Θεωρία Αποφάσεων
Εργαστηριακή Άσκηση Χειμερινού Εξαμήνου 2021-2022

Τοτσίλα Διόνις
AM : 1067532

Δουρούκας Γεώργιος
AM : 1064872

Περιεχόμενα

Εισαγωγή

Σκοπός της εργασίας είναι να αναπτυχθεί ένα μοντέλο, το οποίο βασιζόμενο στις τεχνικές μηχανικής μάθησης, θα μπορεί να διακρίνει τα spam μηνύματα που λαμβάνει ένας χρήστης και να τον ενημερώνει για αυτά. Ως spam χαρακτηρίζεται ένα email το οποίο ο χρήστης δεν έχει επιλέξει αλλά ούτε και επιθυμεί να λαμβάνει και αποτελεί παγίδα που μπορεί να συναντήσει ένας χρήστης χρησιμοποιώντας την διαδικτυακή αλληλογραφία. Εφόσον τα παραπλανητικά μηνύματα δημιουργούν εμπόδιο στην ομαλή λειτουργία της αλληλογραφίας είναι αναγκαίο να βρεθούν λύσεις για την αποκατάσταση του προβλήματος. Οι διαδικασίες αυτές θα αναφερθούν αναλυτικά παρακάτω. Υπάρχουν διαφόρων ειδών τέτοια μηνύματα με τα πιο επικίνδυνα να χαρακτηρίζονται τα phising τα οποία έχουν στόχο να αλιεύσουν σημαντικά προσωπικά στοιχεία από το χρήστη (όπως στοιχεία τραπεζικού λογαριασμού) με χαρακτηριστικό ότι εμφανίζονται πλήρως πειστικά στο χρήστη και είναι ιδιαίτερα απειλητικά για αυτόν. Στην επίλυση όλων τον παραπάνω θα βοηθήσει καταλυτικά η μηχανική μάθηση η οποία είναι ιδιαίτερα πολύτιμη καθώς επιτρέπει στους υπολογιστές να χρησιμοποιούνται για την αυτοματοποίηση των διαδικασιών λήψης αποφάσεων.

1 Προ-επεξεργασία Δεδομένων, αναπαράσταση κειμένων στον διανυσματικό χώρο

Για να είναι εφικτό το training των μοντέλων θα πρέπει τα κείμενα να αναπαρασταθούν στον διανυσματικό χώρο. Προτού όμως γίνει αυτό θα πρέπει αρχικά να γίνει μια προ-επξεργασία του δοθέντος dataset[1] αλλά και να γίνει χρήση τεχνικών γλωσσικής τεχνολογίας οι οποίες θα κάνουν την διαδικασία ταξινόμησης πιο αποτελεσματική. Αρχικά μέσω της βιβλιοθήκης pandas[2] γίνεται ανάγνωση του dataset σε ένα dataframe, στο οποίο παρατηρείται ότι υπάρχουν 3 στήλες που δεν χρειάζονται και πρέπει να αποκοπούν. Στη συνέχεια δημιουργούνται μερικά διαγράμματα τα οποία παρουσιάζουν κάποιες γενικές πληροφορίες για την κατανομή του dataset, και τα χαρακτηριστικά που διακρίνουν την μια κατηγορία από την άλλη.

Παρατηρείται ότι το dataset είναι μη-ισορροπημένο και για να αποφευχθεί κάποιο overfit θα πρέπει να περιορίσουμε τα κείμενα της κατηγορίας ham, έτσι επιλέγοντα τυχαία κείμενα του ποσοστού της τάξεως 70% των αρχικών ham κειμένων

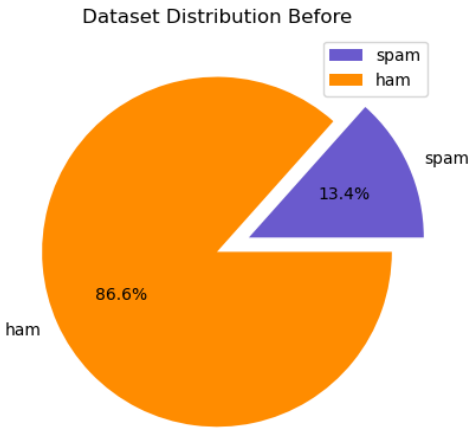

Figure 1: Κατανομή dataset πριν το undersampling

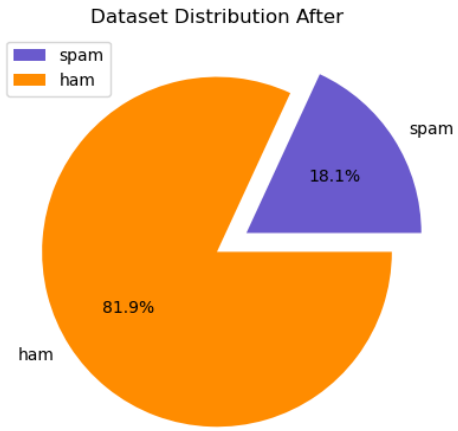

Figure 2: Κατανομή dataset μετά το undersampling

Παρατηρείτε επίσης ότι τα περισσότερα ham κείμενα περιέχουν απο 0 εως 10 λέξεις ενώ τα περισσότερα spam από 20 έως 30, άρα το μήκος του κειμένου μπορεί να δώσει πληροφορία σχετικά με το αν ένα μήνυμα θα μπορούσε να είναι κακόβουλο ή όχι.

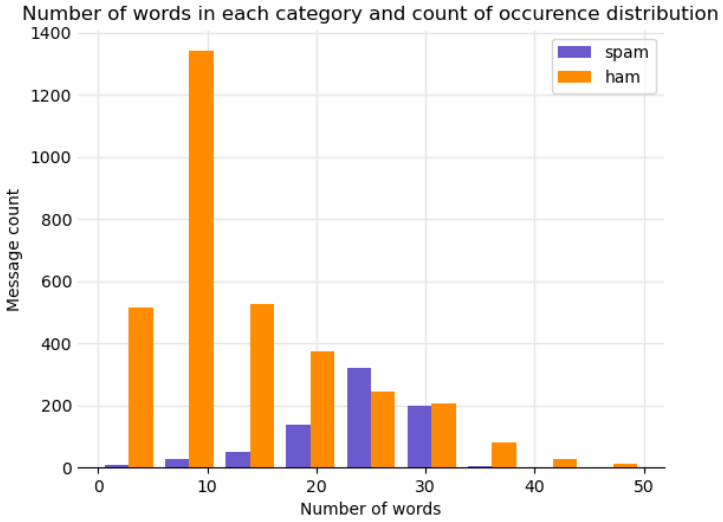

Figure 3: Κατανομή μήκους μηνύματος ανα κατηγορία

Είναι γνωστό πως τα περισσότερα spam μηνύματα έχουν άμεση σχέση με κάποιο χρηματικό ποσό, συνεπώς αφού παρατηρήθηκε ότι τα περισσότερα spam mail έχουν κάποιο χαρακτήρα συναλλάγματος όπως επίσης περιέχουν αριθμούς, κρίθηκε πως και αυτά τα 2 κριτήρια θα πρέπει προστεθούν στις παραμέτρους των διανυσμάτων.

1.0.1 Προεπεξεργασία κειμένων

Σειρά είχε η επεξεργασία των κειμένων, από κάθε κείμενο κρατήθηκαν μόνο οι λέξεις που αποτελούνται από αλφαριθμητικούς, καθώς τα σύμβολα και τα σημεία στίξης δεν μας δίνουν κάποια πληροφορία για το νοηματικό περιεχόμενο του κειμένου. Όλοι οι κεφαλαίοι χαρακτήρες μετατράπηκαν σε πεζούς και αφαιρέθηκαν τα stop words της αγγλικής (λέξεις που χρησιμοποιούνται συχνά σε κείμενα και κάνουν την διάκριση μη όμοιων νοηματικά προτάσεων).

