Learned Personalized Predictors For COVID-19 Infections

Juho Tuomaala

School of Electrical Engineering

Bachelor's thesis
Espoo 18.12.2021

Supervisor

D.Sc. Markus Turunen

Advisor

Prof. Alexander Jung

Aalto University
School of Electrical
Engineering

Aalto University
School of Electrical Engineering

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the bachelor's thesis

Author Juho Tuomaala

Title Learned Personalized Predictors For COVID-19 Infections

Degree programme Bachelor's Programme in Electrical Engineering

Major Bioinformation Technology **Code of major** ELEC3016

Teacher in charge D.Sc. Markus Turunen

Advisor Prof. Alexander Jung

Date 18.12.2021 **Number of pages** 29+12 **Language** English

Abstract

COVID-19 pandemic has been one of the most influential health crises of the recent history. During the pandemic, health care organizations have been faced with challenges related to infection tracking, diagnosing and care prioritization. Many of these operations produce and require different types of data. Analysing the data is key in optimizing these processes, thus making machine learning a viable tool in solving these problems.

The objective of this thesis was to predict whether a patient infected with COVID-19 was hospitalized based on health data. To achieve this, three different supervised machine learning models were trained and evaluated: logistic regression, support vector machines, and gradient boosting. The data used in the analysis was a patient-level symptom information dataset provided by the Centers of Disease Control and Prevention. The dataset included over 30 million datapoints with patient-level data, including hospitalization status, whether the patient was deceased because of the illness, along with demographic and symptoms information.

The most important evaluation metric used in the analysis was the F_1-score. Out of the three candidate models, the gradient boosting model achieved the best F_1-score of 0.574. The corresponding F_1-scores for logistic regression and support vector machines were 0.504 and 0.521, respectively.

Predicting the need for hospitalization proved a challenging machine learning problem using supervised methods. Nevertheless, the models studied in this thesis had meaningful predictive power. Given the results from this thesis along with related, previous studies, it can be concluded that supervised machine learning methods can be successfully applied to problems similar to the one of this thesis.

Keywords Machine Learning, COVID-19

Aalto-yliopisto
Sähkötekniikan korkeakoulu

Aalto-yliopisto, PL 11000, 00076 AALTO
www.aalto.fi
Tekniikan kandidaatintyön tiivistelmä

Tekijä Juho Tuomaala

Työn nimi Opitut henkilökohtaiset ennustusmallit koronavirusinfektiolle

Koulutusohjelma Sähkötekniikan kandidaattiohjelma

Pääaine Bioinformaatioteknologia **Pääaineen koodi** ELEC3016

Vastuuopettaja TkT Markus Turunen

Työn ohjaaja Prof. Alexander Jung

Päivämäärä 18.12.2021 **Sivumäärä** 29+12 **Kieli** Englanti

Tiivistelmä

Maailmanlaajuinen koronaviruspandemia on yksi lähihistorian tärkeimpiä terveyskriisejä. Useat terveydenhuollon operaatiot, kuten tautiketjujen jäljitys, hoitoresurssien allokointi ja tartuntojen diagnosointi tuottavat suuria määriä erilaista dataa, ja niiden toiminta on siitä riippuvaista. Data-analyysi ja erilaiset tilastolliset mallit ovat keskeisiä näiden operaatioiden optimoinnissa, ja koneoppiminen on yksi näihin tehtäviin soveltuva työkalu.

Koneoppimismallit ovat algoritmeja, jotka oppivat säännönmukaisuuksia datasta ja joita voidaan tämän jälkeen käyttää uuden, samankaltaisen datan ominaisuuksien ennustamiseen. Terveydenhuollon tuottama data voi olla esimerkiksi röntgen- tai tietokonekerroskuvia, terveyskyselyiden tuloksia tai tartuntalukutilastoja. Kaikkia näitä voidaan hyödyntää koneoppimismallien kouluttamiseen: koronaviruspandemian aikana erilaisia koneoppimismalleja on jo käytetty muun muassa tartuntojen diagnosointiin ja tartuntamäärien ennustamiseen.

