UNIVERSITY OF ILLINOIS CHICAGO

PROJECT

ML Deployment of University Admit Predictor & Recommender using AWS Beanstalk

IDS 594 – MLOPS

SHUBHAM KHODE & ANIRUDHA BALKRISHNA

Contents

Objective

Every year, approximately 15 million students apply to a university to pursue higher education. Most students apply to multiple universities in the hopes of getting accepted by one of them. Therefore, in order to minimize the number of applications and optimize the university selection process, we have built a university admit predictor that predicts if a student is likely to be admitted or rejected by a university. Our objective is to deploy this supervised machine learning model, such that it can be accessed publicly.

Specifically, we will deploy the model using PAAS (Platform as a Service) architecture that tells users whether they are likely to be admitted or rejected by a university with profile information like the university of choice, competitive examination scores of the user, work experience of the user etc. Additionally, the model also has a recommender system that will recommend top universities based on user's profile where he/she is likely to be admitted.

System & Environment

There are many methods and tools available to deploy a ML model. Our aim is to deploy this ML model as a web application in the simplest & fastest way; a type of system that does most of its work by itself and leaves very little ambiguity.

Keeping these features in mind, we decided to proceed with AWS's Elastic Beanstalk.

About AWS Elastic Beanstalk

Elastic Beanstalk is a platform within AWS that is used for deploying and scaling web applications. In simple terms this platform takes the application code and deploys it while provisioning the supporting architecture and compute resources required for the code to run. Elastic Beanstalk also fully manages the patching and security updates for those provisioned resources.

AWS Elastic Beanstalk allows you to quickly deploy applications and services without having to worry about configuring underlying resources, services, operating systems or web servers. It takes care of the hosting infrastructure, coding language interpreter, operating system, security, https service and application layer.

Pros

- **Simple to Setup -** The main advantage of using AWS Beanstalk is when we deploy an application, an environment is created to house the version of the application we are deploying. The environment hosts the required EC2 instances, storage, load balancer, autoscaling groups or anything else required by this version of the application.
- **Auto Scaling -** Elastic Beanstalk automatically scales your application up and down based on your application's specific need using easily adjustable Auto Scaling settings. For example, you can use CPU utilization metrics to trigger Auto Scaling actions. With Elastic Beanstalk, your application can handle peaks in workload or traffic while minimizing your costs.

- **Control -** User has the freedom to select the AWS resources, such as Amazon EC2 instance type and processor type to run the workload on, that are optimal for an application.
- **Versioning -** We can have multiple environments (in different domains or subdomains) so you can have a stage test site.
- **Monitoring -** Elastic Beanstalk provides a unified user interface (UI) to monitor and manage the health of your applications.

Cons

- **Debugging Deployment Failure -** In case of deployment failure, it is very difficult to pinpoint the reason for failure.
- **Versioning Limit -** Not more than 500 versions can be created.

Cost Benefit Analysis

There is no specific charge for AWS Elastic Beanstalk. It employs a "Pay as you Use" model, so a developer pays for the instances and services he/she uses and when the services are being used. In certain scenarios, AWS EB can save up to 400% on costs compared to systems like Heroku. (Although the core offering is not the same, this is a fairly good comparison)

Methodology

Process

There are 3 basic steps in deploying the model to make it publicly accessible. These steps are mentioned in the representation below –

