handle yourself, to better assist the Prediction API in predicting accurate labels for your test inputs. Although this preprocessing step is not mandatory for using Prediction API, you need it to build a good model corresponding to your training set.

The dataset chosen for the current problem contains about 1.5 million rows and 4 columns. Usually, there may be columns not of interest to a specific problem, so filtering those and including only relevant features will lead to a better model. Accordingly, we prepare our data and make sure to include the label in the first column as mentioned in the API documentation. We now setup the project, which involves 1) creating a Google Cloud Platform project (you can also build on top of an existing one), 2) enabling billing and 3) enabling the prediction API for the project.

Note: A globally unique project id is chosen for the project name and a number is assigned when the cloud platform project is created. Detailed descriptions of these steps are provided at <https://developers.google.com/ad-exchange/rtb/open-bidder/google-app-guide>

Next, you must create a bucket with globally unique name and add the training set file as 'CSV' file to the bucket. For training the model, the "prediction.trainedmodels.insert" method is called, passing a unique name for this predictive model, and the bucket location of the training data as shown below. A full list of the methods is provided at

<https://developers.google.com/apis-explorer/#p/prediction/v1.6/prediction>

POST https://www.googleapis.com/prediction/v1.6/projects/oci-analytics/trainedmodels?key={YOUR\_API\_KEY}  
 {  
 "id": "sentiment-identifier\_12500",  
 "storageDataLocation": "oci-prediction\_api-demo/twitter\_data\_12500.csv"  
 }

Request format for initializing the training

A successful response looks like:

Response 200  
 - Show headers -  
 {  
 "kind": "prediction#training",  
 "id": "sentiment-identifier\_12500",  
 "selfLink": "https://www.googleapis.com/prediction/v1.6/projects/oci-analytics/trainedmodels/sentiment  
 -identifier\_12500",  
 "storageDataLocation": "oci-prediction\_api-demo/twitter\_data\_12500.csv"  
 }

Response to the training request

To check the status of Training, use the "prediction.trainedmodels.get" method, by passing the ID of the predictive model as shown below.

GET https://www.googleapis.com/prediction/v1.6/projects/[PROJECT\_ID]/trainedmodels/sentiment-identifier

Querying about the status of training

After the training is complete, you can send queries to the service to be evaluated against the predictive model. To do so, call the "prediction.trainedmodels.predict" method, passing the name of the model and the query.

POST https://www.googleapis.com/prediction/v1.6/projects/oci-analytics/trainedmodels/sentiment-identifier\_12500/predict?key={YOUR\_API\_KEY}  
 {  
 "input": {  
 "csvInstance": [  
 "I am worried about today's game..."  
 ]  
 }  
 }

Sending a prediction query to the Prediction API

200  
 - Show headers -  
 {  
 "kind": "prediction#output",  
 "id": "sentiment-identifier\_12500",  
 "selfLink": "https://www.googleapis.com/prediction/v1.6/projects/oci-analytics/trainedmodels/sentiment-identifier\_12500/predict",  
 "outputLabel": "NEGATIVE",  
 "outputMulti": [  
 {  
 "label": "NEGATIVE",  
 "score": "0.696235"  
 },  
 {  
 "label": "POSITIVE",  
 "score": "0.303765"  
 }  
 ]  
 }

Prediction response containing the prediction label and probability scores

**Conclusions**

As of now, the Google prediction API provides a machine learning capability that is abstracted and simplified substantially for developers. The control is only in the data preparation and adding additional datasets for updating the model, these forms the either end-points of the machine learning pipeline. Key advantages of this approach are that it saves a lot of time in building the models, and it provides flexibility for adding additional datasets even after the training is completed, enabling simple model updates on the fly.

**Acknowledgements**

We want to thank our colleagues Mike Martinez, Huang-Ming Huang, Carl Turza from OCI for very useful and professional comments that greatly improved the quality of this article.

**Further Reading**

* [The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing](https://software.intel.com/sites/landingpage/IntrinsicsGuide/#techs=AVX), technical details of data flow implementation.