

---

# Realtime Predictive Analytics

Using scikit-learn and RabbitMQ

---

Michael Becker

March 2014 PhillyPUG

---

# Who is this guy?

---

Data guy @ AWeber  
@beckerfuffle  
beckerfuffle.com

These slides and more @ github.com/mdbecker



Thursday, March 20, 14

2

Thanks everybody for coming to my talk! My name is Michael Becker, I work for the Data Analysis and Management Ninjas at AWeber. AWeber is an email service provider with over 120 thousand customers. I'm also the founder of the DataPhilly meetup with over 600 members. If you're not already a member, well shame on you!

You can find me online @beckerfuffle on Twitter. My website is beckerfuffle.com. Materials from this talk can be found on my github.

# What my coworkers think I do

---

$$h_{w,b}(x) = g(w^T x + b)$$

$$\hat{\gamma}^{(i)} = y^{(i)}(w^T x + b)$$

$$\hat{\gamma} = \min_{i=1,\dots,m} \hat{\gamma}^{(i)}$$

$$w^T \left( x^{(i)} - \gamma^{(i)} \frac{w}{||w||} \right) + b = 0$$

---

Thursday, March 20, 14

3

Working on the DAMN team at AWeber, I've introduced several predictive algorithms into our application. My co-workers think I'm some kind of math superhero, reading scholarly papers by day and fighting crime by night! The truth is much more sinister.

# What I actually do

---

```
from sklearn.svm import SVC
```

---

Thursday, March 20, 14

4

Fortunately for me, I don't have to be a math genius to look like a superhero, I just have to use scikit-learn! While I'll cover some math in this talk, I'll mainly be keeping things high-level. This is because I'm not a math genius. If I get any of the math wrong, feel free to call me out on the interwebs.

# What I'll cover

---

- Scikit-learn overview

---

Thursday, March 20, 14

5

This talk will cover a lot of the logistics behind utilizing a trained scikit-learn model in a real-life production environment.

I'll start off by giving a brief overview of supervised machine learning and text processing with scikit-learn.

# What I'll cover

---

- Scikit-learn overview
- Model Distribution

I'll cover how to distribute your model

# What I'll cover

---

- Scikit-learn overview
- Model Distribution
- Data flow

---

Thursday, March 20, 14

7

I'll discuss how to get new data to your model for prediction.

# What I'll cover

---

- Scikit-learn overview
- Model Distribution
- Data flow
- RabbitMQ

---

Thursday, March 20, 14

8

I'll introduce RabbitMQ, what it is and why you should care.



# What I'll cover

---

- Scikit-learn overview
- Model Distribution
- Data flow
- RabbitMQ
- Demo

---

Thursday, March 20, 14

9

I'll demonstrate how we can put all this together into a finished product

# What I'll cover

---

- Scikit-learn overview
- Model Distribution
- Data flow
- RabbitMQ
- Demo
- Scalability