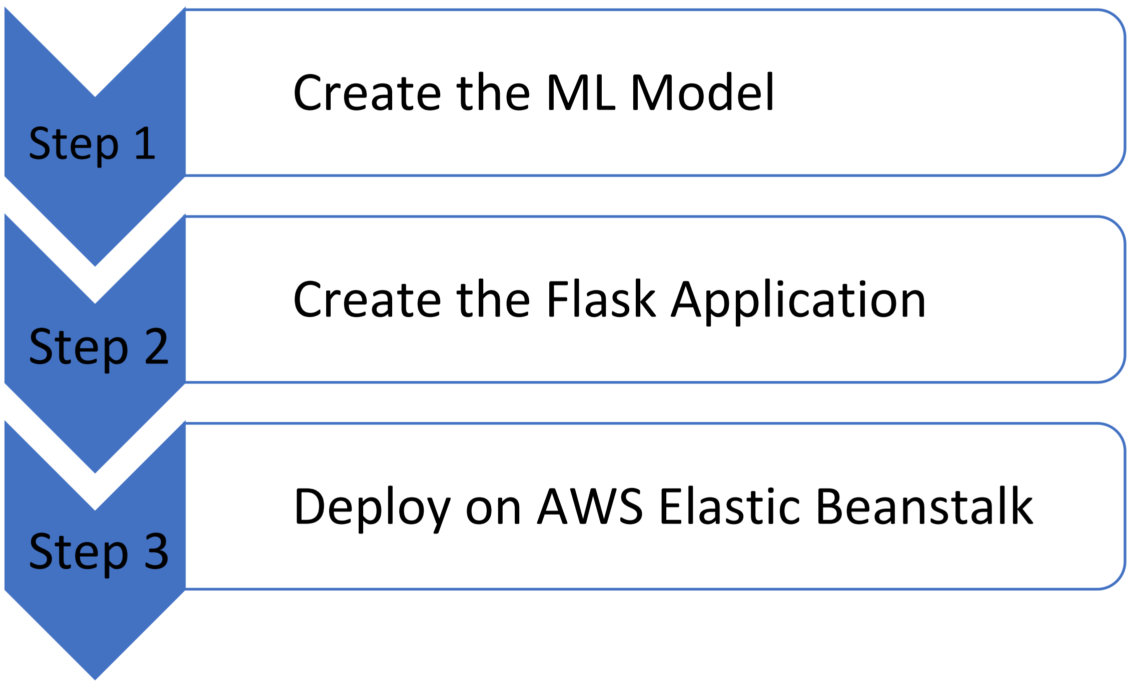


Fig. 2 – Deployment Process

Let us look at each of these steps in detail –

1. ***Creation of the ML Model –***

   The model is currently designed for students that wish to pursue a Master's in Information systems related courses in US.

   a. Data Cleaning and pre-processing:
      The data for the model is obtained from Yocket profiles (Yocket is an application that serves as a social network for students pursuing higher education). The dataset is available on Kaggle, which was cleaned and processed in MS Excel. The undergraduate grades were brought to the same scale (out of 10) while work experience was transformed to months. Input rows missing value for target variable and some additional redundant rows were removed. The final dataset contains 971 rows with 6 features that influence the target label (Admit/Reject). Further, the feature for university name was label encoded using rankings of universities (from US News), while allows us to convert name of university to its numeric equivalent for modelling.

   b. Classifier:
      The predictor is essentially a random forest classifier (with 'gini' as criterion) that predicts if a student is likely to be admitted or rejected by a university.

   c. Recommender:
      In addition to the predictor, we have incorporated a recommender that employs k-nearest neighbours as the algorithm along with cosine similarity as the statistical measure of similarity. A standard scaler is used to ensure distances calculated for each value are in similar range. The recommender returns top 5 'neighbours' where a user may get admitted based on his/her provided profile information.

   The model also uses serialization technique called "pickling" to effectively transfer data as a byte stream. We use pickling thrice - once for the classifier output, then for scaling the values and finally to store the recommender results as '.pkl' objects.

2. ***Creation of Flask Application –***

   Flask provides the framework in python to build our web application. This application takes the profile information as an input from user, feeds it to the model and displays the prediction and recommendations. Behind the scenes, Flask utilizes Jinja engine to dynamically build HTML pages. Additionally, we integrate Flask with WTForms to fetch input from user in a form structure. Pickle files are loaded and de-serialized to convert it back to respective python objects.

3. **Deploying on AWS Elastic Beanstalk –**

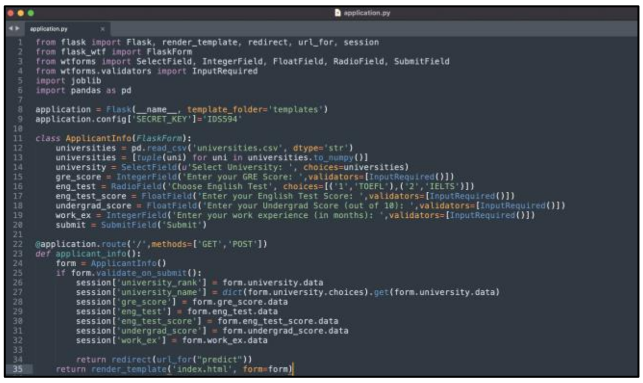
   Before deploying on AWS Elastic Beanstalk, we need to do three important steps. Firstly, generate a ‘requirements.txt’ file (obtained by using the command ‘pip freeze -> requirements.txt’ in terminal) that helps the cloud application know all the libraries that will be required to run with the flask application. Secondly, create a folder called “.ebextensions” and add a configuration file that is used to provide instruction to the cloud application on what function to run and from which container will the function be run. Lastly, zip these files at the root folder level.

   For deployment, create an AWS Elastic Beanstalk environment, application, select platform and upload the zip file.

Tutorial

Flask Application

Our flask application contains a class ‘ApplicantInfo’ which collects input from the user in a flask form and saves it in a session. The default route checks if the input is valid and redirects to another page for prediction. The predict route uses ‘get_prediction’ and ‘get_recommendations’ functions to get their respective outputs and render it on screen.