Tämän kandidaatintyön tarkoitus on luoda koneoppimiseen perustuva ennustusmalli yksittäisten koronavirusinfektioiden vaikutuksille, kuten sairaalahoidon tarpeelle. Työssä paino on erityisesti ohjatuissa koneoppimismalleissa, jotka koulutetaan tunnistamaan esimerkiksi kuoleman tai sairaalahoidon tarpeen riskiä terveysdatan perusteella. Työssä esitellään tarkemmin kolme eri koneoppimismallia: logistinen regressio, tukivektorikone ja gradienttivahvistusmalli. Työssä mallien kouluttamisen ja testauksen lisäksi niistä saatuja tuloksia vertaillaan keskenään ja niiden soveltuvuutta tämän koneoppimisongelman ratkaisuun pohditaan.

Työssä hyödynnettiin Yhdysvaltain tautikeskuksen (engl. Centers for Disease Control and Prevention, lyh. CDC) keräämää dataa yksittäisistä koronaviruspotilaista. Aineisto sisälsi nimettömänä yli 30 miljoonan henkilön oleellisesti tartuntaan liittyvät tiedot, kuten havaitut oireet ja henkilön iän, sekä tiedon siitä, oliko henkilö joutunut sairaalaan, tehohoitoon tai kuollut taudin seurauksena.

Tässä työssä tärkein mallien arviointiin käytetty mittari oli mallin tarkkuuden ja herkkyyden harmoninen keskiarvo eli F_1-luku. F_1-luvun perusteella parhaiten sairaalahoidon tarvetta pystyttiin ennustamaan gradienttivahvistusmallilla. Lopullisessa testauksessa sen F_1-luvuksi sairaalahoitoa tarvitsevan luokan ennustuksille saatiin 0,574, tarkkuudeksi 0,615 ja herkkyydeksi 0,539. Erot eri mallien välillä olivat kuitenkin pieniä: vastaavaksi F_1-luvuksi saatiin logistiselle regressiolle 0,504 ja tukivektorikoneelle 0,521.

Sairaalahoidon tarpeen tai kuoleman ennustaminen yksinkertaisen terveysdatan perusteella osoittautui haastavaksi koneoppimisongelmaksi. Koulutettu malli ei siis ole täydellinen ennustustyökalu vaan kompromissi mallin tarkkuuden ja herkkyyden välillä. Työssä käsiteltyjen kaltaisia koneoppimismalleja ja dataa on kuitenkin mahdollista hyödyntää rinnakkain muiden, esimerkiksi keuhkokuvia analysoivien mallien kanssa.

Avainsanat Koneoppiminen, COVID-19

Contents

1 Symbols and abbreviations

Symbols

$\mathcal{H}^n$	n dimensional hypothesis space
$\mathbf{x_i}$	feature vector of datapoint i
x_{ij}	value of feature j of datapoint i
y_i	target label of datapoint i

Abbreviations

GFT	Google Flu Trends
LASSO	Least Absolute Shrinkage and Selection Operator
CT	Computed Tomography
PCR	Polymerase Chain Reaction
PCA	Principal Component Analysis
SVM	Support Vector Machines
SGD	Stochastic Gradient Descent

2 Introduction

The COVID-19 pandemic has been one of the most influential and widespread health crises in the recent history. Originating in Wuhan, China in late 2019, the virus has since spread across the whole world, causing more than 230 000 000 confirmed infections as of October 2021, according to the World Health Organization [1]. Due to its aggressive spreading and in some cases severe symptoms, health care organizations are faced with a multitude of varying challenges. As an example, these challenges include infection chain tracking, testing and prioritization of hospital care and vaccinations. Solving these problems often involve collecting and analysing data, which makes various machine learning methods viable tools in the solving process.

Machine learning models are algorithms that learn and adapt in an effort to predict and draw conclusions from data [2]. Today, machine learning routines are widely used tools in many fields of science and technology, and as stated by Kushwaha et al. [3], different machine learning models have already been applied to COVID-19 related problems: these include antibody recognition, identifying high risk patients and predicting the virus spreading. Developing and studying these methods is highly important not only because of the COVID-19 pandemic, but also for preparing for the next, eventual global pandemic.