1.0.2 Stemming

Γενικά για λόγους γραμματικής τα κείμενα έχουν διάφορες εκδοχές της ίδιας λέξης, όπως για παράδειγμά οι λέξεις organize, organizes και organizing, είναι προφανές ότι μια πρόταση που περιέχει τη λέξη organize και μια άλλη που περιέχει την λέξη organizes θα έχουν παρόμοιο νοηματικό περιεχόμενο. Στόχος λοιπόν του stemming είναι να "κανονικοποιήσει" παρόμοιες λέξεις δηλαδή να κάνει αναγωγή κλιτικών και παραγωγικών τύπων στο πρώτο κλιτικό. Για την επίτευξή του stemming χρησιμοποιήθηκε ο PorterStemmer της βιβλιοθήκης nltk[3].

1.0.3 TF-IDF

Αφου λοιπόν ολοκληρώθηκε η επεξεργασία των κειμένων, το κάθε κείμενο αναπαραστάθηκε ως διάνυσμα μέσω της μετρικής TF-IDF(Term Frequency - Inverse Document Frecuency). Στόχος της μετρικής αυτής είναι η ανάθεση tf*idf βάρους σε κάθε όρο του κάθε κειμένου και υπολογίζεται απο τον τύπο: $w_{ik} = tf_{ik} * \log(\frac{N}{dfi})$
Όπου:
tf_{ik}= η συχνότητα εμφάνισης του όρου i στο κείμενο k
df_k = Το πλήθος των κειμένων που περιέχουν τον i
N = το συνολικό πλήθος των κειμένων.
Για τον υπολογισμό του μητρώου tf-idf για την συλλογή κειμένων χρησιμοποιήθηκε η TfidfVectorizer της βιβλιοθήκης sklearn[4].

1.0.4 Κώδικας python3 για την προ-επεξεργασία των δεδομένων και την αναπαράσταση στον διανυσματικό χώρο

```
#!/usr/bin/env python
# coding: utf-8
import numpy as np
import pandas as pd

# NLP stuff
from sklearn.feature_extraction.text import TfidfVectorizer
from utils import processing, has_numbers, has_currency, sms_len, decorate_axis
import matplotlib.pyplot as plt

# set to True if you want to plot stuff
diag = False

# set to True if you want to show plots instead of saving them
show = False


# Read dataset into Pandas-Data Frame  beware of the encoding
df = pd.read_csv('data/dataset.csv', encoding="ISO-8859-1")

# since columns Unamed: # are empty we need to drop them and keep only the usefull data (v1, v2) tag and sms respectively
df.drop(['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], axis=1, inplace=True)

# Replace the column names so that they are more identifiable and easily accesed
df = df.rename(columns={"v1": "category", "v2": "text"})

# we will balance the dataset a little bit
ham = df[df['category'] == 'ham']
spam = df[df['category'] == 'spam']

# plot dataset distribution before undersampling
if diag == True:
    colors=['slateblue', 'darkorange']
    plt.pie([len(spam), len(ham)], labels = ['spam', 'ham'], colors = colors, explode = [0.2, 0],  autopct='%1.1f%%')
    plt.legend()
    plt.title('Dataset Distribution Before')
    if show:
        plt.show()
    else:
        plt.savefig("plots/data_dist_bef.png")
    plt.close()

# get 70% of ham demos
ham = ham.sample(frac=0.7)

# plot dataset distribution after undersampling
if diag == True:
    colors=['slateblue', 'darkorange']
    plt.pie([len(spam), len(ham)], labels = ['spam', 'ham'], colors = colors, explode = [0.2, 0],  autopct='%1.1f%%')
    plt.legend()
    plt.title('Dataset Distribution After')
    if show:
        plt.show()
    else:
        plt.savefig("plots/data_dist_aft.png")
    plt.close()

# create and plot a list that contains number of words in  messages and their count of occurence in the dataset
if diag == True:

    dataset_ham_count = ham['text'].str.split().str.len()
    dataset_ham_count.index = dataset_ham_count.index.astype(str) + ' words:'
    dataset_ham_count.sort_index(inplace=True)

    dataset_spam_count = spam['text'].str.split().str.len()
    dataset_spam_count.index = dataset_spam_count.index.astype(str) + ' words:'
    dataset_spam_count.sort_index(inplace=True)
    bins = np.linspace(0, 50, 10)
    fig = plt.figure()
```

```
    ax0 = fig.add_subplot()

    plt.title('Number of words in each category and count of occurence distribution')
    ax0.hist([dataset_spam_count, dataset_ham_count], bins, label=['spam', 'ham'], color=
    colors)
    decorate_axis(ax0)
    plt.legend(loc='upper right')
    plt.xlabel("Number of words")
    plt.ylabel("Message count")
    if show:
        plt.show()
    else:
        plt.savefig("plots/words_per_cat.png")
    plt.close()

# concatenate spam and ham demos
df = pd.concat([spam,ham], ignore_index=True)

# shuffle dataset
df = df.sample(frac=1).reset_index(drop=True)

# apply processing to every SMS and store results to a new column
df['processed'] = df.apply(processing, axis=1)

# add a new column that indicates if the message has numerical values
df['has_num'] = df.apply(has_numbers, axis=1)

# add a new column that indicates if the message has currency related stuff
df['has_money'] = df.apply(has_currency, axis=1)

# add a new column that contains the length of each message
df['length'] = df.apply(sms_len, axis=1).values.reshape(-1,1)

# the original message is no longer needed
df = df.drop(['text'], axis=1)

# tfidf vector representation of texts
Transformer = TfidfVectorizer(max_features=2500, max_df=0.8)
tfidf = Transformer.fit_transform(df.processed.values.astype('U'))
tfidfDF = pd.DataFrame(tfidf.todense())
tfidfDF.columns = sorted(Transformer.vocabulary_)

# store idf of corpus needed for generatoin of tf idf for new (unseen text)
idf = pd.DataFrame(Transformer.idf_).transpose()
idf = pd.DataFrame(data=idf.values, columns=tfidfDF.columns)

final = pd.concat([df, tfidfDF], axis=1)
final = final.drop(['processed'], axis=1)

# split and store train-test because every method will be trained and tested on the same data,
     so that the comparisons are as fair as possible
train = final.head(int(np.floor(len(final) * 0.8)))
test  = final.tail(int(np.ceil(len(final) - len(final) * 0.8)))

train.to_csv('data/train.csv')
test.to_csv('data/test.csv')
```

2 Σύγκριση Διάφορων μοντέλων Μηχανικής Μάθησης

Παρακάτω ακολουθεί ο σχολιασμός του κάθε μοντέλου, για τα περισσότερα μοντέλα έγινε η χρήση της βιβλιοθήκης sklearn, για την λογιστική παρεμβολή και τα νευρωνικά δίκτυα έγινε χρήση της βιβλιοθήκης pytorch[5], καθώς θέλαμε να δούμε μια πιο πρακτική ανάπτυξη κώδικα για μοντέλα μηχανικής μάθησης

2.1 Multinomial Naive Bayes

Ο Naive Bayes[6] είναι ένας από τους απλούς και πιο αποτελεσματικούς αλγόριθμους επιβλεπόμενης μάθησης που είναι γνωστός και ως πιθανοτικός ταξινομητής, καθώς προβλέπει με βάση τη πιθανότητα ενός αντικειμένου. Είναι κατάλληλος για δυαδική ταξινόμηση και έχει καλή απόδοση σε περιπτώσεις κατηγορικών μεταβλητών εισόδου σε σύγκριση με αριθμητικές μεταβλητές. Ακόμη είναι χρήσιμος για την πραγματοποίηση προβλέψεων και την πρόβλεψη δεδομένων με βάση τα ιστορικά αποτελέσματα. Ο ταξινομητής αυτός λειτουργεί με βάση την αρχή της υπό όρους πιθανότητας, όπως δίνεται από το θεώρημα bayes $P(\theta|\mathbf{D}) = P(\theta)\frac{P(\mathbf{D}|\theta)}{P(\mathbf{D})}$