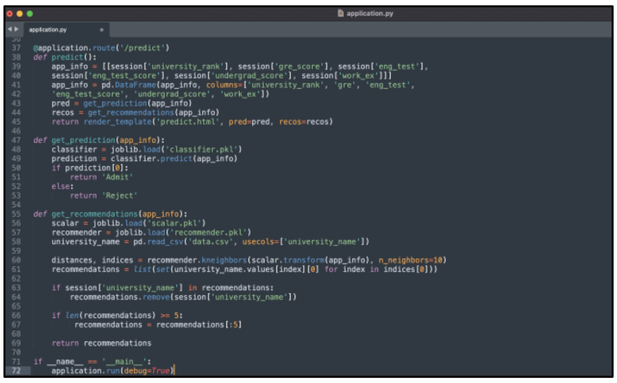
```python
from flask import Flask, render_template, redirect, url_for, session
from flask_wtf import FlaskForm
from wtforms import SelectField, IntegerField, FloatField, RadioField, SubmitField
from wtforms.validators import InputRequired
import joblib
import pandas as pd

application = Flask(__name__, template_folder='templates')
application.config['SECRET_KEY']='IDS594'

class ApplicantInfo(FlaskForm):
    universities = pd.read_csv('universities.csv', dtype='str')
    universities = [tuple(uni) for uni in universities.to_numpy()]
    university = SelectField(u'Select University: ', choices=universities)
    gre_score = IntegerField('Enter your GRE Score: ',validators=[InputRequired()])
    eng_test = RadioField('Choose English Test', choices=[('1','TOEFL'),('2','IELTS')])
    eng_test_score = FloatField('Enter your English Test Score: ',validators=[InputRequired()])
    undergrad_score = FloatField('Enter your Undergrad Score (out of 10): ',validators=[InputRequired()])
    work_ex = IntegerField('Enter your work experience (in months): ',validators=[InputRequired()])
    submit = SubmitField('Submit')

@application.route('/',methods=['GET','POST'])
def applicant_info():
    form = ApplicantInfo()
    if form.validate_on_submit():
        session['university_rank'] = form.university.data
        session['university_name'] = dict(form.university.choices).get(form.university.data)
        session['gre_score'] = form.gre_score.data
        session['eng_test'] = form.eng_test.data
        session['eng_test_score'] = form.eng_test_score.data
        session['undergrad_score'] = form.undergrad_score.data
        session['work_ex'] = form.work_ex.data

        return redirect(url_for("predict"))
    return render_template('index.html', form=form)
```



```python
@application.route('/predict')
def predict():
    app_info = [[session['university_rank'], session['gre_score'], session['eng_test'],
    session['eng_test_score'], session['undergrad_score'], session['work_ex']]]
    app_info = pd.DataFrame(app_info, columns=['university_rank', 'gre', 'eng_test',
    'eng_test_score', 'undergrad_score', 'work_ex'])
    pred = get_prediction(app_info)
    recos = get_recommendations(app_info)
    return render_template('predict.html', pred=pred, recos=recos)

def get_prediction(app_info):
    classifier = joblib.load('classifier.pkl')
    prediction = classifier.predict(app_info)
    if prediction[0]:
        return 'Admit'
    else:
        return 'Reject'

def get_recommendations(app_info):
    scalar = joblib.load('scalar.pkl')
    recommender = joblib.load('recommender.pkl')
    university_name = pd.read_csv('data.csv', usecols=['university_name'])

    distances, indices = recommender.kneighbors(scalar.transform(app_info), n_neighbors=10)
    recommendations = list(set(university_name.values[index][0] for index in indices[0]))

    if session['university_name'] in recommendations:
        recommendations.remove(session['university_name'])

    if len(recommendations) >= 5:
        recommendations = recommendations[:5]

    return recommendations

if __name__ == '__main__':
    application.run(debug=True)
```

Deployment

Step 1: Open AWS Elastic Beanstalk - Sign into an AWS account and find the Elastic beanstalk service (can be located under the compute services in your AWS console)