This thesis aims to study different machine learning models for creating personalized predictors for COVID-19 infections: this means predicting different consequences resulting from COVID-19 infections, such as death or need for intense care, based on an individual's health information and demographic data. An overview of previously done machine learning-based research and analysis on the topic will be provided. Along with presenting and comparing previous results, the thesis will discuss different machine learning models that could be utilized in the analysis; specifically, which models are best for solving the problem and what are their key characteristics. With the previous research in the background, original empirical analysis will be conducted using supervised classification methods.

First, we will review previously done research and literature on the topic. In the next section, the material and methods will be presented, including the data and used machine learning models. Finally, we will evaluate the analysis results and summarize the findings.

3 Background

Different challenges that have arisen with the pandemic can be formulated into a multitude of machine learning problems: these include infection diagnosing, predicting virus spread, screening patients for intense care units and mortality risk assessment. This chapter focuses on providing an overview on previously done research on the topic. More precisely, we examine the different datasets, machine learning models and analysis results in previously conducted research similar to this thesis.

Disease spread forecasting

Predicting the future spreading of the virus and infection rate via machine learning is a common research topic in the field. Many health care operations benefit from being able to accurately predict the number of infections as it allows for more efficient resource allocation. To extract the underlying information, different types of data from various sources has been gathered in previous studies. For example, the data may have included geographical information of previous infections, social media posts, weather conditions, demographic information, or any combination of these. When predicting the future infection numbers, a regression model is the natural choice.

A notable application in disease spread prediction was the Google Flu Trends (GFT) [4]. Operating from 2008 to 2015, GFT utilized search query data and a linear prediction model in an attempt to predict the spread of influenza. As pointed out by Cook et al. [5], GFT managed to predict the spread of the H1N1 virus in 2009 with highly varying accuracy; while the model could accurately predict the infection numbers during regular seasonal flu, it performed poorly during the global H1N1 pandemic, especially during the infection peaks.

The idea of GFT was later applied and further developed in numerous studies. One of the more recent ones, conducted by Guo et al. during the COVID-19 pandemic, collected user posts from a Chinese social media platform Weibo and used them to learn a predictive model for COVID-19 infections in Wuhan, China [6]. In the study, a genetic algorithm was first used to learn a set of relevant keywords (such as "hospital", "testing", "patient" etc.) from the Weibo posts. Next, a linear regression model was trained based on both the frequency of said keywords in recent Weibo posts and time series data of confirmed infections. The model performance was estimated by calculating the Pearson correlation coefficient R between the predicted values and the real outcome: for a predicting period of 7 days, the model was reported to follow the real infection numbers with $R = 0.88$.

Another approach towards a similar problem was taken by Rustam et al. [7]. In order to predict future infection cases, time series data consisting of previous infection numbers were used to train different linear models, namely linear regression, support vector machines, least absolute shrinkage and selection operator (LASSO),

and exponential smoothing. The models were set to predict a 10-day period into the future, and the results were evaluated based on multiple metrics, including R^2 score and mean squared error. Exponential smoothing was found to be the strongest performing model, while the support vector machines was evaluated to be the weakest.

Individual infection diagnosis

Another typical machine learning problem related to COVID-19 and other diseases is diagnosing infections or predicting infection consequences. These types of problems are most often formulated as classification tasks. The datasets used to identify certain viral infections may consists of health data gathered from surveys, physiological measurement values, or medical images such as computed tomography (CT) or X-ray.

In a 2021 study conducted by Han et al. [8], a semi-supervised machine learning model was deployed to detect COVID-19 infections from CT scan images. Models were trained for two distinct tasks: distinguishing between CT images of COVID-19 positive and negative patients, and differentiating between COVID-19 and common pneumonia from the CT images. The models reached accuracies of 99.83% and 97.32% respectively, indicating that the model could be used as a reliable diagnosis tool for COVID-19 infections alongside with polymerase chain reaction (PCR) testing.

A different approach was taken in a study by Muhammad et al. [9], in which different supervised machine learning models were trained to predict positive COVID-19 infections from epidemiology dataset. The data consisted of over 260 000 datapoints with 41 columns, containing both demographic and clinical information. The trained models' performances were evaluated based on the prediction accuracy, sensitivity and specificity. The highest accuracy of 94.99% was achieved with a decision tree model, highest sensitivity of 93.34% with a support vector machine, and highest specificity of 94.30% with a Naïve Bayes Model.