Ο πολυωνυμικός αλγόριθμος Naive Bayes[7] είναι μια πιθανολογική μέθοδος εκμάθησης που χρησιμοποιείται κυρίως στην επεξεργασία φυσικής γλώσσας(NLP). Ο αλγόριθμος βασίζεται στο θεώρημα Bayes και προβλέπει την ετικέτα ενός κειμένου όπως ένα κομμάτι email ή ένα άρθρο εφημερίδας. Υπολογίζει την πιθανότητα κάθε ετικέτας για ένα δεδομένο δείγμα και στη συνέχεια δίνει την ετικέτα με την υψηλότερη πιθανότητα ως έξοδο. Πρόκειται για μια συλλογή πολλών αλγορίθμων όπου οι αλγόριθμοι μοιράζονται μια κοινή αρχή και αυτή είναι ότι κάθε χαρακτηριστικό που ταξινομείται δεν σχετίζεται με κανένα άλλο χαρακτηριστικό. Η παρουσία ή η απουσία ενός χαρακτηριστικού δεν επηρεάζει την παρουσία ή απουσία του άλλου χαρακτηριστικού και για αυτό το λόγο χρησιμοποιήθηκε ο Multinomial Naive Bayes, η επιτυχία του classifier ανερχεται στο 97%. Ακολουθεί το confusion matrix του classification.

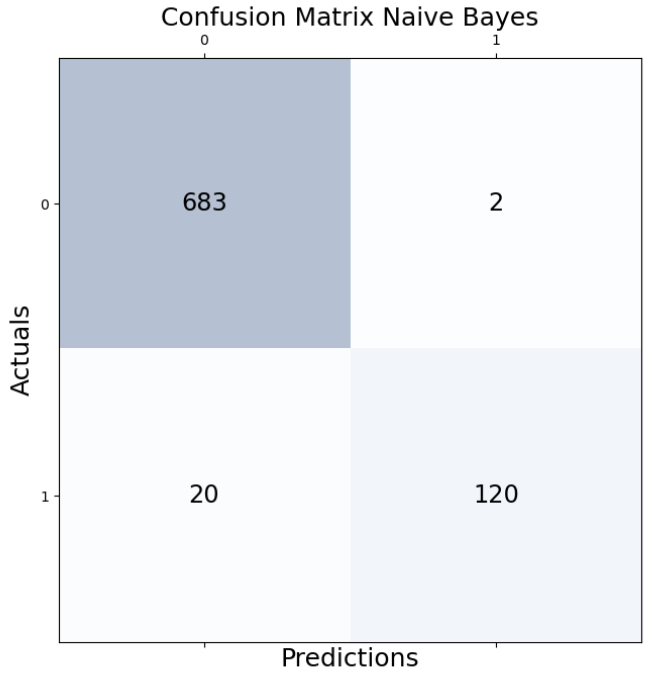

Figure 4: Polynomial Naive Bayes Classification Confusion Matrix

2.1.1 Κώδικας python3 Naive Bayes

```
from utils import read_data, plot_cf
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.naive_bayes import MultinomialNB
import pickle

# Read Train Test data
X_train, y_train = read_data('train')
X_test, y_test = read_data('test')

# init classifier and train
nb = MultinomialNB().fit(X_train,y_train)

# test
y_pred = nb.predict(X_test)

# Classification Evaluation
plot_cf(confusion_matrix(y_test,y_pred),"Naive Bayes")
print(classification_report(y_test,y_pred))
print(accuracy_score(y_test,y_pred))

# Store trained model
pickle.dump(nb, open('models/nb.pkl', 'wb'))
```

2.2 Decision Trees

Τα δέντρα αποφάσεων[8] είναι ένας τύπος εποπτευόμενης μηχανικής μάθησης όπου τα δεδομένα διαχωρίζονται συνεχώς σύμφωνα με μια συγκεκριμένη παράμετρο. Ο στόχος ενός δέντρου αποφάσεων είναι να δημιουργηθεί ένα μοντέλο εκπαίδευσης που μπορεί να χρησιμοποιηθεί για να προβλέψει την κλάση ή την τιμή της μεταβλητής στόχου μαθαίνοντας απλούς κανόνες απόφασης που συνάγονται από προηγούμενα δεδομένα. Αποτελείται από τον εσωτερικό κόμβο, τον κλάδο, και το κόμβο φύλλων. Κάθε εσωτερικός κόμβος αντιπροσωπεύει μια δοκιμή σε ένα χαρακτηριστικό. Κάθε κλάση αντιπροσωπεύει το αποτέλεσμα της δοκιμής και κάθε κόμβος φύλλων αντιπροσωπεύει μια ετικέτα κλάσης.
Βήμα 1: Εκκίνηση του δέντρου από τον ριζικό κόμβο, ο οποίος περιέχει το πλήρες σύνολο δεδομένων.
Βήμα 2: Εντοπίζεται το καλύτερο σύνολο δεδομένων χρησιμοποιώντας το Attribute Selection Measure(ASM).
Βήμα 3: Διαιρείται ο ριζικός κόμβος σε υποσύνολα που περιέχουν πιθανές τιμές.
Βήμα 4: Δημιουργία κόμβου αποφάσεων ο οποίος περιέχει το καλύτερο χαρακτηριστικό.
Βήμα 5: Δημιουργία αναδρομικών νέων δέντρων αποφάσεων χρησιμοποιώντας τα υποσύνολα του συνόλου δεδομένων που δημιουργήθηκαν στο προηγούμενο βήμα.

Η αποδοτικότητα των decision trees ανέρχεται στο 95%. Ακολουθεί το confusion matrix του classification.

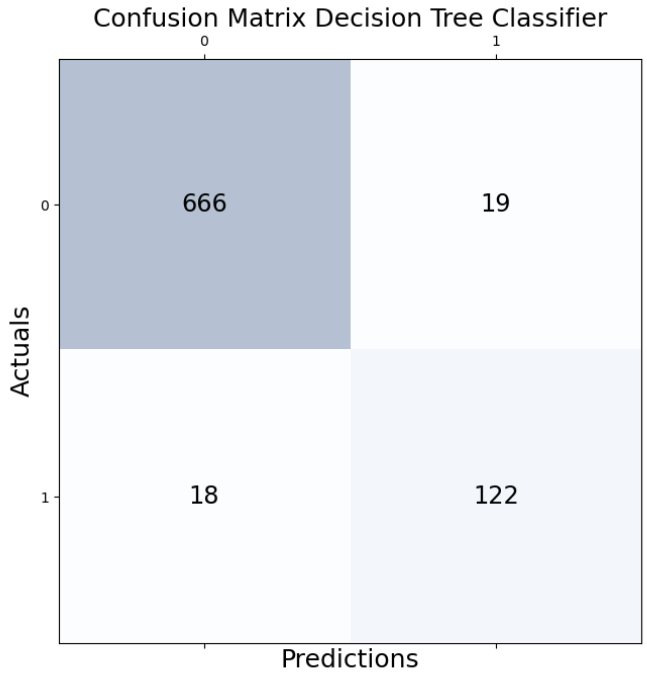

Figure 5: Decision Trees Classification Confusion Matrix

2.2.1 Κώδικας python3 Decision Trees

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from utils import read_data, plot_cf
import pickle