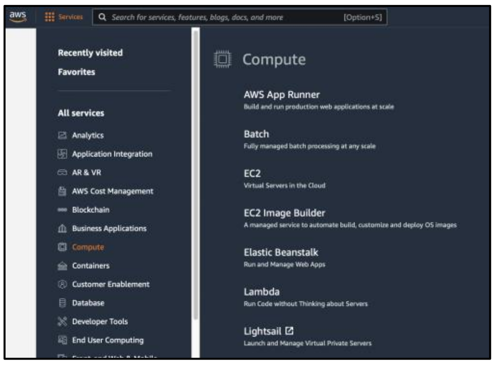
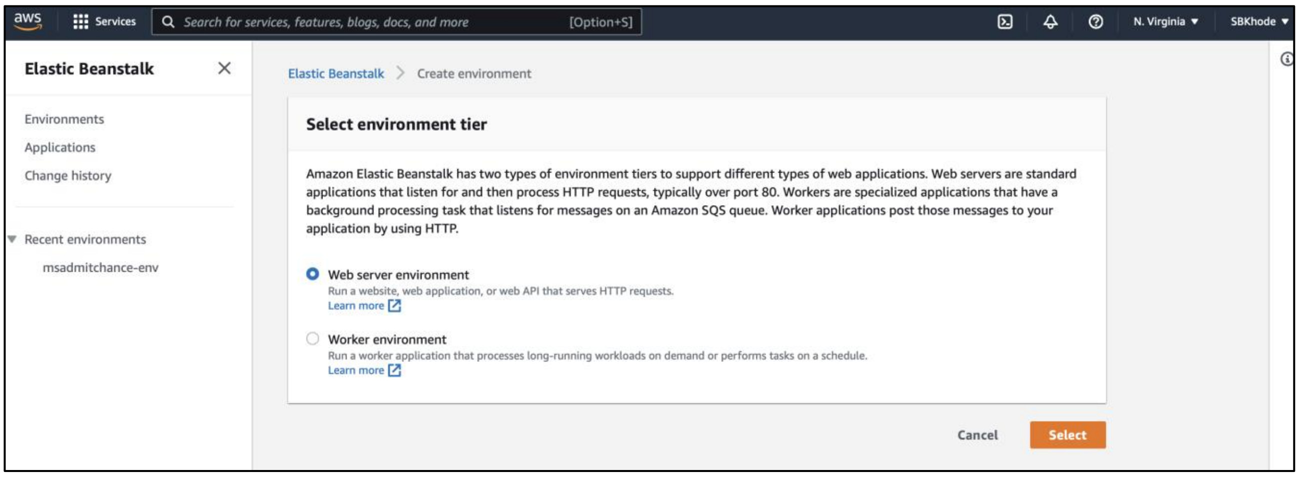


Fig. 3 – AWS Beanstalk

Step 2: Web Application and Environment Setup - First step is to create a new web server environment and application. Give them unique names and add tags, if required.



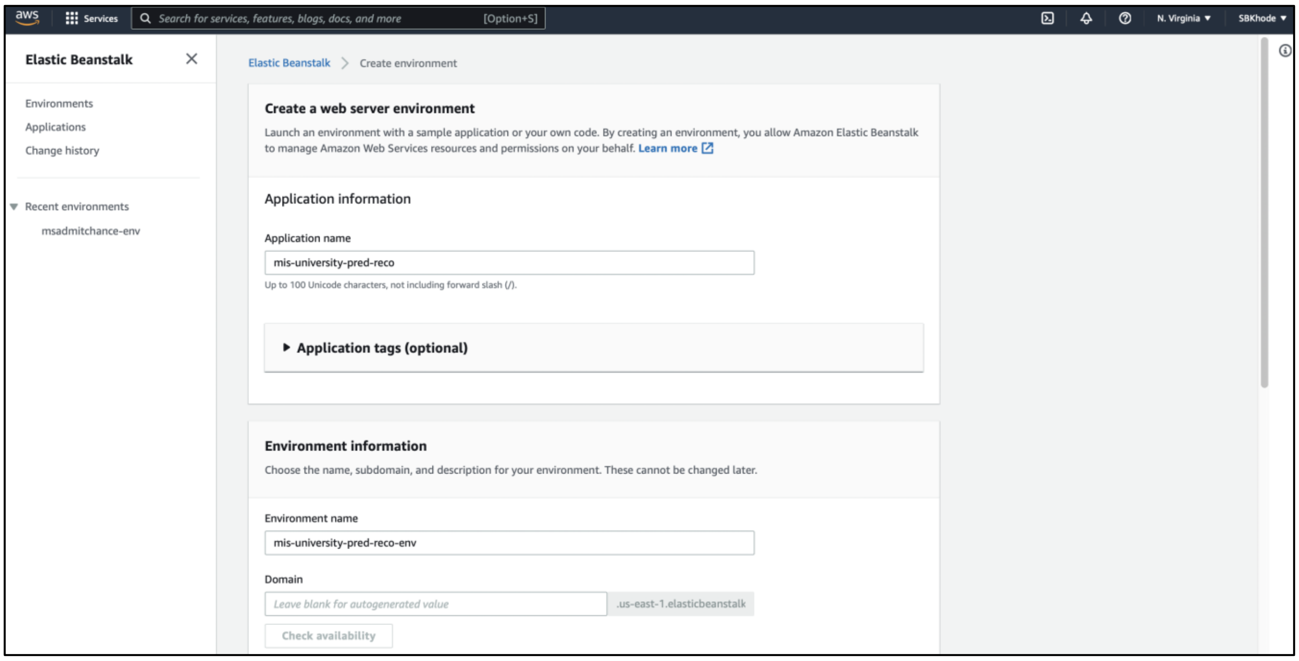
Select environment tier

Amazon Elastic Beanstalk has two types of environment tiers to support different types of web applications. Web servers are standard applications that listen for and then process HTTP requests, typically over port 80. Workers are specialized applications that have a background processing task that listens for messages on an Amazon SQS queue. Worker applications post those messages to your application by using HTTP.