I'll discuss how to scale your model

# What I'll cover

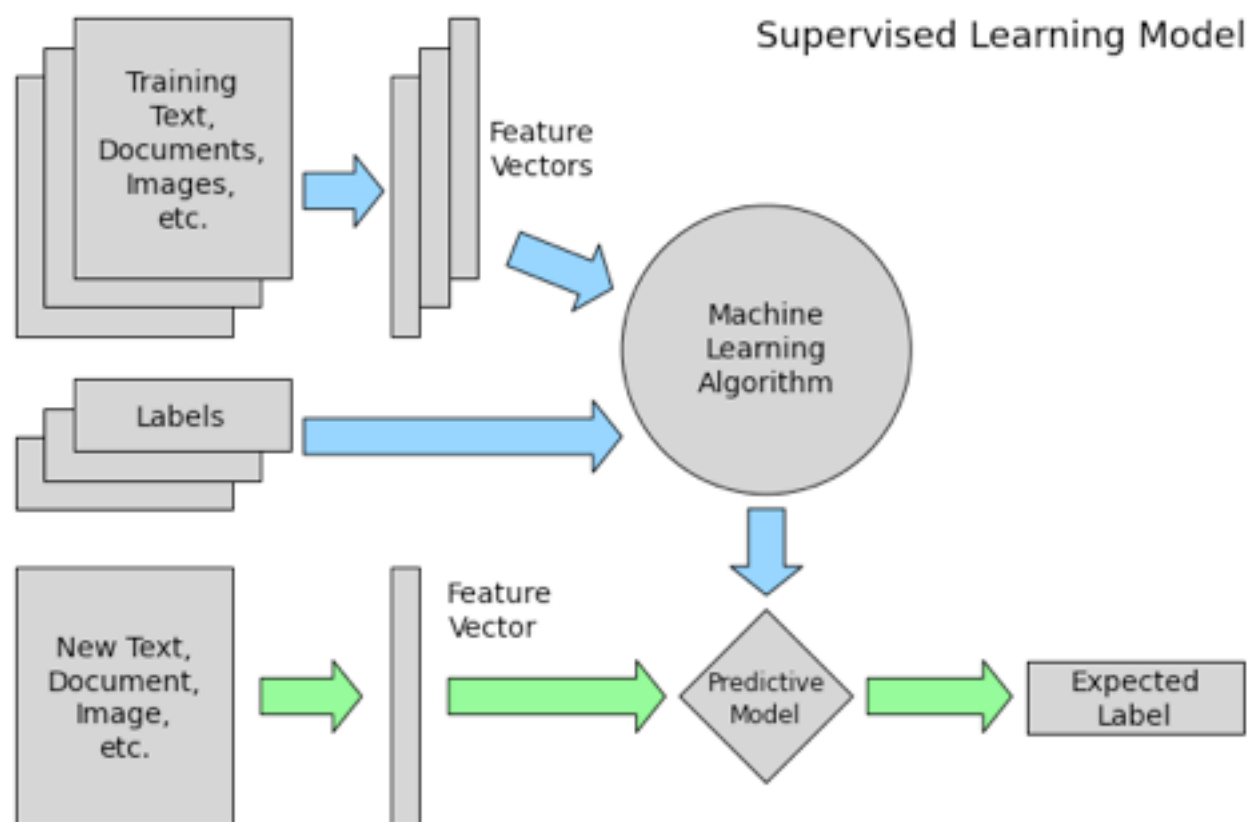
---

- Scikit-learn overview
- Model Distribution
- Data flow
- RabbitMQ
- Demo
- Scalability
- Other considerations

Finally I'll cover some additional things to consider when using scikit learn models in a realtime production environment.

# Supervised Learning

---



Thursday, March 20, 14

12

The demo in this talk will use a supervised learning algorithm. So let's start off, by defining what the training process looks like for a supervised model.

- 1) You start off with some input that may or may not be numerical. For example you might have text documents as input. This is called your training set.
- 2) You also have labels for each piece of training data.
- 3) You vectorize your training data (which means converting it to numerical values).
- 4) Then you train your machine learning algorithm using your vectorized training data, and the labels as input. This is often referred to as fitting your model.
- 5) At this point you have a model that can take a new piece of unlabeled data, and predict the label.
- 6) So again you need to vectorize your new data point.
- 7) Then you input it into your trained algorithm.
- 8) Your algorithm will spit out a predicted label for this new data point.

There is another class of machine learning algorithm called unsupervised learning. In this case your data won't be labeled. Tonight we'll be concentrating on supervised learning.

# 38 top wikipedias

---

Arabic	العربية	Japanese	日本語
Bulgarian	Български	Kazakh	Қазақша
Catalan	Català	Korean	한국어
Czech	Čeština	Lithuanian	Lietuvių
Danish	Dansk	Malay	Bahasa Melayu
German	Deutsch	Dutch	Nederlands
English	English	Norwegian (Bokmål)	Norsk (Bokmål)
Spanish	Español	Polish	Polski
Estonian	Eesti	Portuguese	Português
Basque	Euskara	Romanian	Română
Persian	فارسی	Russian	Русский
Finnish	Suomi	Slovak	Slovenčina
French	Français	Slovenian	Slovenščina
Hebrew	עברית	Serbian	Српски / Srpski
Hindi	हिन्दी	Swedish	Svenska
Croatian	Hrvatski	Turkish	Türkçe
Hungarian	Magyar	Ukrainian	Українська
Indonesian	Bahasa Indonesia	Vietnamese	Tiếng Việt
Italian	Italiano	Waray-Waray	Winaray

In this talk, I'm going to demonstrate one of the first models I created. A model that predicts the language of input text. To create this model, I used 38 of the top Wikipedias based on number of articles. I dumped several of the most popular articles from each of these wikipedias.

# The model

---

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.svm import LinearSVC

vect = TfidfVectorizer(analyzer='char', ngram_range=(2, 3), norm='l2', use_idf=True)
clf = LinearSVC(loss='l2', C=1, dual=True)
text_clf = Pipeline([
    ('vect', vect),
    ('clf', clf),
])
model = text_clf.fit(X_train, y_train)
```

---

Thursday, March 20, 14

14

So to start off I converted the wiki markup to plain text. I trained a LinearSVC (Support Vector Classifier) model using a n-gram approach I read worked well for language classification. This approach involves counting all combinations of **n** character sequences in your dataset. I tested the model and I was seeing ~99% accuracy.

**TODO: Visualization of n-grams. Describe cross validation**

**NEXT SLIDE!!!**

# The model

---

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.svm import LinearSVC

vect = TfidfVectorizer(analyzer='char', ngram_range=(2, 3), norm='l2', use_idf=True)
clf = LinearSVC(loss='l2', C=1, dual=True)
text_clf = Pipeline([
    ('vect', vect),
    ('clf', clf),
])
model = text_clf.fit(X_train, y_train)
```

