# PRIOR KNOWLEDGE

## MACHINE LEARNING

## SUPERVISED LEARNING:

- Suppose you had a basket and it is fulled with some fresher fruits your task is to arrange the same type fruits at one place.
- Suppose the fruits are apple ,banana,cherry,grape.
- So you already know your previous work that, the shape of each and every fruit so it is easy to arrange the same type of fruits at one place.
- Here your previous work is called as train data in data mining.
- So you already learn the things from your train data, this is because of you have a response variable which says you that if some fruit have so and so features it is grape, like that for each and every fruit.
- This type of data you will get from the train data.
- This type of learning s called as supervised learning.
- This type solving problem come under classification.
- So you already learn the things so you can do your job confidently.

UNSUPERVISED LEARNING :

- Suppose you had a basket and it is fulled with some fresh fruits your task is to arrange the same type fruits at one place.
- This time you don't know anything about that fruits , you are first time seeing these fruits so how will you arrange the same type of fruits.
- What you will do first you take on fruit and you will select any physical character of that particular fruit. Suppose you taken colors.

- Then the group will be something like this.
- RED COLOUR GROUP : apple & cherry fruits.
  GREEN COLOR AND SMALL SIZE : grapes
- This type of learning is know unsupervised learning.

## CLASSIFICATION :

- Classification is a process of categorizing a given set of data into classes, it can be performed on both structured or unstructured data.
- The process starts with predicting the class of given data points. The classes are often referred to as target, label or catagories.

## REGRESSION:

- A technique for determining the statistical relationship between two or more variables where a change in a dependent variable is associated with , and depends on , a change in one or more independent variables.
- A regression problem is used when the output variable is a real or continuous value , such as"salary" or "weight".

**LOGISTIC REGRESSION** :

- A type of classification algorithm.
- Based on linear regression to evaluate output and to minimize the error .
- Named after the method it uses to evaluate the outputs the logit function.
- Logistic regression just has a transformation based on linear regression hypothesis.
- For logistic regression, focusing on binary classification here , we have class 0 and 1.

- To compare with the target , we want to constrain predictions tosome values between 0 and 1.
- That's why SIGMOID FUNCTION is applied on the raw model output and provides the ability to predict with probability.
- Logistic function , also called the sigmoid function was developed by statisticians to describe proper of population growth in ecology, rising quickly and maxing out at the carrying capacity of the environment .
- Its an s-shaped curve that can take any real –valued number and map it into value between 0and1, but never exactly at those limits: 1/(1+E^-value).

FLASK IN PYTHON :

- Flask is a micro framework for python
- Easy to code
- Easy to configure
- Flask won't make many decisions for you , such as what database to use
- Has an excellent documentation
- RESTful
- Testable
-

## FLASK IS

OPEN SOURCE



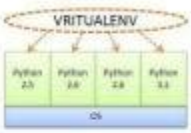
## LETS CODE

## CONFIGURING THE ENVIRONMENT

# VIRTUALENV

Installing

```
pip install virtualenv
```




Configuring

```
cd ~/YOUR_MAIN_FOLDER
mkdir virtualenvs
cd virtualenvs
virtualenv NAME_YOUR_ENV --no-site-packages
```

Activating

```
source NAME_YOUR_ENV/bin/activate
```

Deactivating

```
deactivate
```

# 6 LINES OF CODE AND THIS WORKS


# SQLALCHEMY

Installing `pip install Flask-Sqlalchemy`

Coding the Model

bravi\bravi\app\models.py

```python
from braviapp import db

class News(db.Model):
    # Define the properties mapped to database columns
    id = db.Column(db.Integer, primary_key = True)
    title = db.Column(db.String(100), nullable = False)
    text = db.Column(db.Text, nullable = False)

    def __init__(self, title, text):
        self.title = title
        self.text = text

    def __repr__(self):
        return '<News %r>' % self.title
```

# WTFORMS

Installing `pip install Flask-WTF`

Coding the Form

```
# third party imports
from flask.ext.wtf import Form, TextField, TextAreaField,
Required


class NewsCreateForm(Form):
      title = TextField('Title', [Required()])
      text = TextAreaField('Text', [Required()])
```

bravi\bravi\app\models.py

# TEMPLATES – BASE HTML

```
<!DOCTYPE html>
<html>
    <head>
        <title>{% block title %}Flask - Bravi{% endblock %}</title>
        <link rel="stylesheet" href="/static/css/main.css" />
        {% block css %}{% endblock %}
        {% block script %}{% endblock %}
    </head>
    <body>
        <div id="wrapper{{ wrapper_type }}">
                    <div id="header">
                    {% block header %}
                            <h1><a id="link-title-home"
                            href="{{ url_for('all') }}">Bravi News</a></h1>
                    {% endblock %}
                    </div>
                    <div id="messages">
                    {% for category, msg in get_flashed_messages(with_categories=true) %}
                        <p class="messages flash-{{ category }}">{{ msg }}</p>
                    {% endfor %}
                    </div>
                    <div id="content" class="shadow">
                            {% block content %}{% endblock %}
                    </div>
                    <div id="footer">{% block footer %}{% endblock %}</div>
        </div>
    </body>
</html>
```