[✓ Web server environment]
Run a website, web application, or web API that serves HTTP requests.
Learn more

Worker environment
Run a worker application that processes long-running workloads on demand or performs tasks on a schedule.
Learn more

Fig. 4 – Select web server Environment



Create a web server environment

Launch an environment with a sample application or your own code. By creating an environment, you allow Amazon Elastic Beanstalk to manage Amazon Web Services resources and permissions on your behalf. Learn more

Application information

Application name: mis-university-pred-reco

Up to 100 Unicode characters, not including forward slash (/).

Application tags (optional)

Environment information

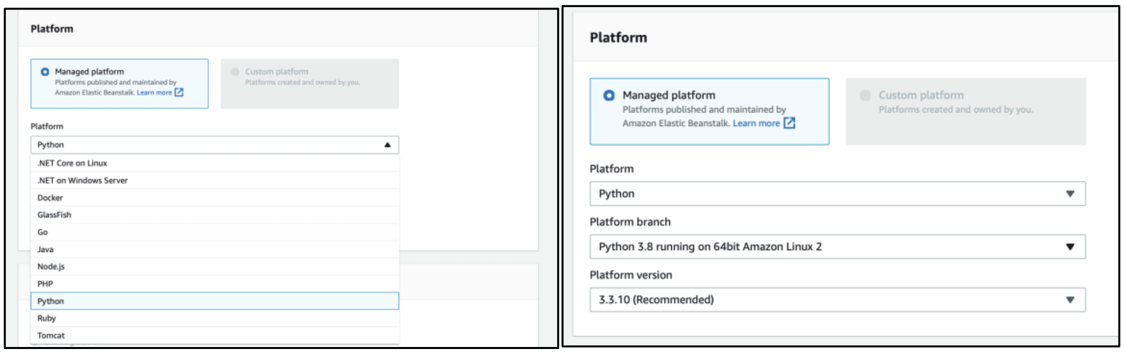
Choose the name, subdomain, and description for your environment. These cannot be changed later.

Environment name: mis-university-pred-reco-env

Domain: Leave blank for autogenerated value .us-east-1.elasticbeanstalk

Fig. 5 – Create Application and Environment

Step 3: Select Platform - Select option ‘Managed Platform’ and choose ‘Python’.



Platform

[✓ Managed platform]
Platforms published and maintained by Amazon Elastic Beanstalk. Learn more

Custom platform
Platforms created and owned by you.

Platform: Python

Platform

[✓ Managed platform]
Platforms published and maintained by Amazon Elastic Beanstalk. Learn more

Custom platform
Platforms created and owned by you.

