Project

January 8, 2022

1 Comparing support vector machine and logistic regression in spam detection

1.0.1 1 Introduction

A common issue with text message traffic is spam messages. Some of them are harmless marketing messages, while others contain phishing or malware that tries to damage or hijack user's system [1]. Machine learning methods can be used to detect spam messages and prevent user from opening the message and possibly loading harmful programs on the device.

This report compares the use of linear support vector machine (SVM) and logistic regression models with stochastic gradient descent (SGD) in spam message classification. To compare the performance of these two models a program was implemented using Python programming language and Pandas and Scikitlearn-libraries. The application domain for the message classifier could be in messaging and email programs to filter spam messages and in social media platforms to filter inappropriate comments and spam.

This project is based on report for course CS-C3240 Machine Learning D and it contains five sections in which the performance of two models is analyzed. This new report contains the code that was used to create and compare the machine learning models and more in depth explanation of the theory in the project. Section 2 discusses the problem formalization and the data used in the experiment. Section 3 explains the methods used to classify the data and the theory behind them. In section 4 the results are analyzed. At the end, section 5 summarizes the findings.

1.0.2 2 Problem formulation

The classification of messages can be formulated into a machine learning problem and in this case it is a classification problem. In this case the data points are messages. The feature characterizing the datapoint is the text in the message and in Python this is represented using String-datatype. The label that the model is trying to predict is represented using one of two categories: spam or ham. This type of classification is described as binary classification because the label can have two different values [2]. The used data contains labels for each datapoint meaning that supervised learning methods are used to classify messages [2]. Part of the used data can be seen by running the code below.

[1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split

# Load a csv-file containing messages into dataframe
df = pd.read_csv("spam.csv", sep=',')
```

```
# Because the dataset is imbalanced, both classes are divided into equal parts
spam = df[df['Category'] == 'spam'].head(740)    # 740 spam mails
ham = df[df['Category'] == 'ham'].head(740)      # 740 ham mails
frames = [spam, ham]
alldata =  pd.concat(frames).sample(frac=1)      # Merge both categories into␣
 ↪one dataframe containing 1480 messages and shuffle
display(alldata.head(5))

# Divide the dataset into 75% training sets and 25% test sets
msg_train, msg_test, label_train, label_test =␣
 ↪train_test_split(alldata['Message'], alldata['Category'], test_size=0.25)
print('Training set shape:', msg_train.shape)
print('Testing set shape:', msg_test.shape)
```

	Category	Message
4166	spam	Dear Voucher Holder, To claim this weeks offer...
65	spam	As a valued customer, I am pleased to advise y...
359	ham	I'm an actor. When i work, i work in the eveni...
551	ham	Imagine you finally get to sink into that bath...
366	ham	Well i know Z will take care of me. So no worr...

```
Training set shape: (1110,)
Testing set shape: (370,)
```

1.0.3 3 Methods

The data is downloaded from kaggle.com- web page [3], and it is stored in a CSV-file. Each row in the CSV-file represents one message. The first column contains the category or label to which the message has been classified being either spam or ham. The second column contains the text found in message.

The loaded dataset contains 5572 messages in total out of which 747 are labelled as spam messages. The dataset is therefore imbalanced and would require further engineering to take this into account and prevent from creating a model that would classify most messages as ham. To avoid this, an equal amount of spam and ham messages were picked from the dataset.

After resampling the original data set, the new dataset consists of 1480 messages in total, out of which 740 messages are labelled as spam messages and another 740 which are labelled as ham messages. The new dataset is shuffled and further divided into training sets containing 75% of the data, and a test sets containing 25% of the data using train_test_split- function. This means that 1110 messages were allocated as training data and 370 messages as testing data.

The model is trained using SGDClassifier-class in sklearn-library which implements the chosen linear model with stochastic gradient descent (SGD) learning. The model is selected by modifying the loss parameter: 'hinge' for linear support vector machine and 'log' for logistic regression. The performance of the chosen model is tested using the test set. The validation error is computed using grid-search cross validation because the dataset is small. By repeating the splitting of different subsets for training and validation the chance of an unlucky split is minimized [2]. The validation

was performed using GridSearchCV-class from sklearn-library. Also some hyperparameters were tuned using the GridSearchCV-class in order to improve the performance of the models. The suitable values for hyperparameters can be seen by running the cells that compute the grid-search cross validation.

Before the data could be used in creating the machine learning model, it is preprocessed using predefined countVectorizer and TfidfTransformer- functions. These functions convert the text in datapoints to matrices of token counts and then transform these matrices to normalized tf (term-frequency) or tf-idf (term-frequency times inverse document-frequency) representation. This enables to scale down the impact of commonly occurring tokens that are less informative than infrequently occurring tokens.

SVM and logistic regression are commonly used models for binary classification problems which is why these models were chosen for mail classification [4]. Other possible models that could have been used for this type of classification include naive Bayes classifier, k-nearest neighbors and decision trees [4]. The SGD was implemented to better perform handling the high dimentional dataset [5].

The loss function used to measure the linear hypothesis in SVM-model is hinge loss $L((x, y), h) := \max\{0, 1 - y \cdot h(x)\}$ [2]. The loss function used to measure the linear hypothesis in logistic regression is logistic loss $L((x, y), h) := \log(1 + \exp(-yh(x)))$ [2]. These functions are used by these models to assess the usefulness of a hypothesis map for classifying datapoints [2]. However, the results from these loss functions cannot be directly compared because they measure the loss in different scales. Therefore, training accuracies are also computed using the zero-one loss. The performance of the models is analyzed using the accuracy obtained from grid-search cross validation using zero-one loss.

[2]:
```
# Train and validate SVM