The previously introduced examples along with numerous related studies suggest that given suitable, plentiful data and a fitting machine learning model, robust and accurate predictors with real-world applicability can be achieved.

4 Methodology

The purpose of this chapter is to explicate the empirical analysis done for the thesis. This includes describing the used dataset, data preprocessing methods, different machine learning models used in the analysis, and model performance evaluation methods. Generalized process flow used in the analysis is illustrated in figure (1). The process steps in the flowchart will be covered in sections to come.

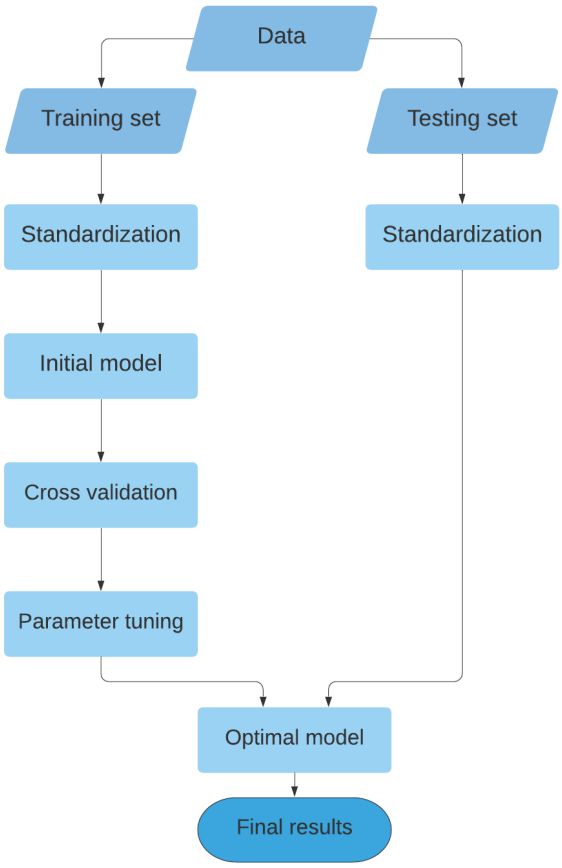


Figure 1: Flowchart of the analysis flow.

4.1 Data and preprocessing

In machine learning, finding proper data is perhaps even more important than the model itself. In this thesis, the analysis was based on COVID-19 Case Surveillance Restricted Access Detailed Data collected in the United States by Centers for Disease Control and Prevention [10] (the CDC does not take responsibility for the scientific validity or accuracy of methodology, results, statistical analyses, or conclusions presented). The initial dataset consists of more than 30 million entries and 32 columns. Due to the terms given by the dataset's use restrictions, the data itself may not be

provided in the thesis. The dataset columns are listed below:

- Initial case report date to CDC
- Date received by CDC or date related to illness/specimen collection (whichever is earlier)
- Symptom onset date, if symptomatic
- Current status (laboratory confirmed or probable)
- Sex
- Age group
- Race and ethnicity
- State of residence
- County of residence
- healthcare worker status
- Pneumonia present
- Acute respiratory distress syndrome present
- Abnormal chest x-ray present
- Was the patient hospitalized?
- Was the patient admitted to an intensive care unit?
- Mechanical ventilation/intubation status
- Date of first positive specimen collection
- Did the patient die as a result of the illness?
- Presence of underlying comorbidity or disease
- Presence of each of the following symptoms: fever, subjective fever, chills, myalgia, rhinorrhea, sore throat, cough, shortness of breath, nausea/vomiting, headache, abdominal pain, diarrhea (each in separate columns)

The patient's need for hospitalization was selected as the target variable (label) in the analysis. In order to possibly predict this property, the most interesting features are the patient's previous health information and age. However, this information is not recorded for the whole dataset, as illustrated in figure (2).