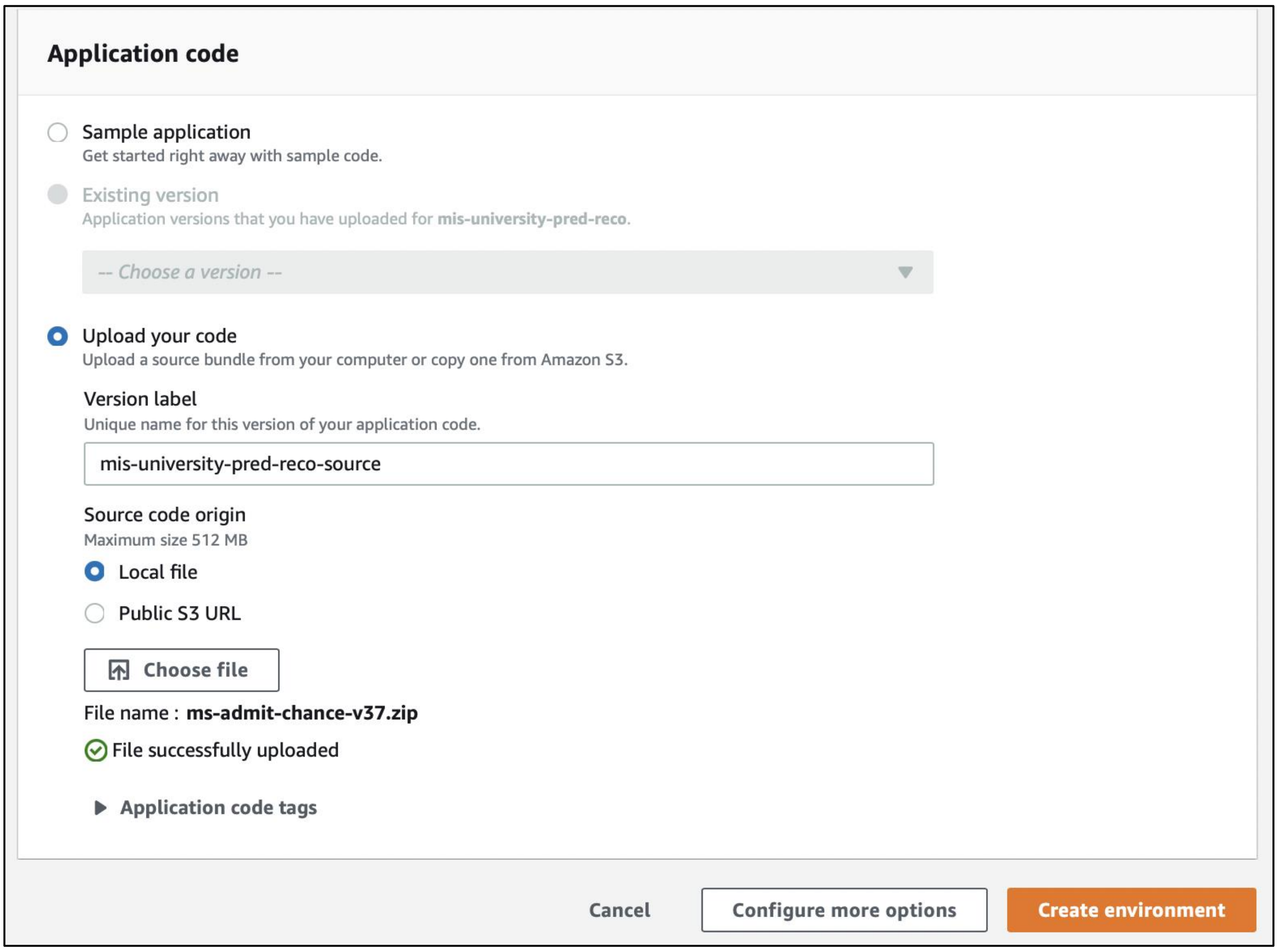
Platform: Python

Platform branch: Python 3.8 running on 64bit Amazon Linux 2

Platform version: 3.3.10 (Recommended)

Fig. 6 – EB Platform Selection

Step 4: Upload application code – We can either upload our code files locally or use S3 storage URL. For our case, we upload our .zip application file. Additional configuration can be set, if needed.



Application code

Sample application
Get started right away with sample code.

Existing version
Application versions that you have uploaded for **mis-university-pred-reco**.

-- Choose a version --

Upload your code
Upload a source bundle from your computer or copy one from Amazon S3.

Version label
Unique name for this version of your application code.

mis-university-pred-reco-source

Source code origin
Maximum size 512 MB

Local file

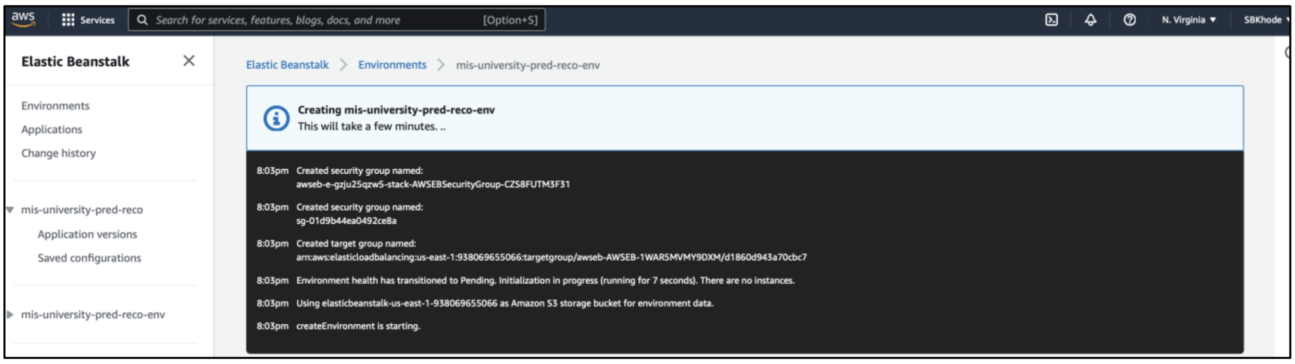
Public S3 URL

File name : **ms-admit-chance-v37.zip**

File successfully uploaded

Fig. 7 – EB Upload application source file

Step 5: Web Application Configuration – Once you create environment, Elastic Beanstalk will start to create all the required resources and display a status screen with the log of the supporting resources.



aws
Services
Search for services, features, blogs, docs, and more [Option+S]
N. Virginia
SBKhode
Elastic Beanstalk
Environments
Applications
Change history
mis-university-pred-reco
Application versions
Saved configurations
mis-university-pred-reco-env
Elastic Beanstalk > Environments > mis-university-pred-reco-env

Creating mis-university-pred-reco-env
This will take a few minutes. ..

