

AASD 4004

Machine Learning - II

Applied AI Solutions Developer Program




Module 03

Text Classification


Vejeý Gandýer



Agenda



- Text Classification
- Classification Pipeline
- Acquire / Load dataset
- Pre-process dataset
- Split dataset
- Feature Extraction
- Train the classifier
- Evaluate the classifier
- Benchmark algorithms with feature extractors



Text Classification

What is it?



Text Classification

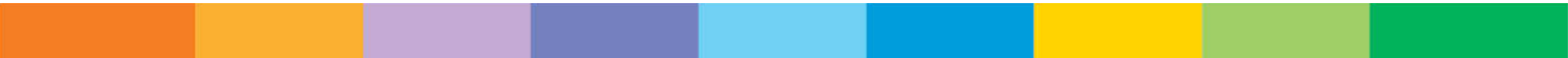
Assigning one or more categories to a given piece of text from a larger set of possible categories

Examples

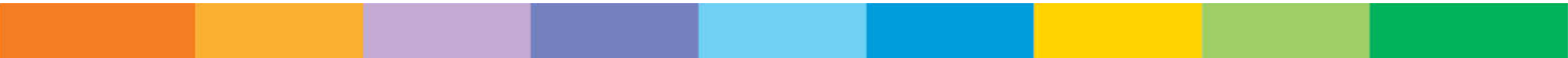
Spam Vs Non-spam emails (Two-class / Binary)

High-risk vs Medium-risk vs Low-risk loan provision (Multi-class)

Article tags classification(Multi-label)



Classification Pipeline



Classification Pipeline

Acquire / Load the dataset

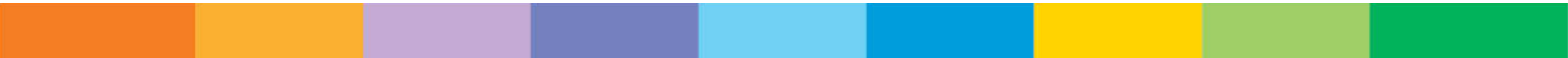
Pre-process the text

Split the dataset

Extract features from text

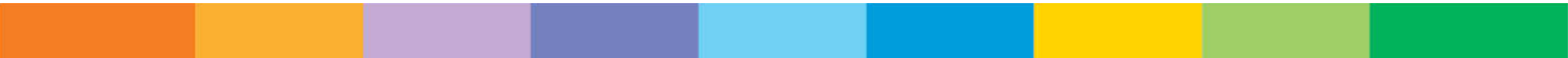
Train the classifier

Evaluate the classifier



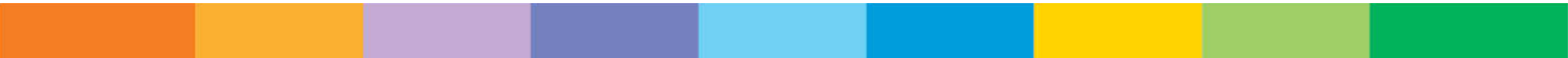
Classification Pipeline

Load dataset



Load dataset

Load the dataset



Classification Pipeline

Pre-process dataset



Pre-process dataset

Pre-process the text

```
#Step 2-3: Pre-process and Vectorize train and test data
vect = CountVectorizer(preprocessor=clean)
#clean is a custom defined function for pre-processing
X_train_dtm = vect.fit_transform(X_train)
X_test_dtm = vect.transform(X_test)
print(X_train_dtm.shape, X_test_dtm.shape)
```

Classification Pipeline

Split dataset



Split the dataset

Split the dataset

```
X = our_data.text  
y = our_data.relevance  
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
```

Classification Pipeline

Extract features



Extract Features

Word Embeddings - Word2Vec (Gensim)

```
data_path= "/your/folder/path"  
path_to_model = os.path.join(data_path, 'GoogleNews-vectors-negative300.bin')  
training_data_path = os.path.join(data_path, "sentiment_sentences.txt")  
#Load W2V model. This will take some time.  
w2v_model = KeyedVectors.load_word2vec_format(path_to_model, binary=True)  
print('done loading Word2Vec')
```

Extract Features

Creating own Embeddings

```
# Creating a feature vector by averaging all embeddings for all sentences
def embedding_feats(list_of_lists):
    DIMENSION = 300
    zero_vector = np.zeros(DIMENSION)
    feats = []
    for tokens in list_of_lists:
        feat_for_this = np.zeros(DIMENSION)
        count_for_this = 0
        for token in tokens:
            if token in w2v_model:
                feat_for_this += w2v_model[token]
                count_for_this += 1
        feats.append(feat_for_this/count_for_this)
    return feats

train_vectors = embedding_feats(texts_processed)
print(len(train_vectors))
```