bravi\templates\base.html

1

# TEMPLATES – CREATE

```
{% extends "base.html" %}

{% block content %}
  <h2>Create News</h2>
  {% from "macros.html" import render_field %}
  <form method="POST" action="." class="form">
    {{ form.csrf_token }}
    {{ render_field(form.title, class="input text") }}
    {{ render_field(form.text, class="input text") }}
    <input type="submit" value="Create">
  </form>
{% endblock %}


{% block footer %}
  <a class="bt-action bt-action-list" href="{{ url_for('all')
}}">
    <span>News</span>
  </a>
{% endblock %}
```

bravi\templates\news_create.html

2

## JINJA 2 – MACROS - DRY

```
{% macro render_field(field) %}
        <div class="form_field">
        {{ field.label(class="label") }}
        {% if field.errors %}
                {% set css_class = 'has_error' +
kwargs.pop('class', '') %}
                {{ field(class=css_class, **kwargs) }}
                <ul class="errors">
                        {% for error in field.errors %}
                        <li>{{ error|e }}</li>
                        {% endfor %}
                </ul>
        {% else %}
                {{ field(**kwargs) }}
        {% endif %}
        </div>
{% endmacro %}
```

bravi\templates\macros.html

## VIEWS

```
from flask import request, flash, redirect, url_for, render_template
from braviapp import braviapp, db
from braviapp.forms import NewsCreateForm
from braviapp.models import News

@braviapp.errorhandler(404)
def not_found(error):
        flash('You tried access a page that not exists')
        return redirect(url_for('all'))

@braviapp.route('/')
def all():
        #from news_model import News
        news = News.query.all()
        return render_template('news_list.html', news=news)

@braviapp.route('/create/', methods=['GET', 'POST'])
def create():
        form = NewsCreateForm(request.form)

        # make sure data are valid
        if form.validate_on_submit():

                news = News(form.title.data, form.text.data)
                # save on database
                db.session.add(news)
                db.session.commit()

                flash('The news has been created successfully')
                return redirect(url_for('all'))
        return render_template('news_create.html', form=form)
```

bravi\braviapp\views.py



## CORE APP

```
# third party imports
from flask import Flask
from flask.ext.sqlalchemy import SQLAlchemy

# Initialize the app from Flask
braviapp = Flask(__name__)
braviapp.config.from_object('settings')

db = SQLAlchemy(braviapp)

# local application imports
from braviapp import views
```

bravi\braviapp__init__.py

## SETTINGS FILE

bravi\settings.py

```
import os
_basedir = os.path.abspath(os.path.dirname(__file__))

DEBUG = False

ADMINS = frozenset(['youremail@yourdomain.com'])
SECRET_KEY = 'SECRET_KEY_FOR_SESSION_SIGNING'

# Define the path of our database inside the root
application, where 'app.db' is the database's name
SQLALCHEMY_DATABASE_URI = 'sqlite:///' +
os.path.join(_basedir, 'app.db')
DATABASE_CONNECT_OPTION = {}

CSRF_ENABLED = True
CSRF_SESSION_KEY = 'SOMETHING_IMPOSSIBLE_TO_GUEES'
```

## SQLALCHEMY

**Helper to reset the database file**

bravi\initialize_db.py

```
from app import db

# Drop all tables from db file
db.drop_all()

# Create all tables on db file,
# copying the structure from the definition on the Models
db.create_all()
```

Running `python initialize_db.py`

## RUNNING

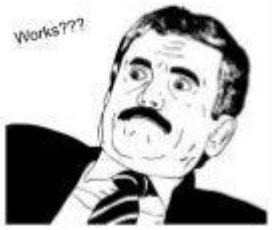
**Helper to initialize the application**

bravi\initialize_app.py

```
from braviapp import braviapp as application
application.run(debug=True,port=8080)
```

Running `python initialize_app.py`

## LETS TRY




NameError
DEBUGGER
Traceback (most recent call last)
return self.wsgi_app(environ, start_response)
response = self.make_response(self.handle_exception(e))
response = self.full_dispatch_request()
rv = self.handle_user_exception(e)
rv = self.dispatch_request()
return self.view_functions[rule.endpoint](**req.view_args)
news = News.query.all()
NameError: global name 'news' is not defined