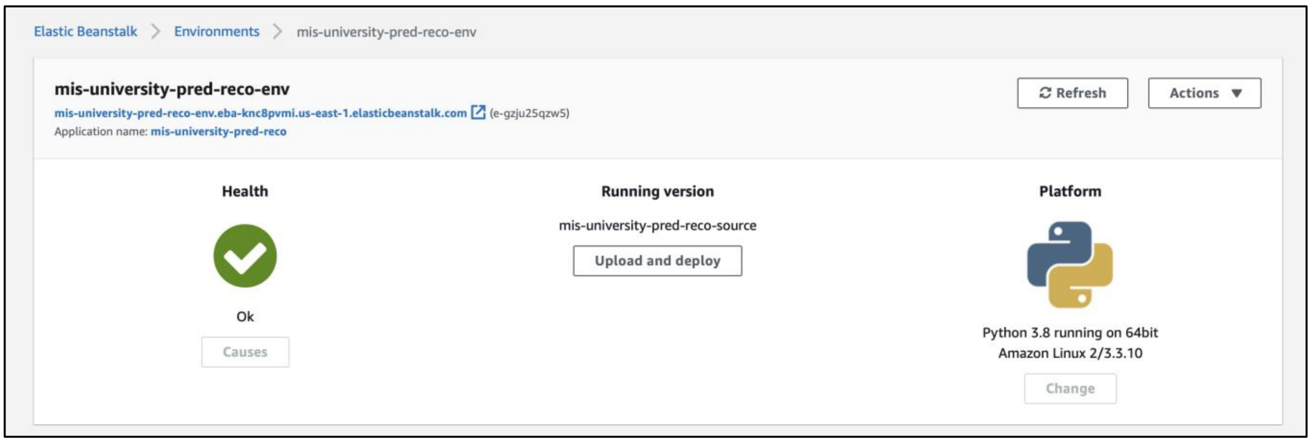
```
8:03pm  Created security group named:
        awseb-e-gzju25qzw5-stack-AWSEBSecurityGroup-CZS8FUTM3F31
8:03pm  Created security group named:
        sg-01d9b44ea0492ce8a
8:03pm  Created target group named:
        arn:aws:elasticloadbalancing:us-east-1:938069655066:targetgroup/awseb-AWSEB-1WAR5MVMY9DXM/d1860d943a70cbc7
8:03pm  Environment health has transitioned to Pending. Initialization in progress (running for 7 seconds). There are no instances.
8:03pm  Using elasticbeanstalk-us-east-1-938069655066 as Amazon S3 storage bucket for environment data.
8:03pm  createEnvironment is starting.
```

Fig. 8 – Deployment configuration display

The process for the sample application creates:

- S3 bucket
- ELB
- Security Group
- Auto Scaling Group
- ASG Group Policy
- CloudWatch Alarms
- Load Balancer Listener
- EC2 instances
- Web App Environment

At the end, the following screen will be displayed showing the health of the application. We can now open the URL displayed above the health check to open the application. The application we created is in this URL.



Elastic Beanstalk > Environments > mis-university-pred-reco-env

mis-university-pred-reco-env

mis-university-pred-reco-env.eba-knc8pvmi.us-east-1.elasticbeanstalk.com (e-gzju25qzw5)

Application name: mis-university-pred-reco

Health: Ok

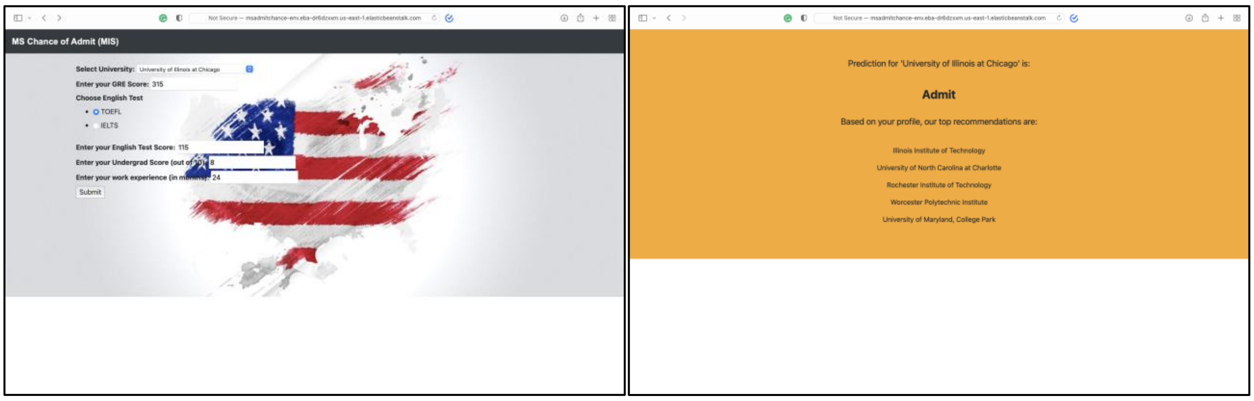
Running version: mis-university-pred-reco-source