Extract Features

Doc2Vec Embeddings

```
#Prepare training data in doc2vec format:
d2vtrain = [TaggedDocument((d),tags=[str(i)]) for i, d in enumerate(train_data)]
#Train a doc2vec model to learn tweet representations. Use only training data!!
model = Doc2Vec(vector_size=50, alpha=0.025, min_count=10, dm =1, epochs=100)
model.build_vocab(d2vtrain)
model.train(d2vtrain, total_examples=model.corpus_count, epochs=model.epochs)
model.save("d2v.model")
print("Model Saved")
```

Classification Pipeline

Train classifier



Train the classifier

Instantiate the classifier

```
#instantiate a Multinomial Naive Bayes classifier  
nb = MultinomialNB()
```

Train the classifier

```
#train the model  
nb.fit(X_train_dtm, y_train)
```

Classification Pipeline

Evaluate classifier



Evaluate the classifier

Evaluate the classifier

```
#make class predictions for test data  
y_pred_class = nb.predict(X_test_dtm)
```

Evaluate the classifier

Reasons for Poor Classifier

Reason 1 Since we extracted all possible features, we ended up in a large, sparse feature vector, where most features are too rare and end up being noise. A sparse feature set also makes training hard.

Reason 2 There are very few examples of relevant articles (~20%) compared to the non-relevant articles (~80%) in the dataset. This class imbalance makes the learning process skewed toward the non-relevant articles category, as there are very few examples of “relevant” articles.

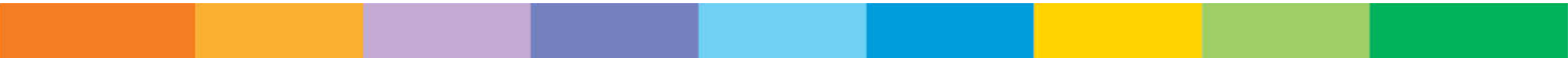
Reason 3 Perhaps we need a better learning algorithm.

Reason 4 Perhaps we need a better pre-processing and feature extraction mechanism.

Reason 5 Perhaps we should look to tuning the classifier’s parameters and hyperparameters.

Benchmarking algorithms

Multinomial Naïve Bayes, Logistic Regression, Support Vector Machines, Decision Trees



Multinomial Naïve Bayes

Multinomial Naïve Bayes

```
from sklearn.naive_bayes import MultinomialNB
vect = CountVectorizer(preprocessor=clean)
X_train_dtm = vect.fit_transform(X_train)
X_test_dtm = vect.transform(X_test)
nb = MultinomialNB()
nb.fit(X_train_dtm, y_train)
y_pred_class = nb.predict(X_test_dtm)
print("Accuracy: ", metrics.accuracy_score(y_test, y_pred_class))
```


Logistic Regression

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(class_weight="balanced")
logreg.fit(X_train_dtm, y_train)
y_pred_class = logreg.predict(X_test_dtm)
print("Accuracy: ", metrics.accuracy_score(y_test, y_pred_class))
```

Support Vector Machine

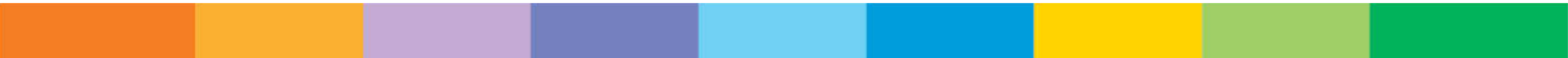
Support Vector Machine

```
from sklearn.svm import LinearSVC
vect = CountVectorizer(preprocessor=clean, max_features=1000)
X_train_dtm = vect.fit_transform(X_train)
X_test_dtm = vect.transform(X_test)
classifier = LinearSVC(class_weight='balanced')
classifier.fit(X_train_dtm, y_train)
y_pred_class = classifier.predict(X_test_dtm)
print("Accuracy: ", metrics.accuracy_score(y_test, y_pred_class))
```

Assignment: Benchmarking algorithms with algorithms, feature extractors

Algorithms: Multinomial Naïve Bayes, Logistic Regression, Support Vector Machines, Decision Trees

Feature Extractors: CountVectorizer, Word2Vec, Doc2Vec, Fastai



Benchmarking Algorithms and Feature Extractors

Create a benchmark analysis with different algorithms and feature extractors.

Algorithms: Multinomial Naïve Bayes, Logistic Regression, Support Vector Machines, Decision Trees

Feature Extractors: CountVectorizer, Word2Vec, Doc2Vec, Fastai

Benchmark all the possible above configurations and choose the best algorithm and feature extractor amongst all configurations

Further Reading

Scikit-learn documentation

<https://scikit-learn.org/>