---

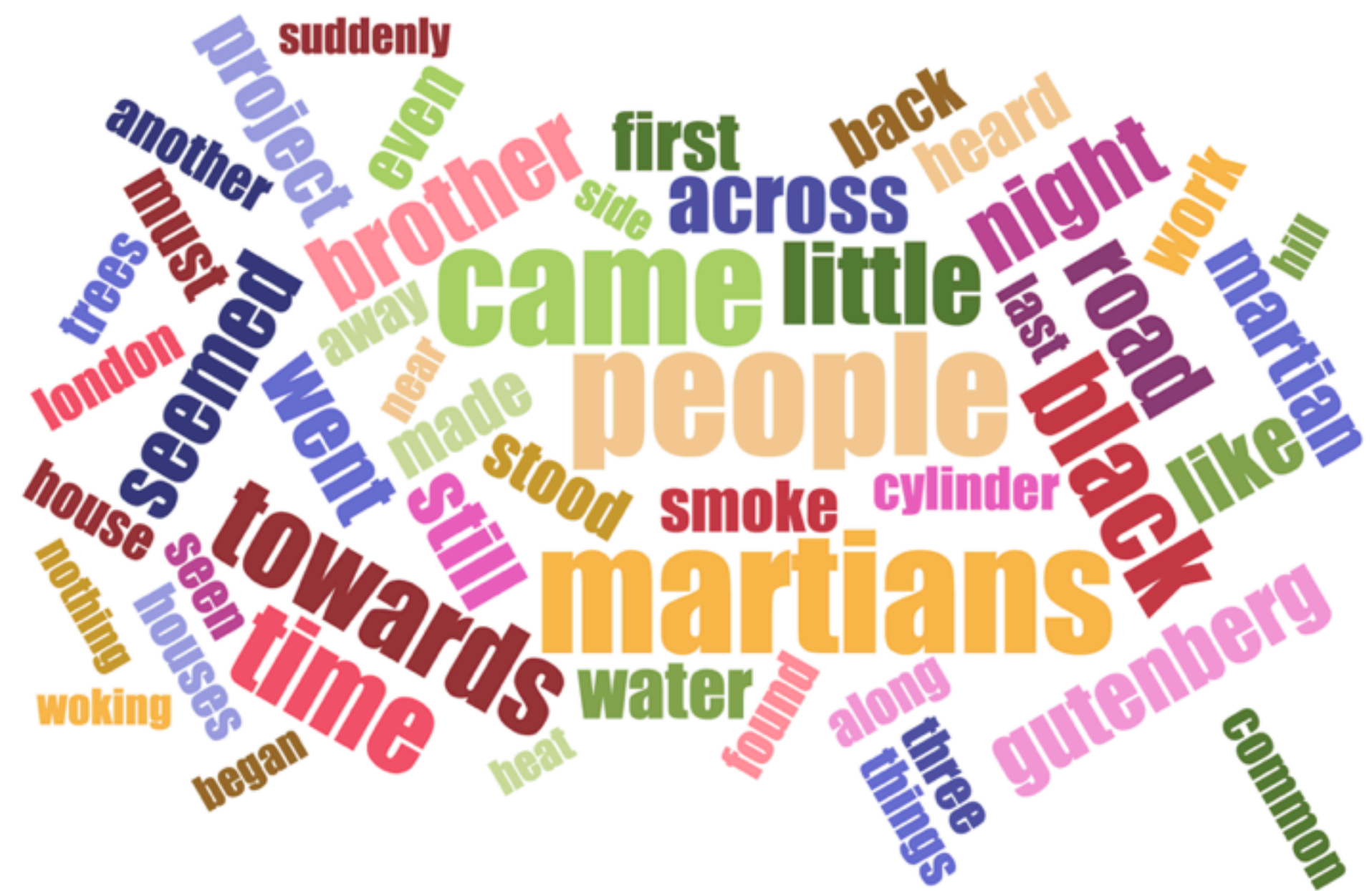
Thursday, March 20, 14

15

Here I've defined a pipeline combining a text feature extractor with a simple classifier. A pipeline is a utility used to build a composite classifier.

To convert the wikipedia articles into numerical form, I'm using a TfidfVectorizer. The vectorizer first counts the number of occurrences of each n-gram in each document to "vectorize the text." It then applies the TF-IDF (term frequency–inverse document frequency) algorithm. TF-IDF reflects how important a word is to a specific document in a collection of documents. The TF-IDF value increases based on the number of times a n-gram appears in the document, but is offset by the frequency of the n-gram in the rest of the documents. So let's take a look at a simple example.