Platform: Python 3.8 running on 64bit Amazon Linux 2/3.3.10

Fig. 9 – Health Status Screen

Result

The final deployed model is available on this link. The webpage is displayed below –



Not Secure — msadmitchance-env.eba-dr6dzxem.us-east-1.elasticbeanstalk.com

MS Chance of Admit (MIS)

Select University: University of Illinois at Chicago

Enter your GRE Score: 315

Choose English Test

- [✓ TOEFL]
- IELTS

Enter your English Test Score: 115

Enter your Undergrad Score (out of 10): 8

Enter your work experience (in months): 24

Not Secure — msadmitchance-env.eba-dr6dzxem.us-east-1.elasticbeanstalk.com

Prediction for 'University of Illinois at Chicago' is:

Admit

Based on your profile, our top recommendations are:

Illinois Institute of Technology

University of North Carolina at Charlotte

Rochester Institute of Technology

Worcester Polytechnic Institute

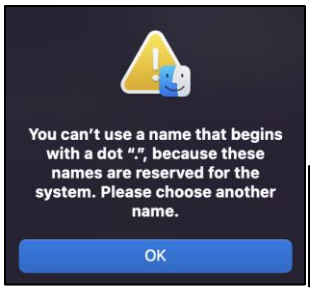
University of Maryland, College Park

Fig. 10 – Final model as a web application

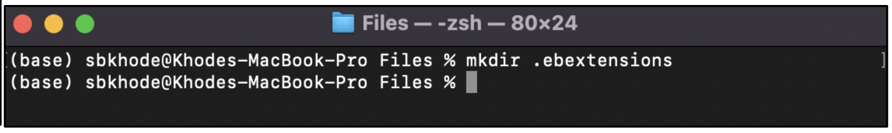
As evident in the image above, the model requires a University, GRE Score, English Proficiency test choice, English test score, Undergraduate score & Work experience in months. Once the details are filled the result will be displayed.

Challenges and Lesson Learnt

1. The flask application file should be named as 'application.py', else it won't work. Also, throughout the code, the flask object should also be named as 'application'.
2. If you are using macOS, it doesn't allow you to create a folder that starts with '.' (we need it for '.ebextensions') folder. To fix this, create such folders directly using 'mkdir' command via terminal.

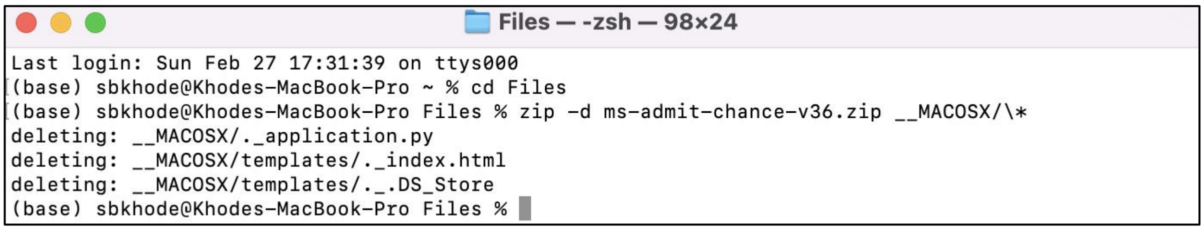


You can't use a name that begins with a dot ".", because these names are reserved for the system. Please choose another name.



```
(base) sbkhode@Khodes-MacBook-Pro Files % mkdir .ebextensions
(base) sbkhode@Khodes-MacBook-Pro Files %
```

3. AWS EB requires all files to be in proper defined folder structure. If you are using macOS, the built-in zip compressor causes an extra folder titled "__MACOSX" to be created in the extracted zip. To fix this, use command 'zip -d filename.zip __MACOSX/*' in terminal.



```
Last login: Sun Feb 27 17:31:39 on ttys000
(base) sbkhode@Khodes-MacBook-Pro ~ % cd Files
(base) sbkhode@Khodes-MacBook-Pro Files % zip -d ms-admit-chance-v36.zip __MACOSX/\*
deleting: __MACOSX/._application.py
deleting: __MACOSX/templates/._index.html
deleting: __MACOSX/templates/._.DS_Store
(base) sbkhode@Khodes-MacBook-Pro Files %
```

Future Scope

The next step would be to establish a system to maintain version control & retraining. AWS Elastic Beanstalk supports lifecycle management of our application versions on the AWS Management Console. Application version lifecycle management allows us to auto-delete the undeployed application versions based on application version count or application version age. Previously, one could only configure application version lifecycle policy through the APIs, SDKs, EB CLI, and AWS CLI. Now, we can set the lifecycle settings through the AWS Management Console.

References

[1] Deployment Architecture

[2] AWS Elastic Beanstalk

[3] Process of Setting up Flask API & Deployment