# Read Train Test data
X_train, y_train = read_data('train')
X_test, y_test = read_data('test')

# init classifier
dtc = DecisionTreeClassifier(min_samples_split=7, random_state=252)

# train
dtc.fit(X_train, y_train)

# test
y_pred = dtc.predict(X_test)

# Classification Evaluation
plot_cf(confusion_matrix(y_test,y_pred),"Decision Tree Classifier")
print(classification_report(y_test,y_pred))
print(accuracy_score(y_test,y_pred))

# Store trained model
pickle.dump(dtc, open('models/dtc.pkl', 'wb'))
```

2.3 Extra Trees

Τα Extra Trees[9] είναι ένας αλγόριθμος μηχανικής μάθησης που συνδυάζει τις προβλέψεις από πολλά δέντρα αποφάσεων. Συχνά μπορεί να επιτύχει εξίσου καλή ή καλύτερη απόδοση από τον αλγόριθμο τυχαίων δασών.Επίσης έχουν το χαρακτηριστικό ότι ενώ προσθέτουν τυχαία δέντρα στο δάσος τους παράλληλα εξακολουθούν να έχουν βελτιστοποίηση. Οι προβλέψεις γίνονται με τη λήψη μέσου όρου πρόβλεψης των δέντρων απόφασης σε περίπτωση παλινδρόμησης ή χρησιμοποιώντας την πλειοψηφία στην περίπτωση ταξινόμησης. Η αποδοτικότητα των Extra trees ανέρχεται στο 97.6%. Ακολουθεί το confusion matrix του classification.

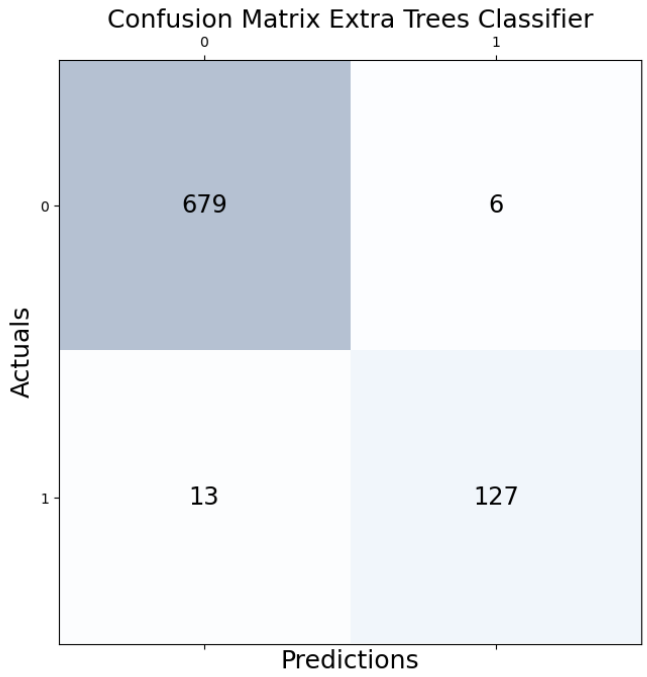

Figure 6: Extra Trees Classification Confusion Matrix

2.3.1 Κώδικας python3 Extra Trees

```python
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import pickle
from utils import read_data, plot_cf

# Read Train Test Data
X_train, y_train = read_data('train')
X_test, y_test = read_data('test')

# init classifier
etc = ExtraTreesClassifier(n_estimators=37, random_state=252)

# Train
etc.fit(X_train,y_train)

# Test
y_pred = etc.predict(X_test)


# Classification Evaluation
plot_cf(confusion_matrix(y_test,y_pred),"Extra Trees Classifier")
print(classification_report(y_test,y_pred))
print(accuracy_score(y_test,y_pred))

# Store trained model
pickle.dump(etc, open('models/etc.pkl', 'wb'))
```

2.4 Logistic Regression

Η λογιστική παλινδρόμηση[10] είναι ένας από τους πιο δημοφιλείς αλγόριθμους μηχανικής μάθησης ο οποίος εμπίπτει στην τεχνική εποπτευόμενης μάθησης και αποτελεί έναν πολύτιμο αλγόριθμο που χρησιμεύει για την εκτίμηση διακριτών τιμών όπως 0/1, ναι/όχι, αληθές/ψευδές. Χρησιμοποιείται για τα προβλήματα κατηγοριοποίησης και είναι ένας αλγόριθμος προγνωστικής ανάλυσης και βασίζεται στην έννοια της πιθανότητας. Λειτουργεί με μία σύνθετη συνάρτηση κόστους η οποία μπορεί να οριστεί ως λογιστική συνάρτηση.Sigmoid function είναι μια μαθηματική συνάρτηση που παίρνει οποιοδήποτε πραγματικό αριθμό και τον αντιστοιχίζει σε μια πιθανότητα μεταξύ 1 και 0. Η αποδοτικότητα της Λογιστικής Παρεμβολής ανέρχεται στο 97.2%. Ακολουθεί το confusion matrix του classification.

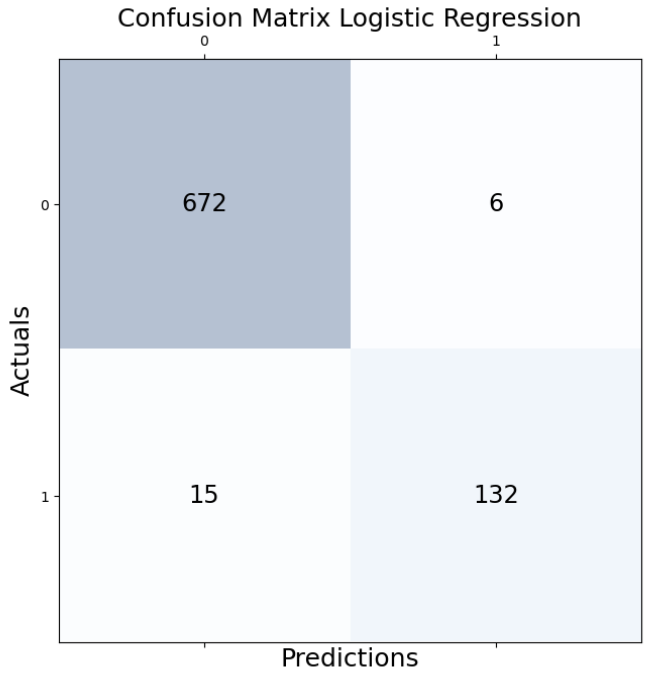

Figure 7: Logistic Regression Confusion Matrix

2.4.1 Κώδικας python3 Logistic Regression

```
import torch
from torch.utils.data import DataLoader
from utils import  CustomDataset, read_data, plot_cf
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
train = True

# Simple Logistic Regresion Module (sigmoid activated)
class LogisticRegression(torch.nn.Module):
    def __init__(self, input_dim, output_dim):
         super(LogisticRegression, self).__init__()
         self.linear = torch.nn.Linear(input_dim, output_dim)

    def forward(self, x):
        outputs = torch.sigmoid(self.linear(x))
        return outputs

# Read Test Data
X_train, y_train = read_data('train')
X_test, y_test = read_data('test')
train_dataset = CustomDataset(X_train, y_train)

# Train in batches
dataloader = DataLoader(train_dataset, batch_size=128, shuffle=True)
batches_per_epoch = X_train.shape[0]//128

# Hyper Parameters
epochs = 1000
learning_rate = 1e-2

# 1 binary output (binary classification)
output_dim = 1
input_dim = X_test.shape[1]
model = LogisticRegression(input_dim,output_dim)
criterion = torch.nn.BCELoss()
optimizer = torch.optim.AdamW(model.parameters(), lr=learning_rate)

if train:
    # train loop
    for epoch in range(epochs):
        running_loss = 0.0
        for i, data in enumerate(dataloader, 0):
            inputs, outputs = data
            batch_loss = 0.0
            # zero the parameter gradients
            optimizer.zero_grad()