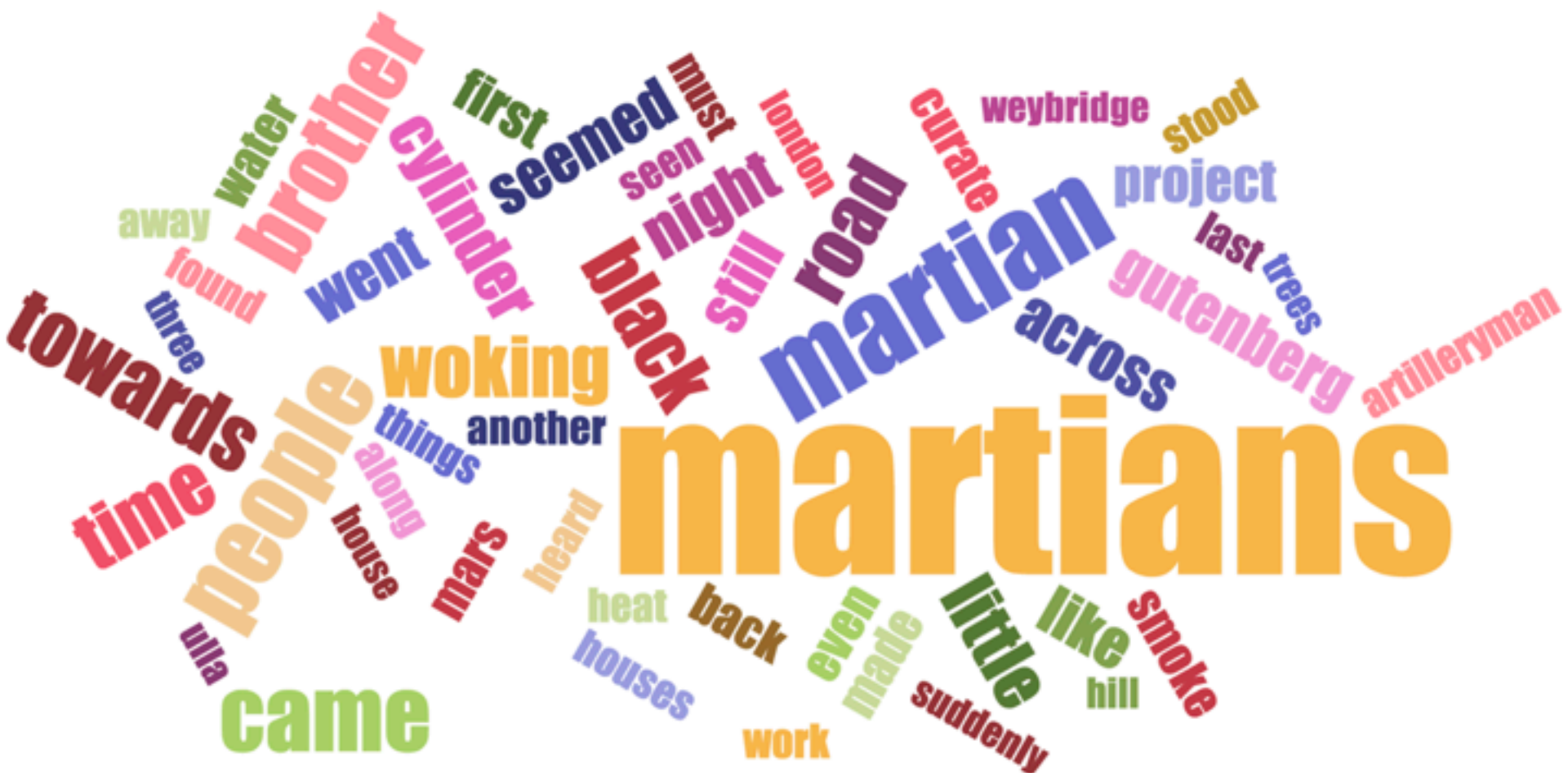


Thursday, March 20, 14

16

I downloaded 6 H.G Wells books from project gutenber. Here's what War of the Worlds looks like if we just look at raw word counts with no TF-IDF. The size of the word is based on the number of times a word shows up in the book.





Thursday, March 20, 14

17

And here is what it looks like with TF-IDF. Notice that the words “martian” and “martians” are comparatively bigger. They were weighted higher than more common words like “people” and “time” because these words are unique to this book. TF-IDF does have trade-offs though. On very large corpora it might not be computationally practical. In my case I used it because it increased the accuracy of my model by around 1%.

# Distributing the model

---



---

Thursday, March 20, 14

18

So now that you have this awesome model, "now what?"

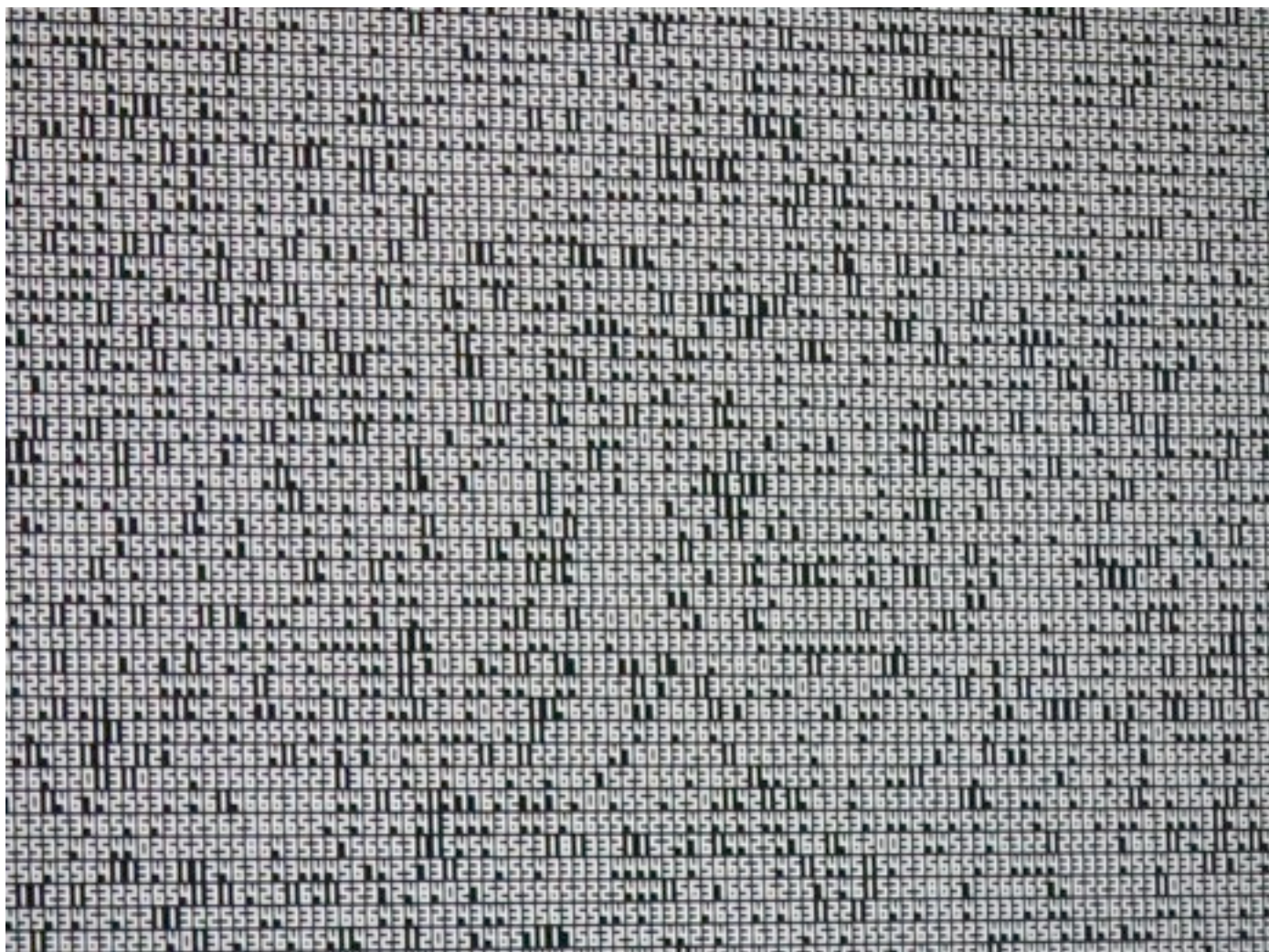
One of the first problems you'll want to solve is how to distribute your model? The recommended method for distributing your model is to use the built-in pickle module to serialize the model to disk and distribute it as part of your application. You can also store it in a database such as GridFS or Amazon S3. In the case of my model, it took up roughly 200MB in memory. This is pretty big, but easily storable on disk (and more importantly in memory).