from sklearn.pipeline import Pipeline
from sklearn.linear_model import SGDClassifier
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer

print('Support vector machine:')
# Feature extraction and model selection using Pipeline object (setting the␣
 ↪loss function as 'hinge' chooses the SVM-model)
text_clf = Pipeline([('vect', CountVectorizer()),('tfidf',␣
 ↪TfidfTransformer()),('clf', SGDClassifier(loss='hinge',␣
 ↪penalty='l2',alpha=1e-3, random_state=42,max_iter=5, tol=None)),])

# Train the SVM-model
from sklearn.metrics import hinge_loss
svm_train_err = text_clf.fit(msg_train, label_train).score(msg_train,␣
 ↪label_train)
print('Training error:', svm_train_err)
svm_hinge_loss = hinge_loss(label_train, text_clf.decision_function(msg_train))
print('Hinge loss:', svm_hinge_loss)
```

```
Support vector machine:
```

```
Training error: 0.9945945945945946
Hinge loss: 0.039014559070511885
```

[3]:
```python
# Show the usable hyperparameters for the chosen model
print(text_clf.get_params().keys())
```

```
dict_keys(['memory', 'steps', 'verbose', 'vect', 'tfidf', 'clf',
'vect__analyzer', 'vect__binary', 'vect__decode_error', 'vect__dtype',
'vect__encoding', 'vect__input', 'vect__lowercase', 'vect__max_df',
'vect__max_features', 'vect__min_df', 'vect__ngram_range', 'vect__preprocessor',
'vect__stop_words', 'vect__strip_accents', 'vect__token_pattern',
'vect__tokenizer', 'vect__vocabulary', 'tfidf__norm', 'tfidf__smooth_idf',
'tfidf__sublinear_tf', 'tfidf__use_idf', 'clf__alpha', 'clf__average',
'clf__class_weight', 'clf__early_stopping', 'clf__epsilon', 'clf__eta0',
'clf__fit_intercept', 'clf__l1_ratio', 'clf__learning_rate', 'clf__loss',
'clf__max_iter', 'clf__n_iter_no_change', 'clf__n_jobs', 'clf__penalty',
'clf__power_t', 'clf__random_state', 'clf__shuffle', 'clf__tol',
'clf__validation_fraction', 'clf__verbose', 'clf__warm_start'])
```

[4]:
```python
# Hyperparameter tuning using GridSearchCV. Note: this computation might take␣
↪some time.

from sklearn.model_selection import GridSearchCV

# Find values for chosen hyperparameters
parameters = {
    'vect__max_df': (0.5, 0.75, 1.0),
    'tfidf__use_idf': (True, False),
    'clf__alpha': (0.00001, 0.000001),
    'clf__penalty': ('l2', 'elasticnet'),
    'clf__max_iter': (10, 50, 80),
}

grid_search = GridSearchCV(text_clf, parameters, n_jobs=-1)
grid_search.fit(msg_train, label_train)
svm_cv = grid_search.best_score_     # Best CV-score
print("Best score using SVM: %0.3f" % svm_cv)
print("Best parameter set using SVM:")
best_parameters = grid_search.best_estimator_.get_params()
for param_name in sorted(parameters.keys()):
    print("\t%s: %r" % (param_name, best_parameters[param_name]))
```