            # forward + backward + optimize
            output = model(inputs.view(-1, input_dim))
            loss = criterion( output, outputs.view(-1, output.shape[1]))
            running_loss += loss.item()
            loss.backward()
            optimizer.step()
        train_mean_loss = running_loss/batches_per_epoch
        print("Epoch: ", epoch+1, "Loss: ", train_mean_loss)
    print('Finished Training')

    # store trained model
    torch.save(model.state_dict(),'models/logistic_regression.pt')
else:
    model.load_state_dict(torch.load('models/logistic_regression.pt'))
model.eval()
# test
# no derivative calculation so that the calculations are faster
with torch.no_grad():
    y_pred = []
    for i in range(X_test.shape[0]):
        # round to 0 or 1 so that class is determined
        output =torch.round(model(X_test[i])).detach().numpy()
        y_pred.append(output)
    plot_cf(confusion_matrix(y_test,y_pred),"Logistic Regression")
    print(confusion_matrix(y_test, y_pred))
    print(classification_report(y_test,y_pred))
    print(accuracy_score(y_test,y_pred))
```

2.5 Neural Network

Τα νευρωνικά δίκτυα[11] , γνωστά και ως τεχνητά νευρωνικά δίκτυα ή προσομοιωμένα νευρωνικά δίκτυα, αποτελούν υποσύνολο της μηχανικής μάθησης και βρίσκονται στην καρδιά των αλγορίθμων βαθιάς μάθησης. Είναι μια σειρά αλγορίθμων που προσπαθεί να αναγνωρίσει τις υποκείμενες σχέσεις σε ένα σύνολο δεδομένων μέσω μιας διαδικασίας που μιμείται τον τρόπο λειτουργίας του ανθρώπινου εγκεφάλου. Επίσης μπορούν να προσαρμοστούν στις μεταβαλλόμενες εισόδους και έτσι το δίκτυο παράγει το καλύτερο δυνατό αποτέλεσμα χωρίς να χρειάζεται επανασχεδιασμός των κριτηρίων εξόδου. Οι πληροφορίες τροφοδοτούνται στο επίπεδο εισόδου το οποίο τις μεταφέρει στο κρυφό επίπεδο. Οι διασυνδέσεις μεταξύ των δύο επιπέδων εκχωρούν βάρη σε κάθε είσοδο τυχαία. Η loss function σε ένα νευρωνικό δίκτυο ποσοτικοποιεί τη διαφορά μεταξύ του αναμενόμενου αποτελέσματος και του αποτελέσματος που παράγεται από το μοντέλο μηχανικής μάθησης. Συγκεκριμένα η binary cross entropy συγκρίνει κάθε μια από τις προβλεπόμενες πιθανότητες με την έξοδο κλάσης που μπορεί να είναι είτε 0 είτε 1. Στη συνέχεια υπολογίζει τις πιθανότητες με βάση την απόσταση από την αναμενόμενη τιμή. Αυτό δείχνει πόσο κοντά ή μακρυά είναι από την πραγματική τιμή. Ο Adam[12] είναι ένας προσαρμοστικός αλγόριθμος βελτιστοποίησης ρυθμού μάθησης που έχει σχεδιαστεί ειδικά για την εκπαίδευση, την κατάλληλη δηλαδή προσαρμογή των βαρέων που προαναφέραμε, σε βαθιά νευρωνικά δίκτυα. Συνδυάζει τις καλύτερες ιδιότητες των αλγορίθμων AdaGrad[13] και RMSProp για να παρέχει έναν αλγόριθμο βελτιστοποίησης που μπορεί να χειριστεί αραιές κλίσεις σε θορυβώδη προβλήματα. Η αποδοτικότητα των νευρωνικών δικτύων ανέρχεται στο 97.45%. Ακολουθεί το confusion matrix του classification.

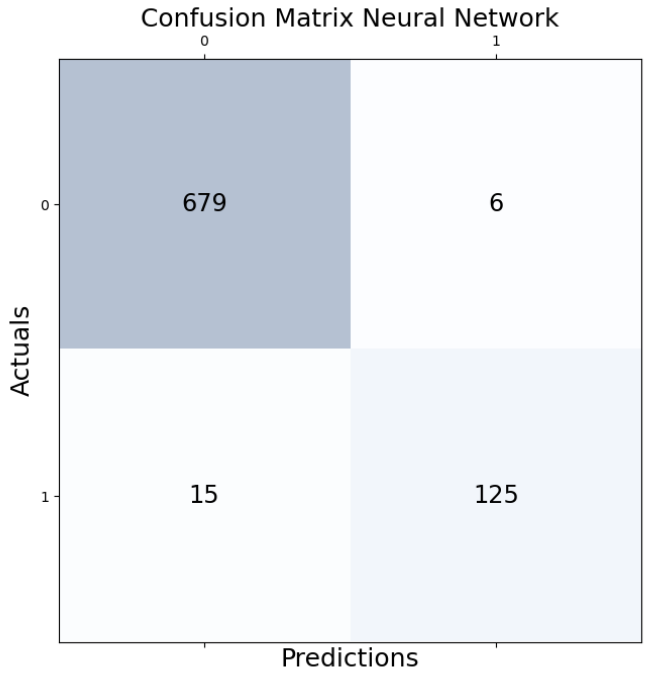

Figure 8: Neural Network Confusion Matrix

2.5.1 Κώδικας python3 Neural Network

```
import torch
from torch.utils.data import DataLoader
from utils import CustomDataset, read_data, plot_cf
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import torch.nn as nn
import torch.nn.functional as F

train = False
# Read Test Data
X_train, y_train = read_data('train')
X_test, y_test = read_data('test')
train_dataset = CustomDataset(X_train, y_train)
print(X_train.shape)
print(X_test.shape)

```

```
dataloader = DataLoader(train_dataset, batch_size=128, shuffle=True)


# Hyper Parameters
epochs = 500

# input_dim = tfidf vector length
input_dim = X_train.shape[1]

# 1 binary output (binary classification)
output_dim = 1
learning_rate = 1e-3
class BinaryClassification(nn.Module):
    def __init__(self, input_dim):
        super(BinaryClassification, self).__init__()
        self.layer_1 = nn.Linear(input_dim, 512)
        self.layer_2 = nn.Linear(512, 128)
        self.layer_3 = nn.Linear(128, 32)
        self.layer_out = nn.Linear(32, 1)
        self.relu = nn.ReLU()


    def forward(self, inputs):
        x = self.relu(self.layer_1(inputs))
        x = self.relu(self.layer_2(x))
        x = self.relu(self.layer_3(x))
        x = torch.sigmoid(self.layer_out(x))


        return x

model = BinaryClassification(input_dim)
criterion = torch.nn.BCELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
batches_per_epoch = X_train.shape[0]//128

if train:
    for epoch in range(epochs):
        running_loss = 0.0
        for i, data in enumerate(dataloader, 0):
            # get the inputs; data is a list of [inputs, output]
            inputs, outputs = data
            # print(inputs)
            batch_loss = 0.0
            # zero the parameter gradients
            optimizer.zero_grad()
            # forward + backward + optimize
            output = model(inputs.view(-1, input_dim))
            # print(output)
            loss = criterion( output, outputs.view(-1, output.shape[1]))
            running_loss += loss.item()
            loss.backward()
            optimizer.step()
            # scheduler.step(loss.item())
        # scheduler.step()
        train_mean_loss = running_loss/batches_per_epoch