One caveat to this approach is if you upgrade scikit-learn, your pickled model is not guaranteed to work. It's important that you keep detailed records of your training set, and your models training parameters if you want to be able to upgrade scikit-learn in the future. Take care when upgrading scikit-learn in production to make sure you're not going to break your application.



# Data input

---



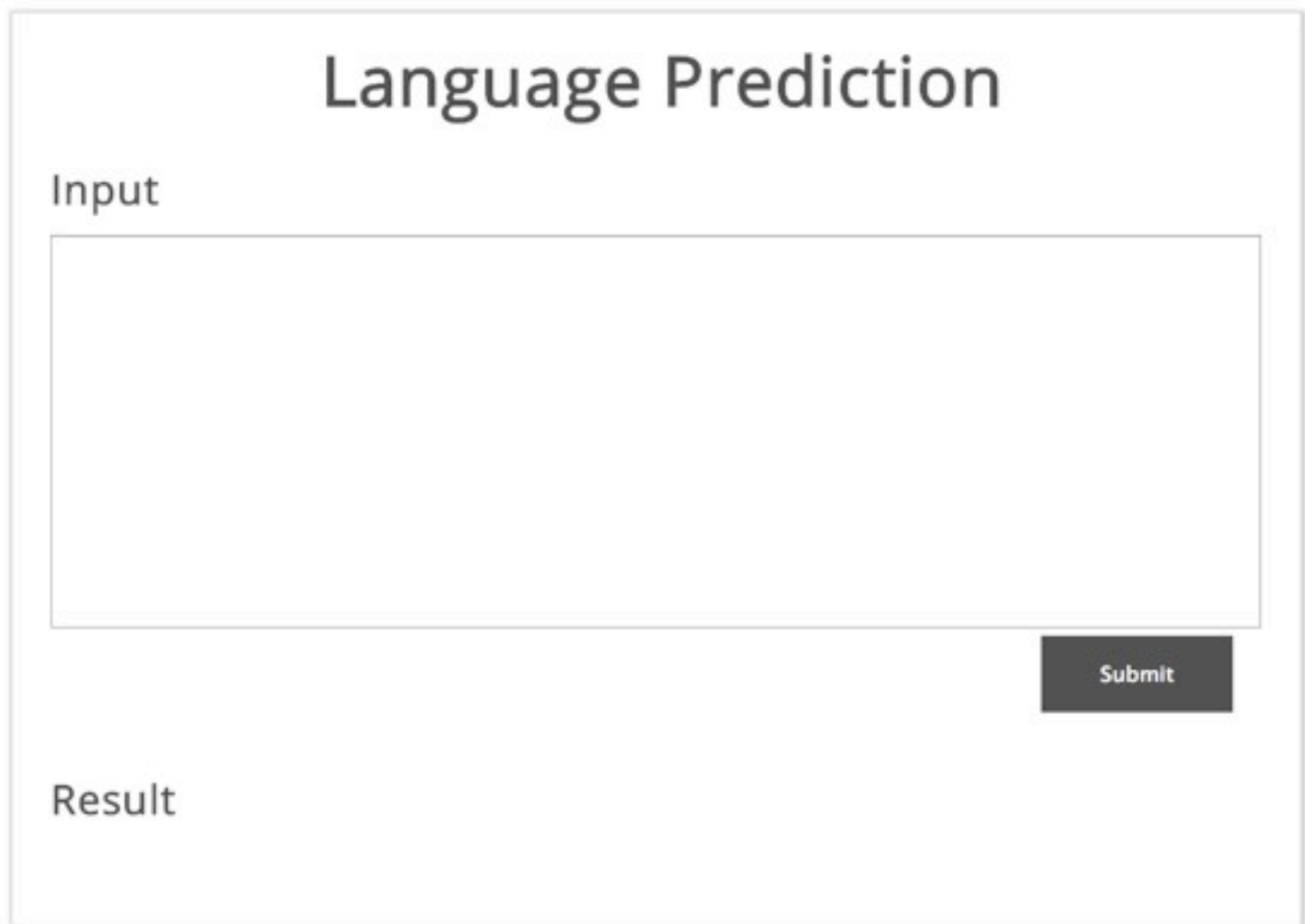
Thursday, March 20, 14

19

Next let's discuss how we're going to get data into our model. Your data could be coming from many types of sources, a web front-end, a DB trigger, etc.. In many cases, you can't easily control the rate of incoming data and you don't want to hold up the front-end or the database while you wait for a prediction to be made. In these cases, it's useful to be able to process your data asynchronously.