```
Best score using SVM: 0.959
Best parameter set using SVM:
        clf__alpha: 1e-05
        clf__max_iter: 80
        clf__penalty: 'l2'
        tfidf__use_idf: True
```

```
        vect__max_df: 0.5
```

[5]:
```
# Train and validate logistic regression
print('Logistic regression:')
# Feature extraction and model selection using Pipeline object (setting the␣
 ↪loss function as 'log' chooses the Logistic regression model)
text_clf2 = Pipeline([('vect', CountVectorizer()),('tfidf',␣
 ↪TfidfTransformer()),('clf', SGDClassifier(loss='log',␣
 ↪penalty='l2',alpha=1e-3, random_state=42,max_iter=5, tol=None)),])

# Train
from sklearn.metrics import log_loss
log_train_err = text_clf2.fit(msg_train, label_train).score(msg_train,␣
 ↪label_train)
print('Training error:', log_train_err)
logistic_loss = log_loss(label_train, text_clf2.predict_proba(msg_train))
print('Logistic loss:', logistic_loss)
```

```
Logistic regression:
Training error: 0.9864864864864865
Logistic loss: 0.25350041090341996
```

[6]:
```
# Find values for chosen hyperparameters
parameters2 = {
    'vect__max_df': (0.5, 0.75, 1.0),
    'tfidf__use_idf': (True, False),
    'clf__alpha': (0.00001, 0.000001),
    'clf__penalty': ('l2', 'elasticnet'),
    'clf__max_iter': (10, 50, 80),
}

grid_search2 = GridSearchCV(text_clf2, parameters2, n_jobs=-1)
grid_search2.fit(msg_train, label_train)
log_cv = grid_search2.best_score_     # Best CV-score
print("Best score using logistic regression: %0.3f" % log_cv)
print("Best parameter set using logistic regression:")
best_parameters2 = grid_search2.best_estimator_.get_params()
for param_name in sorted(parameters2.keys()):
    print("\t%s: %r" % (param_name, best_parameters2[param_name]))
```

```
Best score using logistic regression: 0.964
Best parameter set using logistic regression:
        clf__alpha: 1e-05
        clf__max_iter: 80
        clf__penalty: 'l2'
        tfidf__use_idf: True
        vect__max_df: 0.5
```

[7]:
```
# Test the better performing model (SVM)
predicted = text_clf.predict(msg_test)
test_err = np.mean(predicted == label_test)
print('Testing error:', test_err)

# Report
from sklearn import metrics
target_names = ['ham', 'spam']
print(metrics.classification_report(label_test, predicted,␣
 ↪target_names=target_names))
```

```
Testing error: 0.9351351351351351
              precision    recall  f1-score   support

         ham       0.92      0.96      0.94       197
        spam       0.96      0.90      0.93       173

    accuracy                           0.94       370
   macro avg       0.94      0.93      0.93       370
weighted avg       0.94      0.94      0.93       370
```

1.0.4 4 Results

First the SVM-model is trained using the hinge loss function and training data set. The resulting training error computed using hinge loss is 0.05 and accuracy using zero-one loss is 0.99. The validation error is computed using grid-search cross validation and zero-one loss. The validation error is 0.04 meaning that the accuracy is 0.96. The small drop in the validation accuracy compared to training accuracy can result from slight overfitting. The reason for using the zero-one loss is to be able to compare the two validation errors or accuracies.

Next the logistic regression model is trained using the logistic loss function. The training error computed using the logistic loss equals to 0.26 and accuracy using zero-one loss is 0.97. The validation error is again computed using grid-search cross validation and zero-one loss. The calculated validation error is 0.05 and accuracy 0.95. Also some hyperparameters were computed using the grid-search cross validation.

The results imply that the SVM-model performs slightly better for chosen classification task because the validation accuracy is higher. Now the SVM-model can be further tested using the unused test set allocated from the whole data to evaluate the performance. The testing error is 0.05 and accuracy is 0.95 when using the zero-one loss. All the results can be seen by running the code below. Note that the results might slightly vary depending of the order of the data.

Table 1. The chosen model (SVM) is highlighted due to having better validation accuracy. The training errors (losses) are computed using hinge loss and logistic loss and therefore should not be compared directly to each other. Note that since the SVM and logistic regression cross-validation scores are very similar, sometimes the score for logistic regression is higher depending on the ordering of the data. Usually the SVM CV-score is higher than the logistic regression CV-score.

[8]:
```
from tabulate import tabulate