        print("Epoch: ", epoch+1, "Loss: ", train_mean_loss)
    print('Finished Training')
    torch.save(model.state_dict(),'models/nn.pt')
else:
    model.load_state_dict(torch.load('models/nn.pt'))
with torch.no_grad():
    model.eval()
    y_pred = []
    for i in range(X_test.shape[0]):
        # round to 0 or 1 so that class is determined
        output =torch.round(model(X_test[i])).detach().numpy()
        y_pred.append(output)
    plot_cf(confusion_matrix(y_test,y_pred),"Neural Network")
    print(confusion_matrix(y_test, y_pred))
    print(classification_report(y_test,y_pred))
    print(accuracy_score(y_test,y_pred))
```

2.6 Συμπεράσματα - Εφαρμογή σε απλό διακοσμητή chat room

Classifier	tag	precision	recall	f1-score	support
Multinomial Naive Bayes	ham	0.97	1.0	0.98	685
Multinomial Naive Bayes	spam	0.98	0.86	0.92	140
Decision Trees	ham	0.97	0.97	0.97	685
Decision Trees	spam	0.87	0.87	0.87	140
Extra Trees	ham	0.98	0.99	0.99	685
Extra Trees	spam	0.95	0.91	0.93	140
Logistic Regression	ham	0.99	0.98	0.98	685
Logistic Regression	spam	0.91	0.93	0.92	140
Neural Network	ham	0.98	0.99	0.98	685
Neural Network	spam	0.95	0.89	0.92	140

Από τα παραπάνω καταλήξαμε στην επιλογή των τριών πιο αποδοτικών μοντέλων μέσα από τα οποία κάθε μήνυμα του chat room, αφού περάσει την προεπεξεργασία που αναφέραμε, σε αυτό το σημείο πρέπει να τονίσουμε ότι έχουμε αποθηκεύσει ένα διάνυσμα που περιέχει το idf του train dataset, έτσι ώστε να μπορεί να υπολογιστεί το χαρακτηριστικό διάνυσμα για κάθε νέο μήνυμα και να μην περιοριστούμε στην χρήση μόνο των προτάσεων που δίνονται απο το dataset. Σε περίπτωση που ένα μήνυμα χαρακτηριστεί ως spam από τουλάχιστον δύο μοντέλα τότε στέλνεται ένα μήνυμα προς όλους τους χρήστες το οποίο λέει ότι ένας συγκεκριμένος χρήστης προσπάθησε να στείλει ένα spam μήνυμα, σε αντίθετη περίπτωση το μήνυμα αποστέλεται κανονικά. Παρακάτω παραθέτονται εικόνες με παραδείγματα χρήσης του chat room για την κατασκευή του οποιού συμβουλευτήκαμε το παρακάτω άρθρο https://www.geeksforgeeks.org/gui-chat-application-using-tkinter-in-python/

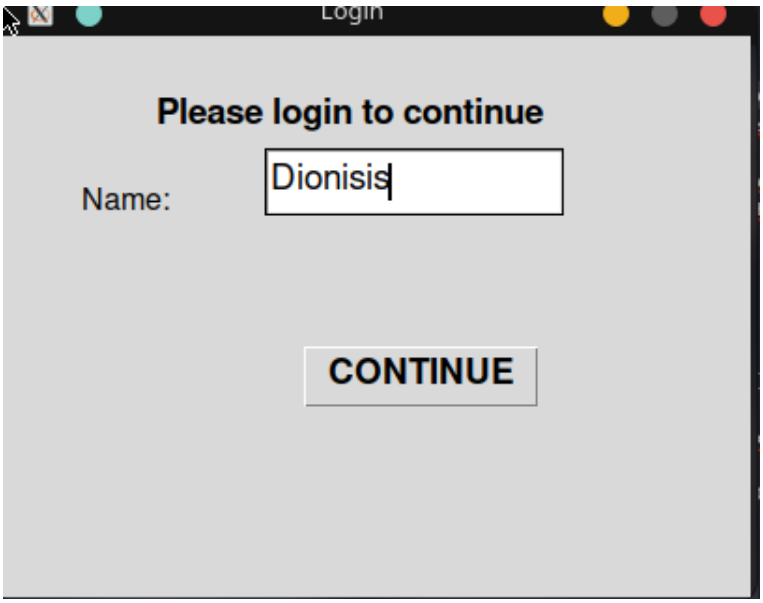

Figure 9: Σύνδεση Χρήστη στο chat room

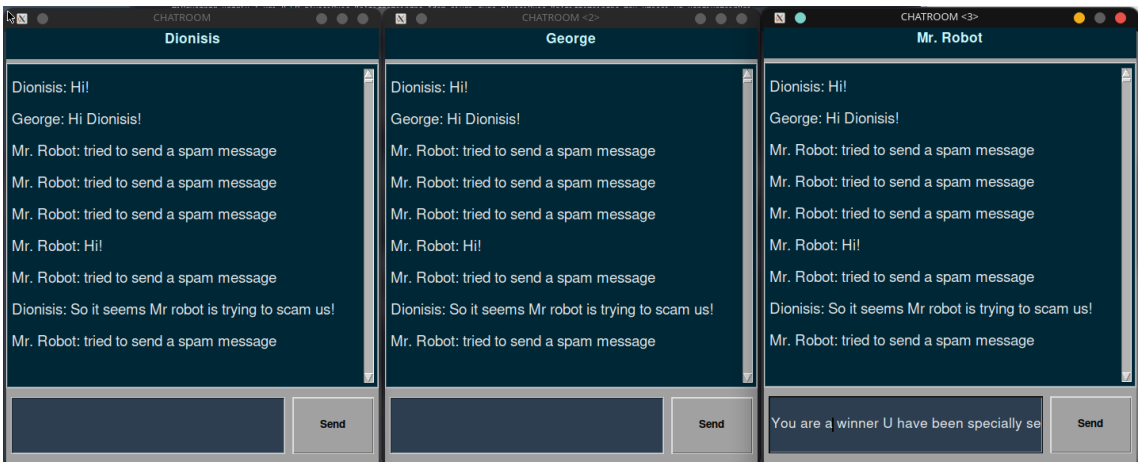

Figure 10: Παράδειγμα αποτροπής αποστολής ενός spam μηνύματος στο chat

2.6.1 Κώδικας python3 Server του chat room

```
import socket
import threading
from collections import Counter
import pandas as pd
from utils import *
import pickle
import torch.nn as nn
import torch
class BinaryClassification(nn.Module):
    def __init__(self, input_dim):
        super(BinaryClassification, self).__init__()
        self.layer_1 = nn.Linear(input_dim, 512)
        self.layer_2 = nn.Linear(512, 128)
        self.layer_3 = nn.Linear(128, 32)
        self.layer_out = nn.Linear(32, 1)
        self.relu = nn.ReLU()


    def forward(self, inputs):
        x = self.relu(self.layer_1(inputs))
        x = self.relu(self.layer_2(x))
        x = self.relu(self.layer_3(x))
        x = torch.sigmoid(self.layer_out(x))


        return x
import numpy as np

# read columns template pandas series representing a tf idf vector for every message to be classified
template = pd.read_csv('data/test.csv',nrows=1)


# read idf needed for the tfidf vector representation of every new message for every term of the message
# that is in the corpus vocabulary tf * idf we used this method since we wanted to calculate the tfidf
# vector even for messages that have not been already represented
idf = pd.read_csv("data/idf.csv")

# load extra trees trained model
extra_trees = pickle.load(open('models/etc.pkl', 'rb'))
# load extra trees trained model
random_forest = pickle.load(open('models/rf.pkl', 'rb'))

# load trained neural net
model = BinaryClassification(len(template.iloc[0])-2)
model.load_state_dict(torch.load('models/nn.pt'))
model.eval()


