AN INITIATIVE OF:

science, technology & innovation
Department: Science, Technology and Innovation
REPUBLIC OF SOUTH AFRICA

CSIR Touching lives through innovation | 80th anniversary

nicis
NATIONAL INTEGRATED CYBERINFRASTRUCTURE SYSTEM
DIRISA

Understanding Vectorization

DIRISA Datathon | Instructor: Kgaugelo Mmakola

Vectorization – Turning Text into Numbers

- **Problem:**
  💬 Our model can't understand text like "You've won a prize!"
  🧠 It only understands numbers.
- We use **TfidfVectorizer** to convert words into **numeric features**.
- **What does TF-IDF mean?**
  - **TF** = Term Frequency → How often a word appears in a message
  - **IDF** = Inverse Document Frequency → How rare the word is across all messages
- Together, this helps us **give higher scores to important, unique words** (like “win”, “claim”, “click”).

Why do we need TfidfVectorizer?

- Since the models **can't read text. The TfidfVectorizer;**
  - Turn words into numbers
  - Highlight important words
  - Create inputs the model can learn from

How It Works Step-by-Step

- Imagine these 3 SMS messages:
  - ["Win a free prize now", "Claim your prize", "Let's have lunch tomorrow."]
- First step is **Tokenization -** Break the text into words (called "tokens"):
  - ["win", "free", "prize", "now", "claim", "your", "let's", "have", "lunch", "tomorrow"]
- Step 2: **Term Frequency (TF) -** Count how often each word appears **in a message**.
  - “price” appears in message 1 and 2 = frequent

How It Works Step-by-Step

- **Step 3: Inverse Document Frequency (IDF)**

Check how **rare** a word is **across all messages**.

  - Words like "win", "claim", or "prize" are more unique to spam → higher weight
  - Words like "your“, “the” appear everywhere, → low importance

- **Step 4: TF × IDF**

Multiply the TF and IDF values to get the final weight:

  - Common in all messages = low value (e.g., "the", "you")
  - Common in message but rare overall = high value (⚠️ likely spam)

TfidfVectorizer is a class

- In Python, a **class** is like a **blueprint**.
- You use it to make an **object** with specific behaviors.
- For example, **TfidfVectorizer** has methods like:
  - .fit()
  - .transform()
  - .fit_transform()
- When you call those, the object knows what to do.

TfidfVectorizer Code simplification

```
Pipeline([
    ('clean', CustomTextCleaner()),          # Optional: remove emojis, URLs, etc.
    ('tfidf', TfidfVectorizer(max_features=5000)),
    ('model', LogisticRegression())
])
```

```
vectorizer = TfidfVectorizer()
X_tfidf = vectorizer.fit_transform(X_train)
```

Recap

Imagine you're summarizing a textbook. You highlight the **most important, unique** terms, not the ones that appear in every paragraph. That’s what **TfidfVectorizer** does: it picks out the most **meaningful and informative** words from all the messages.

AN INITIATIVE OF:

science, technology & innovation

Department:
Science, Technology and Innovation
REPUBLIC OF SOUTH AFRICA

CSIR | 80th anniversary

Touching lives through innovation

nicis

NATIONAL INTEGRATED
CYBERINFRASTRUCTURE SYSTEM

DIRISA

Model Evaluation – SMS Spam Detection

DIRISA Datathon | Instructor: Kgaugelo Mmakola

What Are We Trying to Do?

We trained different machine learning models to tell whether a message is **spam** or **ham** (not spam).
Now, we want to find out:

- 👉 **Which model is the best?**

Step 1: Evaluation Metrics

- To judge the models, we use **evaluation metrics** – numbers that tell us how well a model is performing.

Evaluation Model

1. **Accuracy -** Measures: “How often is the model right?”

- example: If 90 out of 100 messages were correctly predicted → Accuracy = 90%
- 🔥 **Good if spam and ham are equally common**
- ⚠️ **Warning**: If 90% of messages are ham, a model can be lazy and still get 90% accuracy just by guessing "ham" every time!
- Why is **Accuracy good only when spam and ham are equally common**?
- **Accuracy gives you a fair idea of how good the model is**

Evaluation Model

1. **F1 Score -** Measures: “How good is the model at catching spam *without making too many mistakes*?”
2. **Best for imbalanced data (like when spam is rare)**
3. **How to Check If Spam Is Rare? -** print(y.value_counts())

When to Use Which Metric?

Metric	Use When…
Accuracy	Spam and ham are about the same amount
F1 Score	Spam is rare (which is most real cases!)

When to Use Which Metric?

Situation	Best Metric	Why?
Spam = Ham (balanced)	Accuracy	Because it reflects both fairly
Spam < Ham (imbalanced)	F1 Score	Because accuracy hides mistakes

So we should use **F1 Score** to see if the model actually **catches spam correctly** and **doesn't mess up**

Final Takeaway

- Use **accuracy** if both spam and ham are equal
- Use **F1 score** if spam is **less common**
- In real life, **spam is rare**, so **F1 is the better choice**

Thank you!!

NATIONAL INTEGRATED CYBERINFRASTRUCTURE SYSTEM

DIRISA