Most machine learning models require complete data, and therefore the partly missing records in our dataset pose a problem. There are two main ways to handle the missing data: simple removal of the entries that are missing information, i.e. list-wise deletion, or artificially constructing the missing values, i.e. data imputation. For data imputation, multiple strategies exist: for instance, missing data can be artificially generated via mean imputation, that is, replacing the missing entries with the mean value of the variable. However, more sophisticated strategies may yield more accurate data. Data imputation itself can be formulated as a machine learning problem, where the training dataset is the complete portion of the original data, and the missing values form the target variable. These strategies are often enhanced with the multiple imputation technique originally described by Rubin in [13]. In multiple imputation, an average of multiple differently imputed datasets are considered in the final model to account for the uncertainty produced in the imputation process.

There is no easy way to tell whether imputation or removal of missing values is

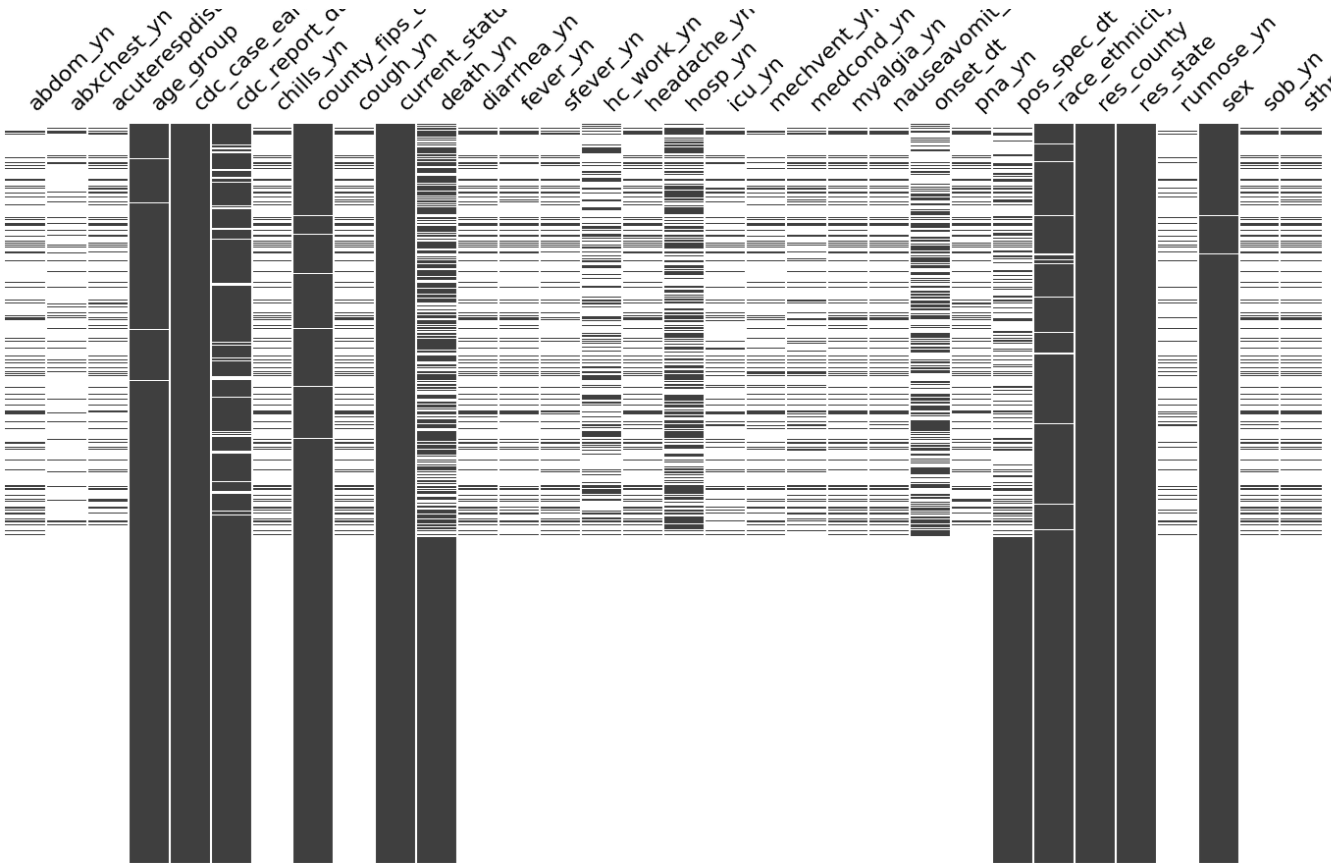


Figure 2: missing data in the dataset. The fullness of the grey vertical lines represents the completeness of the data for the given column; for example, age group and the case report date are almost fully recorded, whereas the symptoms data is missing for most of the initial dataset.