PORT = 5000
```

```
50
51  # An IPv4 address is obtained
52  # for the server.
53  SERVER = socket.gethostbyname(socket.gethostname())
54
55  # Address is stored as a tuple
56  ADDRESS = (SERVER, PORT)
57
58  FORMAT = "utf-8"
59
60  # Lists that will contains
61  # all the clients connected to
62  # the server and their names.
63  clients, names = [], []
64
65  # Create a new socket for
66  # the server
67  server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
68
69  # bind the address of the
70  # server to the socket
71  server.bind(ADDRESS)
72
73  # function to start the connection
74  def startChat():
75
76      print("server is working on " + SERVER)
77
78      # listening for connections
79      server.listen()
80
81      while True:
82
83          # accept connections and returns
84          # a new connection to the client
85          #  and  the address bound to it
86          conn, addr = server.accept()
87          conn.send("NAME".encode(FORMAT))
88
89          # 1024 represents the max amount
90          # of data that can be received (bytes)
91          name = conn.recv(1024).decode(FORMAT)
92
93          # append the name and client
94          # to the respective list
95          names.append(name)
96          clients.append(conn)
97          print(f"Name is :{name}")
98          # broadcast message
99          broadcastMessage(f"{name} has joined the chat!".encode(FORMAT))
100         conn.send('Connection successful!'.encode(FORMAT))
101
102         # Start the handling thread
103         thread = threading.Thread(target=handle, args=(conn, addr))
104         thread.start()
105
106         # no. of clients connected
107         # to the server
108         print(f"active connections {threading.activeCount()-1}")
109
110 # method to handle the
111 # incoming messages
112 def handle(conn, addr):
113
114     print(f"new connection {addr}")
115     connected = True
116
117     while connected:
118         # receive message
119         message = conn.recv(1024)
120         print(message)
121         # broadcast message
122         broadcastMessage(str.encode(is_spam(message)))
123
124     # close the connection
125     conn.close()
```

```
126
127 # method for broadcasting
128 # messages to the each clients
129 def broadcastMessage(message):
130     for client in clients:
131         client.send(message)
132
133
134 def is_spam(message):
135     # read 3 of the best models and pass the text trough them
136     # if at least one method cosiders that the message is spam instead of the message
137     # an error is broadcasted
138
139     # byte to str
140     message = message.decode()
141
142     # exctract user from message
143     user = ""
144     for i in range(0,len(message)):
145         user+=str(message[i])
146         if message[i] ==":":
147             break
148
149     # isolate message withot the user:
150     new_mes = message[i+1:]
151
152     # preprocess message
153     processed = message_processing(new_mes)
154
155     # calculate parameters
156     length = len(new_mes)
157     has_num = has_numbers_text(new_mes)
158     has_cur = has_currency_text(new_mes)
159
160     # count term frequency for every word of the message
161     tf = Counter(processed)
162
163     # calculate tf-idf weight using corpus idf
164     for col in template.columns:
165         template[col].values[:] = 0.0
166
167
168     # update template vector (zeros initialized)
169     template.has_num.replace(0, has_num)
170     template.has_money.replace(0, has_cur)
171     template.length.replace(0, length)
172
173     # convert message to tf-df vector
174     # load random forest
175     for key in tf:
176         try:
177             # operation to raise exception if word is not in corpus vocabulary
178             a = (template[key])
179             # update coresponding tf-idf value
180             template[key]=tf.get(key) * idf.key.iloc[[0]].values
181         except:
182             print("word not in corpus vocabulary")
183
184     # sum of the models output (range 0 to 3 )
185     sum = 0
186     # extract vector to be inserted in models
187     input = template.iloc[0][2:].copy().to_numpy().reshape(-1,len(template.iloc[0][2:]))
188     # extra trees output
189     sum += extra_trees.predict(input)
190     # random forest output
191     sum += random_forest.predict(input)
192     # neural net output
193     with torch.no_grad():
194         sum+=torch.round(model(torch.Tensor(input.astype(float)))).item()
195     # if no more than 1 model classified the message as spam return it as is
196     if(sum<2):
197         return(message)
198     # if at least two model marked the message as spam
199     else:
200         # Print in console that a spam message was found by the user
201         print("SPAM Message found from : ", user)
```

```
        # return warning to be displayed so that all users know that a user tried to scam them
        return(str(user)+" tried to send a spam message")

# call the method to
# begin the communication
startChat()
```

2.6.2 Κώδικας python3 Client του chat room

```
# import all the required  modules
import socket
import threading
from tkinter import *
from tkinter import font
from tkinter import ttk

# import all functions /
#  everything from chat.py file
from chat import *

PORT = 5000
SERVER = "127.0.1.1"
ADDRESS = (SERVER, PORT)
FORMAT = "utf-8"

# Create a new client socket
# and connect to the server
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect(ADDRESS)


# GUI class for the chat
class GUI:
    # constructor method
    def __init__(self):

        # chat window which is currently hidden
        self.Window = Tk()
        self.Window.withdraw()

        # login window
        self.login = Toplevel()
        # set the title
        self.login.title("Login")
        self.login.resizable(width=False, height=False)
        self.login.configure(width=400, height=300)
        # create a Label
        self.pls = Label(self.login,
                         text="Please login to continue",
                         justify=CENTER,
                         font="Helvetica 14 bold")

        self.pls.place(relheight=0.15, relx=0.2, rely=0.07)
        # create a Label
        self.labelName = Label(self.login, text="Name: ", font="Helvetica 12")

        self.labelName.place(relheight=0.2, relx=0.1, rely=0.2)

        # create a entry box for
        # tyoing the message
        self.entryName = Entry(self.login, font="Helvetica 14")

        self.entryName.place(relwidth=0.4, relheight=0.12, relx=0.35, rely=0.2)

        # set the focus of the cursor
        self.entryName.focus()

        # create a Continue Button
        # along with action
        self.go = Button(self.login,
                         text="CONTINUE",
                         font="Helvetica 14 bold",
                         command=lambda: self.goAhead(self.entryName.get()))

        self.go.place(relx=0.4, rely=0.55)
        self.Window.mainloop()

```

```
    def goAhead(self, name):
        self.login.destroy()
        self.layout(name)

        # the thread to receive messages
        rcv = threading.Thread(target=self.receive)
        rcv.start()

    # The main layout of the chat
    def layout(self, name):

        self.name = name
        # to show chat window
        self.Window.deiconify()
        self.Window.title("CHATROOM")
        self.Window.resizable(width=False,
                              height=False)
        self.Window.configure(width=470,
                              height=550,
                              bg="#002736")
        self.labelHead = Label(self.Window, bg="#002736",
                               fg="#C9FBFF",
                               text=self.name,
                               font="Helvetica 13 bold",
                               pady=5)

        self.labelHead.place(relwidth=1)
        self.line = Label(self.Window,
                          width=450,
                          bg="#A1A1A1")

        self.line.place(relwidth=1,
                        rely=0.07,
                        relheight=0.012)

        self.textCons = Text(self.Window,
                             width=20,
                             height=2,
                             bg="#002736",
                             fg="#EAECEE",
                             font="Helvetica 14",
                             padx=5,
                             pady=5)

        self.textCons.place(relheight=0.745,
                            relwidth=1,
                            rely=0.08)

        self.labelBottom = Label(self.Window,
                                 bg="#A1A1A1",
                                 height=80)

        self.labelBottom.place(relwidth=1,
                               rely=0.825)

        self.entryMsg = Entry(self.labelBottom,
                              bg="#2C3E50",
                              fg="#EAECEE",
                              font="Helvetica 13")

        # place the given widget
        # into the gui window
        self.entryMsg.place(relwidth=0.74,
                            relheight=0.06,
                            rely=0.008,
                            relx=0.011)

        self.entryMsg.focus()