# Compute a table with all the results
lst = [["Loss", svm_hinge_loss, logistic_loss, 1-svm_cv, 1-log_cv, 1-test_err],␣
 ↪["Accuracy", svm_train_err, log_train_err, "\033[1m" + str(svm_cv) +␣
 ↪"\033[0m", log_cv, test_err]]
table = tabulate(lst, headers=['', 'SVM Train', 'Log. reg. train', 'SVM g.s.␣
 ↪CV', 'Log. reg. g.s. CV', 'SVM test'], tablefmt='plain')
print(table)
```

```
            SVM Train    Log. reg. train    SVM g.s. CV    Log. reg. g.s. CV
SVM test
Loss        0.0390146           0.2535       0.0405405              0.036
0.0648649
Accuracy    0.994595            0.986486     0.959459
0.963964    0.935135
```

The accuracy result achieved with the test set is optimal implying that SVM is a good candidate model for message classification. Other research papers where experiments have been conducted on similar datasets also imply that SVM is one of the best models to use in such applications [6][7]. To improve the accuracy, a larger dataset could result in more accurate model. Next step in creating more accurate model in the future includes collecting more training data and further hyperparameter optimization to optimize the model performance.

1.0.5 5 Conclusion

This report analyzed the use of SVM and logistic regression models in message classification. The training accuracy was higher in SVM-model compared to logistic regression model. The better performing model was chosen using grid-search cross validation. The accuracy obtained using the cross validation revealed that SVM performed better compared to logistic regression for the chosen classification task because the validation accuracy was higher. The validation accuracy was slightly lower compared to the training accuracy which may imply overfitting.

The performance of SVM was further tested using the previously unused test set. The resulting accuracy using the test set concludes that the model did well in classifying messages with accuracy of 0.95. This kind of model could be used as message classifier because only small proportion of messages could end up misclassified. Some improvements could still be achieved by using larger dataset for training and by further optimizing the hyperparameters. Next steps in optimizing the model would require collecting larger data set for training.

1.0.6 References

1. Johnson, J., “Spam: share of global email traffic 2014-2020”, 2021. [online]. Available: https://www.statista.com/statistics/420391/spam-email-traffic-share/ [Accessed March 28, 2021]
2. Jung, A., “Machine learning: The Basics”, 2021. Available: https://github.com/alexjungaalto/MachineLearningTheBasics/blob/master/MLBasicsBook.pdf [Accessed March 28, 2021]

3. Team AI, “Spam Text Message Classification”. [online]. Available: https://www.kaggle.com/team-ai/spam-text-message-classification [Accessed March 28, 2021]

4. Brownlee, J., 2020. “4 Types of Classification Tasks in Machine Learning”, In Python Machine Learning. [online]. Available: https://machinelearningmastery.com/types-of-classification-in-machine-learning/ [Accessed March 28, 2021]

5. Wikipedia, “Stochastic gradient descent”. [online]. Available: https://en.wikipedia.org/wiki/Stochastic_gradient_descent

6. Almeida, T., Hidalgo, J. M. G., & Silva, T. P., 2013. ”Towards sms spam filtering: Results under a new dataset”. International Journal of Information Security Science, 2(1), 1-18. Available: http://www.ijiss.org/ijiss/index.php/ijiss/article/viewFile/34/pdf_4 [Accessed March 28, 2021]

7. Almeida, T. A., Hidalgo, J. M. G., & Yamakami, A., 2011. “Contributions to the study of SMS spam filtering: new collection and results”, In Proceedings of the 11th ACM symposium on Document engineering (pp. 259-262). Available: https://dl.acm.org/doi/pdf/10.1145/2034691.2034742 [Accessed March 28, 2021]

[ ]: