Spam Classification

Author: Abel Stanley/13517068

1. Algoritma Machine Learning
    a. Neural Networks

Model Machine Learning Neural Network dibantu dengan Machine Learning API Keras.

Konfigurasi Umum Model NN:

1. Menggunakan 3 jenis callback:
    a. Tensorboard: untuk membantu visualisasi saat plotting graph training menjadi lebih deskriptif
    b. Early stopping: stop training ketika model accuracy dan loss tidak membaik setelah 5 epoch (patience = 5)
    c. Learning rate reduction: dengan ReduceLROnPlateau untuk membantu proses training saat kasus stagnasi terjadi

i. RNN - LSTM

Konfigurasi model:

1. Optimizer: RMSProp
    a. Menurut hasil penelitian saya, untuk RNN optimizer yang paling cocok adalah RMSProp karena Stochastic Gradient Descent mendapatkan advantage dari learning rate dan momentum dari setiap batch untuk optimasi bobot model dengan informasi dari loss function yang digunakan
2. Loss function: binary cross entropy
    a. Karena output dari model klasifikasi ini berupa biner: spam atau tidak spam
    b. Fungsi ini akan kalkulasi loss yang akan menghasilkan ouput biner
3. Metrics: Accuracy, Precision, dan Recall

Model arsitektur NN yang digunakan meliputi beberapa layers yaitu:

1. Embedding
    - Embedding adalah layer yang memetakan setiap kata menjadi vektor berdimensi N yang berisi nilai riil.
    - Dimensi yang digunakan adalah 100. Layer ini dipilih karena umumnya, dua jenis kata yang memiliki arti yang berdekatan itu mempunyai vektor yang mirip.
    - Layer ini menggunakan pre-trained layer dari GloVe.
2. RNN - LSTM (units = 128)

- Long short-term memory layer
- Aktivasi yang digunakan adalah tanh
-

3. Dropout (rate = 0.25)
    - Dropout layer akan secara random mengeset unit input menjadi 0 dengan frekuensi rate.
    - Dropout Layer digunakan agar setiap iterasi epoch seakan-akan sebuah *fresh experience* yang bertujuan untuk mengurangi kemungkinan overfitting.
    - Rate 0.25% digunakan karena pada tahap awal sebaiknya tidak terlalu banyak yang di-dropout agar tidak kehilangan banyak data.
4. Dense (units = 256, activation:"relu")
    - Fully connected NN layer biasa.
    - Units yang dipakai adalah 256.
    - ReLU digunakan untuk kasus ini karena benefit-nya yang besar dalam yaitu *sparsity* dan kemungkinan *vanishing gradient* yang lebih rendah. Relu juga secara komputasi lebih efisien dibandingkan fungsi aktivasi lain seperti Sigmoid sehingga cocok untuk layer dengan units yang banyak. Menurut hasil penelitian saya, NN dengan ReLU pada hidden layer akan menunjukkan konvergensi yang lebih baik daripada fungsi aktivasi lain.
5. Dropout (rate = 0.5)
    - Sama seperti dropout layer sebelumnya tetapi rate 0.5% digunakan karena dari hasil penelitian saya, di tahap akhir tidak masalah jika dropout rate yang digunakan lebih besar
6. Dense (units = 1, activation: "Sigmoid)
    - Output layer
    - Menggunakan Fungsi aktivasi sigmoid karena loss function yang digunakan adalah binary cross entropy. Jika nilai hasil output layer ini > 0.5 maka masuk ke class 1, else class 0.
    - ReLU tidak cocok di kasus loss function yang binary cross entropy.

ii. Fully Connected Network (ANN)

Konfigurasi model:

1. Optimizer: Adam
    a. Menurut hasil penelitian saya, Optimizer Adam adalah adaptive learning rate yang terbaik secara general
2. Loss function: binary cross entropy
    a. Karena output dari model klasifikasi ini berupa biner: spam atau tidak spam

    b. Fungsi ini akan kalkulasi loss yang akan menghasilkan ouput biner
3. Metrics: Accuracy, Precision, dan Recall

Model arsitektur NN yang digunakan meliputi beberapa layers yaitu:

1. Dense (units = 8270, activation:"relu")
    - Fully connected NN layer biasa.
    - Units yang dipakai adalah 8270. Dalam kasus ini saya bereksperimen dengan jumlah units yang sangat besar.
    - ReLU digunakan untuk kasus ini karena benefit-nya yang besar dalam yaitu *sparsity* dan kemungkinan *vanishing gradient* yang lebih rendah. Relu juga secara komputasi lebih efisien dibandingkan fungsi aktivasi lain seperti Sigmoid sehingga cocok untuk layer dengan units yang banyak. Menurut hasil penelitian saya, NN dengan ReLU pada hidden layer akan menunjukkan konvergensi yang lebih baik daripada fungsi aktivasi lain.
2. Dropout (rate = 0.25)
3. Dense (units = 4000, activation: "ReLU")
4. Dropout (rate = 0.3)
5. Dense (units = 1000, activation: "ReLU")
6. Dropout (rate = 0.4)
7. Dense (units = 400, activation: "ReLU")
8. Dropout (rate = 0.5)
9. Dense (units = 1, activation: "Sigmoid")

Catatan:

1. Dropout rate kecil di awal dan besar di akhir untuk mencegah data kebanyakan hilang di tahap awal
2. Dense layer memiliki units yang banyak di awal dan secara gradual akan berkurang semakin kedalam NN. Arsitektur ini mencontoh arsitektur NN yang umum di literatur penelitian.

b. Multinomial Naive Bayesian

Model dibuat dengan bantuan API ML sklearn.

Konfigurasi model:

1. Menggunakan Pipeline agar pemanggilan predict dan reuse lebih mudah. Pipeline terdiri dari:
    a. CountVectorizer
    b. TfidfTransformer
    c. MultinomialNB
2. Pencarian hyperparameter optimal dilakukan dengan bantuan GridSearchCV. Tuning parameter dilakukan dengan dataset validation. Parameter yang dicoba untuk di-tuning adalah:
    a. CountVectorizer: ngram_range
    b. TfIdfTransformer: norm, use_idf, sublinear_tf, smooth_idf
    c. MultinomialNB: fit_prior, alpha

c. Support Vector Machine

Model dibuat dengan bantuan API ML sklearn.

Konfigurasi model:

1. Menggunakan API Support Vector Classification (SVC)
2. Pencarian hyperparameter optimal dilakukan dengan bantuan GridSearchCV. Tuning parameter dilakukan dengan dataset validation. Parameter yang dicoba untuk di-tuning adalah:
    a. SVC: C, gamma, kernel

2. Final Hyperparameter Model Machine Learning

a. Neural Networks

i. RNN - LSTM:

Layer (type)	Output Shape	Param #
embedding_6 (Embedding)	(None, 100, 100)	685400
lstm_6 (LSTM)	(None, 128)	117248
dropout_16 (Dropout)	(None, 128)	0
dense_78 (Dense)	(None, 256)	33024
dropout_17 (Dropout)	(None, 256)	0
dense_79 (Dense)	(None, 1)	257

Total params: 835,929
Trainable params: 150,529
Non-trainable params: 685,400

ii. ANN:

Layer (type)	Output Shape	Param #
dense_27 (Dense)	(None, 8270)	835270
dropout_22 (Dropout)	(None, 8270)	0
dense_28 (Dense)	(None, 4000)	33084000
dropout_23 (Dropout)	(None, 4000)	0
dense_29 (Dense)	(None, 1000)	4001000

```
_________________________________________________________________
dropout_24 (Dropout)         (None, 1000)              0
_________________________________________________________________
dense_30 (Dense)             (None, 400)               400400
_________________________________________________________________
dropout_25 (Dropout)         (None, 400)               0
_________________________________________________________________
dense_31 (Dense)             (None, 1)                 401
=================================================================
Total params: 38,321,071
Trainable params: 38,321,071
Non-trainable params: 0
_________________________________________________________________
```

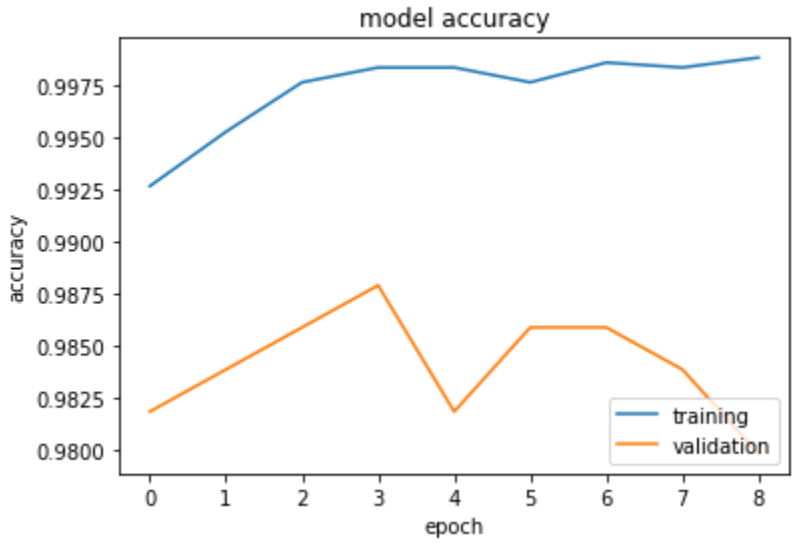
b. Multinomial Naive Bayes

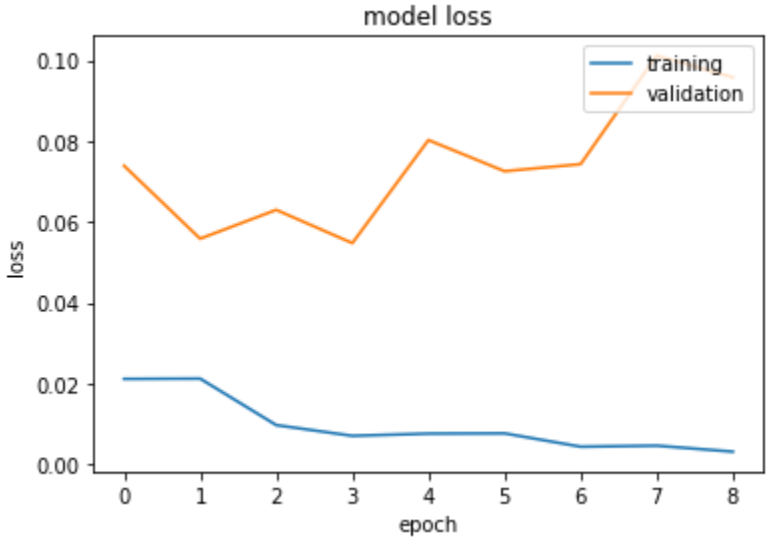
```
Pipeline(memory=None,
         steps=[('bow',
                 CountVectorizer(analyzer='word', binary=False,
                                 decode_error='strict',
                                 dtype=<class 'numpy.int64'>, encoding='utf-8',
                                 input='content', lowercase=True, max_df=1.0,
                                 max_features=None, min_df=1,
                                 ngram_range=(1, 2), preprocessor=None,
                                 stop_words=None, strip_accents=None,
                                 token_pattern='(?u)\\b\\w\\w+\\b',
                                 tokenizer=None, vocabulary=None)),
                ('tfidf',
                 TfidfTransformer(norm='l2', smooth_idf=True, sublinear_tf=True,
                                  use_idf=True)),
                ('multinb',
                 MultinomialNB(alpha=0.1, class_prior=None, fit_prior=True))],
         verbose=False)
```

c. SVM

```
SVC(C=10, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.1, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

3. Hasil Experiment
    a. Neural Networks
        i. RNN - LSTM:
            1. Training Graph





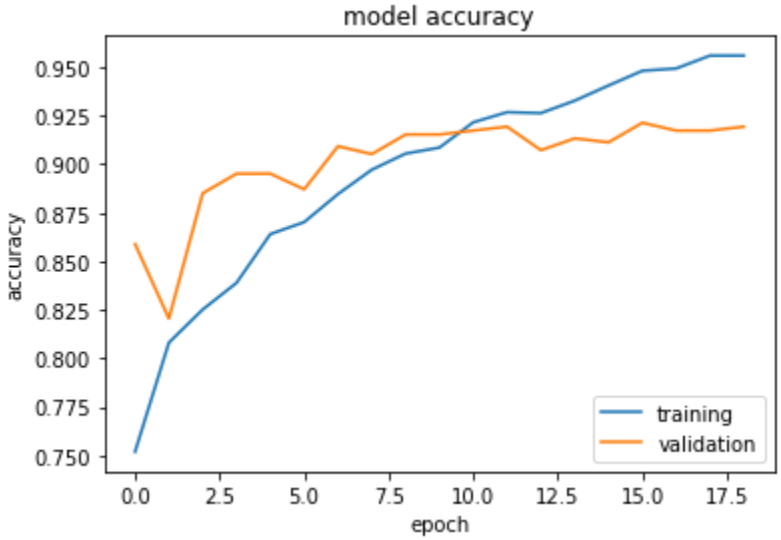
            2. Evaluation with Keras
               18/18 [==============================] - 0s 14ms/step - loss: 0.0921 - accuracy: 0.9854 - precision: 0.9797 - recall: 0.8715
               - Accuracy: 98.54%
               - Precision: 97.97%
               - Recall: 87.15%

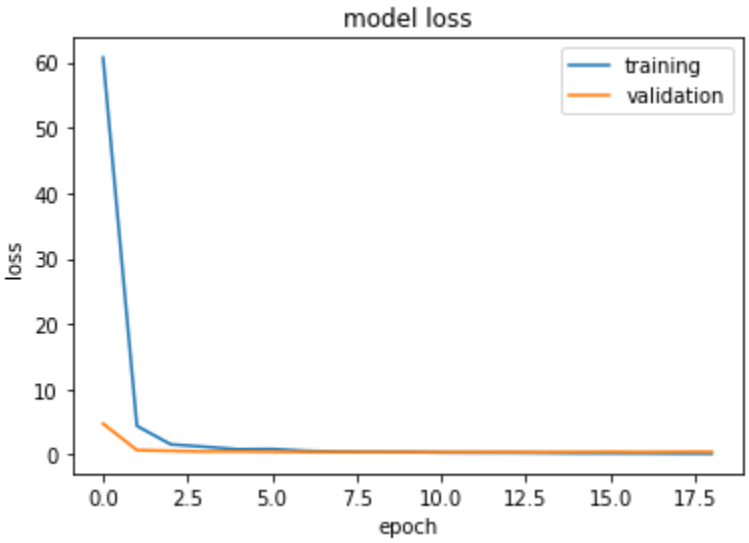
            3. Evaluation with Sklearn Report

	precision	recall	f1-score	support
0.0	0.99	1.00	0.99	477
1.0	0.98	0.90	0.94	72
accuracy			0.99	549
macro avg	0.99	0.95	0.97	549
weighted avg	0.99	0.99	0.99	549

ii. ANN:

1. Training Graph





2. Evaluation with Keras

18/18 [==============================] - 0s 18ms/step - loss: 0.3976 - accuracy: 0.9071 - precision_4: 0.6909 - recall_4: 0.5278

- Accuracy: 90.71%
- Precision: 69.09%
- Recall: 52.78%

3. Evaluation with Sklearn Report

	precision	recall	f1-score	support
0.0	0.93	0.96	0.95	477
1.0	0.69	0.53	0.60	72
accuracy			0.91	549
macro avg	0.81	0.75	0.77	549
weighted avg	0.90	0.91	0.90	549

b. Multinomial Naive Bayes:

i. Evaluation with Sklearn Report (DEFAULT PARAM)

	precision	recall	f1-score	support
ham	0.97	1.00	0.98	477
spam	1.00	0.76	0.87	72
accuracy			0.97	549
macro avg	0.98	0.88	0.92	549
weighted avg	0.97	0.97	0.97	549

ii. Evaluation with Sklearn Report (TUNED PARAM)

	precision	recall	f1-score	support
ham	0.99	1.00	0.99	477
spam	1.00	0.90	0.95	72
accuracy			0.99	549
macro avg	0.99	0.95	0.97	549
weighted avg	0.99	0.99	0.99	549

c. SVM Support Vector Classification:

i. Evaluation with Sklearn Report (DEFAULT PARAM)

SVM Accuracy Score -> 98.72495446265937

	precision	recall	f1-score	support
ham	0.99	1.00	0.99	477
spam	1.00	0.90	0.95	72
accuracy			0.99	549
macro avg	0.99	0.95	0.97	549
weighted avg	0.99	0.99	0.99	549

ii. Evaluation with Sklearn Report (TUNED PARAM)

SVM's accuracy score : 98.54280510018215

	precision	recall	f1-score	support
0	0.99	1.00	0.99	477
1	0.97	0.92	0.94	72
accuracy			0.99	549
macro avg	0.98	0.96	0.97	549
weighted avg	0.99	0.99	0.99	549

4. Kesimpulan

a. Berdasarkan hasil eksperimen diatas, urutan kinerja model machine learning diurutkan sebagai berikut:
   - **i. Accuracy**
     RNN LSTM == MultinomialNB(TUNED) == SVM(TUNED) == SVM(DEFAULT) > MultinomialNB (DEFAULT) > ANN
   - **ii. Precision**
     RNN LSTM == MultinomialNB(TUNED) == SVM(DEFAULT) > MultinomialNB(DEFAULT) == SVM(TUNED) >>> ANN
   - **iii. Recall**
     SVM(TUNED) > RNN LSTM == MultinomialNB(TUNED) == SVM(DEFAULT) > MultinomialNB(DEFAULT) >>> ANN
   - **iv. F1-score**
     RNN LSTM == MultinomialNB(TUNED) == SVM(TUNED) == SVM(DEFAULT) > MultinomialNB(DEFAULT) >>> ANN

b. Hypothesis:
   - i. RNN LSTM dan MultinomialNB(TUNED) overall memiliki performa yang terbaik.
   - ii. RNN LSTM sudah wajar untuk memiliki result yang baik karena memiliki beberapa fitur yang membedakan dirinya dengan algoritma ML tradisional seperti mengabstraksikan fitur.
   - iii. Dari grafik training dapat diterka:
     1. Training RNN LSTM sudah baik. Training loss dan validation loss tidak berbeda sebegitu jauh (0.10%) artinya tidak overfitting.
     2. Di kasus ANN, training loss dan validation loss adalah setara. Kasus ini menunjukkan tanda-tanda *underfitting*.
   - iv. MultinomialNB adalah model ML yang paling sederhana di eksperimen ini, namun memberikan hasil yang baik Umumnya Multinomial NB digunakan sebagai baseline bawah kinerja model, namun untuk kasus klasifikasi ternyata NB memiliki kinerja yang baik.
   - v. Performa SVM dinyatakan baik, namun ternyata sangat bergantung dengan fungsi kernel yang digunakan. Maka dari itu, digunakan Grid Search CV untuk mengetahui kernel function terbaik dalam kasus dataset spam. Didapatkan bahwa kernel function Linear jauh lebih baik kinerja-nya dibandingkan dengan Radial Basis Function, Sigmoid, dan Polynomial untuk kasus ini.
   - vi. Kinerja ANN buruk karena:

1. Banyak units yang terlalu besar relatif dari jumlah training data yang diberikan. Hal ini menyebabkan overfitting atau kasus exploding gradient
2. Banyak units juga terlalu besar dibandingkan dengan input size dan output size. Ada *rule of thumb* dari Jeff Heaton, penulis Introduction to Neural Networks in Java. Beliau mengatakan bahwa:
   'the optimal size of the hidden layer is usually between the size of the input and size of the output layers'.
   Maka arsitektur ANN yang dibuat melanggar *rule of thumb* tersebut

vii. Model ML yang dipakai dalam eksperimen ini merupakan model terakhir saat training selesai. Untuk perkembangan lebih lanjut kedepannya, dapat ditambahkan callback model checkpoint untuk menyimpan model dengan kinerja yang terbaik sesuai dengan metrik yang dipakai.