# The client

---



The image shows a web form titled "Language Prediction". It features a large text input area labeled "Input" and a "Submit" button. Below the input area is a label "Result".

Thursday, March 20, 14

20

In the example I'm giving today, we created a simple web front-end (similar to google translate) where a user can enter some text to be classified, and get a classification back. We don't want to hold up a thread or process in the client waiting on our classifier to do its thing. Rather the front-end sends the input to a REST API which will record the text input and return a `tracking_id` that the client can then use to get the result. There are several other ways we could design this, but for my application, this design works fine.



# Message loss

---



Thursday, March 20, 14

21

Decoupling the UI from the backend in this way solves one design issue. However another thing to consider is whether you can afford to lose messages. If all of your data needs to be processed you have 2 options. You either need to have a built in retry mechanism in the front end, or you need a persistent and durable queue to hold your messages in the backend.

# Enter RabbitMQ

---

Reliability

Flexible Routing

Clustering



# RabbitMQ

HA Queues

Many clients

---

Thursday, March 20, 14

22

Enter RabbitMQ. One of the many features provided by RabbitMQ is Highly Available Queues. By using RabbitMQ, you can ensure that every message is processed without needing to implement a fancy (and likely error prone) retry mechanism in your front-end.

# AMQP

---



---

Thursday, March 20, 14

23

RabbitMQ uses AMQP (Advanced Message Queuing Protocol) for all client communication. Using AMQP allows clients running on different platforms or written in different languages, to easily send messages to each other. From a high level, AMQP enables clients to publish messages, and other clients to consume those messages. It does all this without requiring you to roll your own protocol or library.



# Data processing

---



Thursday, March 20, 14

24

Once you hook your data input source into RabbitMQ and start publishing data, all you need to do is put your model in a persistent worker and start consuming input.



# The worker

---

```
class LanguagePredictorWorker(object):

    classifier, label_encoder = load_pickled_files(clf)
    def __init__(self):
        subscribe_to_queue()
        self.language_coll = get_db_collection()

    def process_event(self, body, message):
        text = body['text']
        _id = body['_id']
        language = self.predict_language_for_text(text)
        result = self.language_coll.update(
            {'_id': ObjectId(_id), 'text_input': text},
            {'$set': {'language': language}},
        )
        message.ack()

    def predict_language_for_text(self, text):
        lang_vector = self.classifier.predict([text])
        lang_labels = self.label_encoder.inverse_transform(lang_vector)
        return lang_labels[0]

worker = LanguagePredictorWorker()
worker.main()
```

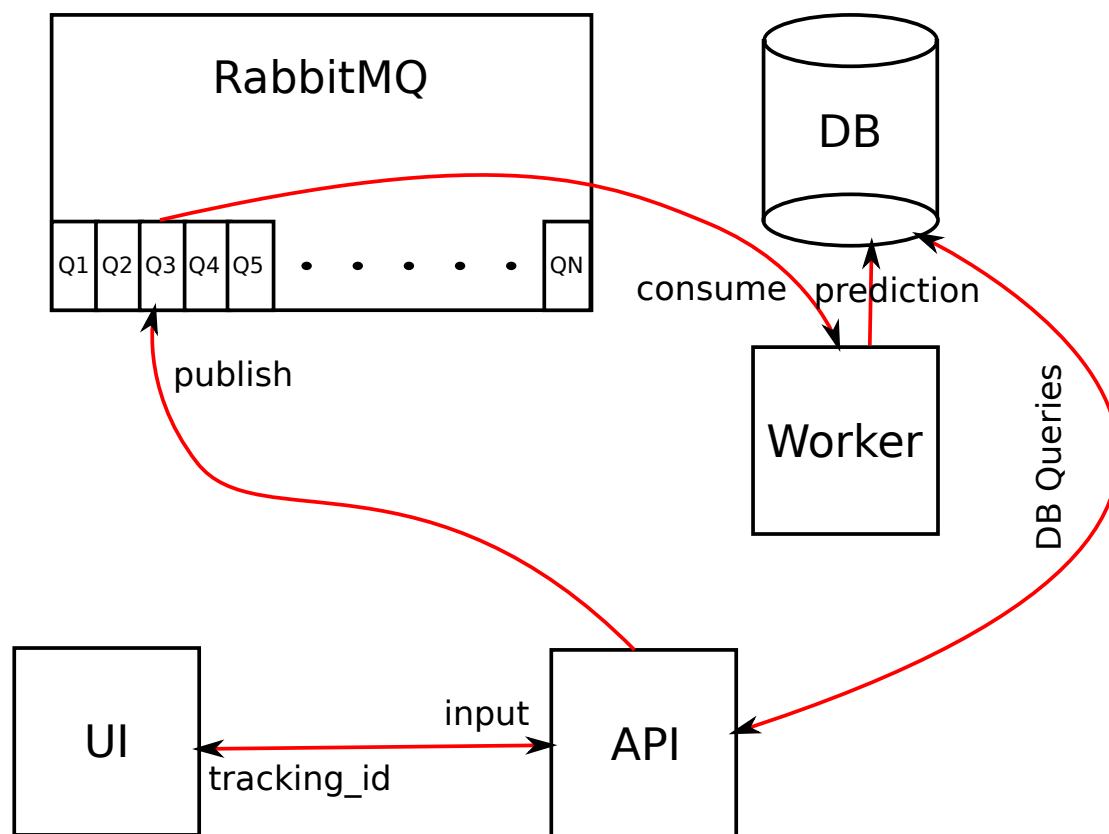
Thursday, March 20, 14

25

In the case of my language classification model, we implemented a simple worker that unpickles the classifier and subscribes to an input queue. It then runs an event loop (main) that pulls new messages as they become available and passes them to process\_event. Process event calls predict on our model and converts the numerical prediction to a human readable format. This prediction is then stored in our DB for the front-end to retrieve.