        # create a Send Button
        self.buttonMsg = Button(self.labelBottom,
                                text="Send",
                                font="Helvetica 10 bold",
                                width=20,
                                bg="#A1A1A1",
                                command=lambda: self.sendButton(self.entryMsg.get()))
```

```
145
146         self.buttonMsg.place(relx=0.77,
147                              rely=0.008,
148                              relheight=0.06,
149                              relwidth=0.22)
150
151         self.textCons.config(cursor="arrow")
152
153         # create a scroll bar
154         scrollbar = Scrollbar(self.textCons)
155
156         # place the scroll bar
157         # into the gui window
158         scrollbar.place(relheight=1,
159                         relx=0.974)
160
161         scrollbar.config(command=self.textCons.yview)
162
163         self.textCons.config(state=DISABLED)
164
165     # function to basically start the thread for sending messages
166     def sendButton(self, msg):
167         self.textCons.config(state=DISABLED)
168         self.msg = msg
169         self.entryMsg.delete(0, END)
170         snd = threading.Thread(target=self.sendMessage)
171         snd.start()
172
173     # function to receive messages
174     def receive(self):
175         while True:
176             try:
177                 message = client.recv(1024).decode(FORMAT)
178
179                 # if the messages from the server is NAME send the client's name
180                 if message == 'NAME':
181                     client.send(self.name.encode(FORMAT))
182                 else:
183                     # insert messages to text box
184                     self.textCons.config(state=NORMAL)
185                     self.textCons.insert(END,
186                                          message+"\n\n")
187
188                     self.textCons.config(state=DISABLED)
189                     self.textCons.see(END)
190             except:
191                 # an error will be printed on the command line or console if there's an error
192                 print("An error occured!")
193                 client.close()
194                 break
195
196     # function to send messages
197     def sendMessage(self):
198         self.textCons.config(state=DISABLED)
199         while True:
200             message = (f"{self.name}: {self.msg}")
201             client.send(message.encode(FORMAT))
202             break
203
204
205 # create a GUI class object
206 g = GUI()
```

3 Παράρτημα

Παρακάτω ακολουθεί μια βιβλιοθήκη που περιέχει συναρτήσεις που χρησιμοποιούνται συχνά απο διάφορα scripts.

```

import torch
import pandas as pd
from torch.utils.data import Dataset
from nltk import word_tokenize
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords as sw
import re

# text processing
ps = PorterStemmer()
stop_words = set(sw.words("english"))

# check if pandas series text field has a number
def has_numbers(row):
    text = row["text"]
    string = word_tokenize(text)
    for word in string:
        if bool(re.search(r'\d', word)):
            return 1
    return 0

# check if pandas series text field has a currency symbol
def has_currency(row):
    text = row["text"]
    string = word_tokenize(text)
    for word in string:
        if('$' in word or '  ' in word or '    ' in word):
            return 1
    return 0

# check if a string has numbers
def has_numbers_text(text):
    string = word_tokenize(text)
    for word in string:
        if bool(re.search(r'\d', word)):
            return 1
    return 0

# check if a string has currency symbols
def has_currency_text(text):
    string = word_tokenize(text)
    for word in string:
        if('$' in word or '  ' in word or '    ' in word):
            return 1
    return 0

# calculate text field length from pandas series
def sms_len(row):
    return len(row["text"])

# stem words that are not stop words and are alphabetic strings from text field of pandas series
def processing(row):
    text = row["text"]
    tokens = word_tokenize(text)
    stemmed_tokens = []
    stemmed_tokens = [ps.stem(word.lower()) for word in tokens if (word not in stop_words and word.isalpha())]
    joined = (" ".join(stemmed_tokens))

    return joined

# stem words that are not stop words and are alphabetic strings from string
def message_processing(text):
    tokens = word_tokenize(text)
    stemmed_tokens = []
    stemmed_tokens = [ps.stem(word.lower()) for word in tokens if (word not in stop_words and word.isalpha())]
    joined = (" ".join(stemmed_tokens))
    return word_tokenize(joined)

# list or numpy dataset to torch tensor
```

```
class CustomDataset(Dataset):
    """ Custom Spam or Ham Dataset. """

    def __init__(self, x, y):
        """
        Args:
            x (numpy array): # tf-idf vector representation of message
            y (numpy array): # class spam or ham
        """
        self.x = torch.Tensor(x)
        self.y = torch.Tensor(y)

    def __len__(self):
        return len(self.x)

    def __getitem__(self, idx):
        return self.x[idx], self.y[idx]

# read train or test dataset
def read_data(id):
    '''
        id = test or train
    '''
    # Read Train Data
    df = pd.read_csv('data/' + id +'.csv', index_col=0)
    df.category = pd.factorize(df.category)[0]
    X, y = df.iloc[:, 1:].to_numpy(), df["category"].to_numpy()
    X = torch.Tensor(X)
    y = torch.Tensor(y)
    return X, y

# plot confusion matrix
def plot_cf(conf_matrix,name,show=False):
    import matplotlib.pyplot as plt
    fig, ax = plt.subplots(figsize=(7.5, 7.5))
    ax.matshow(conf_matrix, cmap=plt.cm.Blues, alpha=0.3)
    for i in range(conf_matrix.shape[0]):
        for j in range(conf_matrix.shape[1]):
            ax.text(x=j, y=i,s=conf_matrix[i, j], va='center', ha='center', size='xx-large')

    plt.xlabel('Predictions', fontsize=18)
    plt.ylabel('Actuals', fontsize=18)
    plt.title('Confusion Matrix '+name, fontsize=18)
    if show:
        plt.show()
    else:
        plt.savefig("plots/"+name+'_cf.png')

# decorate axis from plot
def decorate_axis(ax, remove_left=True):
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.spines['left'].set_visible(False)
    ax.get_xaxis().tick_bottom()
    ax.get_yaxis().tick_left()
    ax.tick_params(axis='x', direction='out')
    ax.tick_params(axis='y', length=0)

    ax.grid(axis='y', color="0.9", linestyle='-', linewidth=1)
    ax.grid(axis='x', color="0.9", linestyle='-', linewidth=1)
    ax.set_axisbelow(True)
```

4 Βιβλιογραφία

[1] Tiago A. Almeida, José María Gómez Hidalgo, and Akebo Yamakami. "Contributions to the study of SMS spam filtering: new collection and results". In: *DocEng '11*. 2011.

[2] Wes McKinney. "Data Structures for Statistical Computing in Python". In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman. 2010, pp. 56–61. DOI: 10.25080/Majora-92bf1922-00a.

[3] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

[4] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[5] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[6] Geoffrey I. Webb. "Naïve Bayes". In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 713–714. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_576. URL: https://doi.org/10.1007/978-0-387-30164-8_576.

[7] Ashraf M. Kibriya et al. "Multinomial Naive Bayes for Text Categorization Revisited". In: *AI 2004: Advances in Artificial Intelligence*. Ed. by Geoffrey I. Webb and Xinghuo Yu. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 488–499. ISBN: 978-3-540-30549-1.

[8] Xindong Wu et al. "Top 10 algorithms in data mining". In: *Knowledge and information systems* 14.1 (2008), pp. 1–37.

[9] Jaak Simm, Ildefons Magrans de Abril, and Masashi Sugiyama. *Tree-Based Ensemble Multi-Task Learning Method for Classification and Regression*. 6. The Institute of Electronics, Information and Communication Engineers, 2014, pp. 1677–1681. URL: http://CRAN.R-project.org/package=extraTrees.

[10] David R Cox. "The regression analysis of binary sequences". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 20.2 (1958), pp. 215–232.

[11] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[12] Diederik Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations* (Dec. 2014).

[13] Agnes Lydia and Sagayaraj Francis. "Adagrad - An Optimizer for Stochastic Gradient Descent". In: Volume 6 (May 2019), pp. 566–568.