# The design

---



Thursday, March 20, 14

26

So that's basically it. Our design looks a little something like this:

The input comes from the UI where the user enters some text they wish to classify.

The UI hits a Flask REST API via a GET request.

The API stores the request in the DB.

The API sends a message to RabbitMQ with the text to classify and the tracking\_id for storing the resulting classification.

The API returns a json response to the UI with the tracking\_id.

The worker pulls the message off the queue in RabbitMQ.

The worker calls predict on the classifier with the text as input. The classifier returns a prediction. The worker updates the database with the result.

The UI displays the result.

There isn't one right way to design a solution like this. For example instead of polling you could utilize websockets. Then you could have the worker respond directly back to the client, possibly via RabbitMQ.

# Demo time!

Language Prediction

Input

Adolf II av Holstein

Adolf II av Holstein, född 1128, död 6 juli 1164 (stupad vid Verchen i Demmin), begravd i Minden, var greve av Holstein 1131–1164. Son till greve Adolf I av Holstein (död 1131) och Hildewa.

Biografi[redigera]

Adolf II efterträdde fadern i Schauenburg och Holstein-Wagrien. I flera år låg dock Wagrien under kontroll av Pribislaw av Mecklenburg. I tyska tronkriget stod Adolf på welfisk sida, och han var 1138–1142 förjagad av Albrekt Björnen då Adolf vägrade erkänna denne som sachsisk hertig. Som ny greve i Holstein och Stormarn insatte Albrekt då Heinrich von Badwide, men denne kunde Adolf senare driva ut med welfisk hjälp. 1143 återfick Adolf av Henrik Lejonet sitt tidigare grevskap mot en stor summa pengar, och detta år förenades landskapet Wagrien slutgiltigt med Holstein.

Adolf grundade 1134 Segeberg vars borg senare brändes ned av den av Adolf på flykt fördrivne Heinrich von Badwide. Borgen återuppbyggdes och blev Adolfs viktigaste stödjepunkt. 1143 grundade Adolf även Alt-Lübeck som förstördes 1147 av furst Niklot av obotriterna. Området överlämnades slutligen till Henrik Lejonet vilken 1158 nygrundade Lübeck.

Submit

Result

Swedish Svenska

Alright so let's see what this all looks like in action!

~~Danish & Norwegian~~ use the same character set, so I'll use them for my demo.  
Scots & English



# Scaling

---



Thursday, March 20, 14

28

Besides the basic design concerns I've already covered, there's a few more things worth mentioning.

The worst thing that can happen when you're processing data asynchronously is for your queue to backup. Backups will result in longer processing times, and if unbounded, you'll likely crash RabbitMQ. The easiest way to scale your workers is to start another instance. Using this strategy, processing should scale roughly linearly. In my experience, you can easily handle thousands of messages a second this way.

# Realtime vs batch

---



Thursday, March 20, 14

29

Another way to scale your worker is to convert it to processing requests in batches. Many of the algorithms in scikit-learn scale super-linearly when you pass multiple samples to the predict method. The downside of this is that you will no longer be able to process results in realtime. However, if you're restricted on resources (memory & cpu), this might be a worthwhile alternative.



# Monitoring

---



Thursday, March 20, 14

30

Keep an eye on your queue sizes, alert when they backup. Scale as needed (possibly automatically).

# Load

---



Thursday, March 20, 14

31

Understand your load requirements. Load test end-to-end to verify you can handle the expected load.



# Verify

---



Thursday, March 20, 14

32

Periodically re-verify your algorithm using new data. Build in a feedback loop so that you can collect new labeled samples to verify the performance

Version control your classifier. Keep detailed changelogs and performance metrics/characteristics.

This is a hard problem to solve in this case. How would we approach this? (How would you do this? Humans.) Mechanical turk?



# Thank you

---

API & Worker: Kelly O'Brien ([linkedin.com/in/kellyobie](https://www.linkedin.com/in/kellyobie))

UI: Matt Parke ([ordinaryrobot.com](https://ordinaryrobot.com))

Classifier: Michael Becker ([github.com/mdbecker](https://github.com/mdbecker))

Images: [Wikipedia](#)



---

Thursday, March 20, 14

33

I'd like to thank Kelly O'brien and Matt Parke for helping me with the front-end and back-end for the demo. Without them things would be a lot less exciting!

# My info

---

Tweet me [@beckerfuffle](https://twitter.com/beckerfuffle)

Find me at [beckerfuffle.com](https://beckerfuffle.com)

These slides and more @ [github.com/mdbecker](https://github.com/mdbecker)

You can find me online [@beckerfuffle](https://twitter.com/beckerfuffle) on Twitter. At [beckerfuffle.com](https://beckerfuffle.com), and I'm also [mdbecker](https://github.com/mdbecker) on github. I'll be posting the materials for this talk on my github.

**LAST SLIDE**

# Demo time!

Language Prediction

Input

Adolf II av Holstein

Adolf II av Holstein, född 1128, död 6 juli 1164 (stupad vid Verchen i Demmin), begravd i Minden, var greve av Holstein 1131–1164. Son till greve Adolf I av Holstein (död 1131) och Hildewa.  
Biografi[redigera]

Adolf II efterträdde fadern i Schauenburg och Holstein-Wagrien. I flera år låg dock Wagrien under kontroll av Pribislaw av Mecklenburg. I tyska tronkriget stod Adolf på welfisk sida, och han var 1138–1142 förjagad av Albrekt Björnen då Adolf vägrade erkänna denne som sachsisk hertig. Som ny greve i Holstein och Stormarn insatte Albrekt då Heinrich von Badwide, men denne kunde Adolf senare driva ut med welfisk hjälp. 1143 återfick Adolf av Henrik Lejonet sitt tidigare grevskap mot en stor summa pengar, och detta år förenades landskapet Wagrien slutgiltigt med Holstein. Adolf grundade 1134 Segeberg vars borg senare brändes ned av den av Adolf på flykt fördrivne Heinrich von Badwide. Borgen återuppbyggdes och blev Adolfs viktigaste stödjepunkt. 1143 grundade Adolf även Alt-Lübeck som förstördes 1147 av furst Niklot av obotriterna. Området överlämnades slutligen till Henrik Lejonet vilken 1158 nygrundade Lübeck.

Submit

Result

Swedish Svenska

Alright so let's see what this all looks like in action!

# Demo time!

Language Prediction

Input

Manhattanprosjektet (engelsk: Manhattan Project) var et forsknings- og utviklingsprogram, som under amerikansk ledelse og med deltakelse av Storbritannia og Canada førte til fremstillingen av de første atombombene under andre verdenskrig. Fra 1942 til 1946 ble prosjektet ledet av generalmajor Leslie Groves fra den amerikanske hærens ingeniørkorps (US Army Corps of Engineers). Hærens del av prosjektet fikk betegnelsen Manhattan District. Manhattan avløste etter hvert det offisielle kodenavnet Development of Substitute Materials som betegnelse for hele prosjektet. Underveis slukte Manhattanprosjektet også den tidligere britiske motparten, kalt Tube Alloys. Blant fysikerne som tok del i Manhattanprosjektet var Albert Einstein, Edward Teller og danske Niels Bohr. Manhattanprosjektet begynte i det små i 1938, men det vokste raskt og tilsammen beskjeftiget det over 130 000 personer og kostet nesten 2 milliarder dollar (tilsvarende ca. 150 milliarder i 2012[1]). Over 90 % av pengene gikk til byggingen av fabrikker og produksjonen av spaltbart materiale, mens under 10 % gikk til selve utviklingen av våpnene. Forskning og utvikling foregikk på mer enn 30 forskjellige steder rundt om i USA, Storbritannia og Canada, og noen av disse var hemmelige. Det ble bygget to typer atombomber under andre verdenskrig. En forholdsvis enkel uranbombe som benyttet uran-235, som er en relativt sjelden uranisotop som kun utgjør 0,7 % av naturlig

Submit

Result

Norwegian (Bokmål) Norsk (Bokmål)

Alright so let’s see what this all looks like in action!

# Demo time!

## Language Prediction

### Input

مره أخرى لأنه ينفذ أوامر القادة ولد (بول تيببتس) في عام 1915 م واسم والدته "اينولا جاي" اشتهر تيببتس بأنه أفضل طياري الجيش الأمريكي وقتها وكان تيببتس، الذي اشتهر بأنه أفضل طياري الجيش الأمريكي وقتها، وفي عام 1945 م ترقى إلى رتبة كولونيل وهي تعادل رتبة (عقيد)في سلاح الجو الأمريكي في 6 -8- 1945 قاد الطائرة الأمريكية التي ألقت القنبلة الذرية على هيروشيما في اليابان وهي قنبلة تحتوي على 60 كيلوغراما (130 رطلا) من مادة اليورانيوم - 235 هيروشيما هي مدينة في اليابان، تقع في جزيرة "هونشو"، وتشرف على "خليج هيروشيما". وكان تعداد سكانها عام 1945 م وأما الطائرة التي تحمل القنبلة فكانت قاذفة من (350,000 Little Boy نسمة وكان اسم الطائرة الكودي (بالشفرة السرية) هو (الولد الصغير النوع (بي 29) وكان بول تيببتس أول من اختبر هذه الطائرة وأطلق عليها اسم أمه "اينولا جاي". وتقرر تنفيذ مهمة قصف هيروشيما يوم 6-8- 1945 م حيث كان الطقس مواتيا لتنفيذ المهمة الخطيرة بعد أيام من السحب التي تجمعت فوق هيروشيما، فانطلق بول تيببتس بطائرته وهي تحمل القنبلة الذرية من قاعدة «ثورث فيلد» في جزيرة تينيان، غرب المحيط الهادئ، مصحوباً بطائرتين أخريين. وقبل القصف بساعة اكتشف نظام الإنذار المبكر الياباني دخول الطائرات للمجال الجوي الياباني فنبه السلطات في كبرى المدن بما فيها هيروشيما. لكن بول تيببتس كان في طريقه للمدينة المستهدفة يقود الطائرة القاذفة، واستطاع أن يتهرب من الدفاعات اليابانية ليصل مدينة هيروشيما وحوالي الساعة الثامنة صباحا تمكنت أجهزة الرادار في هيروشيما من تحديد الطائرات الأمريكية لكن المسؤولين العسكريين قرروا أن عددها الصغير لا يستدعي التصدي لها بطائرات مضادة على ضوء سياستهم الرامية لتوفير وقود الطائرات 0 وفي تمام الساعة الثامنة والربع قصف بول تيببتس القنبلة الرهيبة من طائرته «بي 29» على

Submit

### Result

العربية Arabic

Alright so let’s see what this all looks like in action!