the best option. Both methods may alter the distributions of the data and skew the end results. For the analysis in this thesis, the interesting features are rather sparsely recorded: only 475290 datapoints out of over 30 000 000 contain all the information. By applying data imputation, we would run the risk of heavily skewing the data to one direction or another. Moreover, nearly 500 000 datapoints is more than enough for the machine learning models to function properly. Hence, list-wise deletion was used in this analysis.

After the removal of incomplete records, 475290 datapoints were left in the dataset. It now consists of 15 columns: whether the patient was hospitalized or not, age group, and various symptom information. A histogram of age groups in the dataset is illustrated in figure (3).

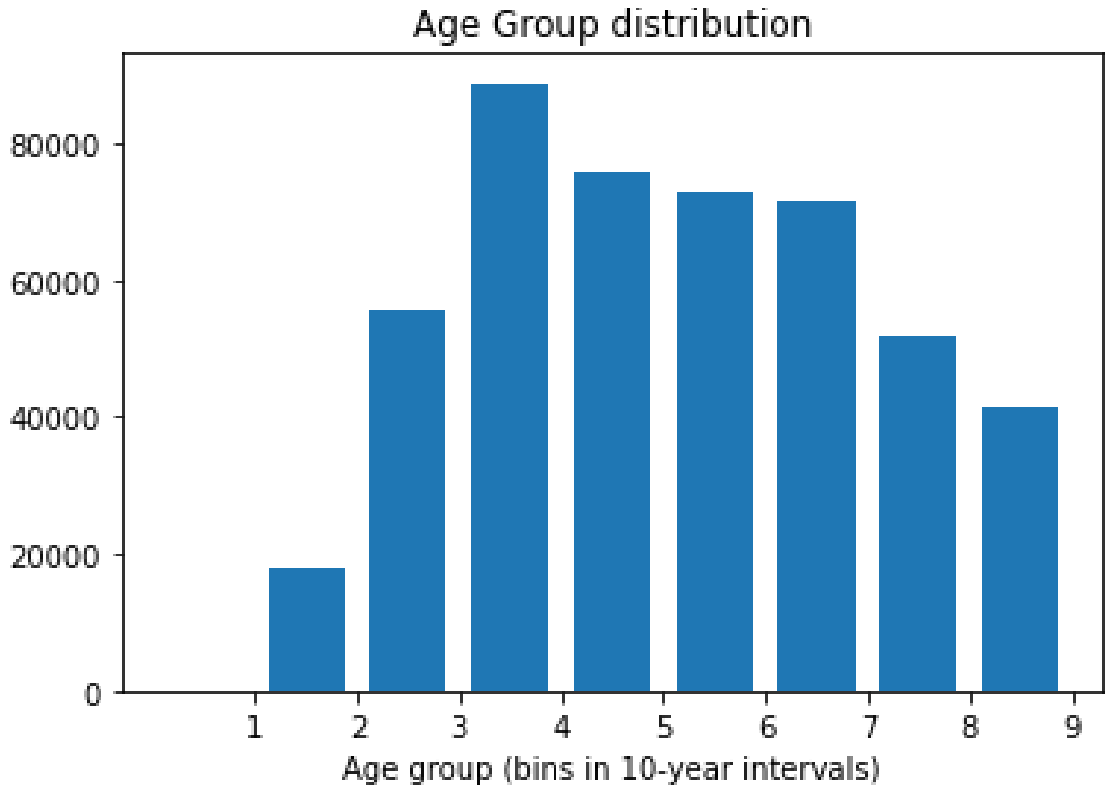


Figure 3: Histogram of the age group variable in the dataset. The bins contain the ages in 10-year intervals from 0 to 80+ years.

Visualizing a high dimensional dataset such as the one in question may prove challenging. A popular method for reducing data dimensionality is principal component analysis (PCA). With PCA, datapoints can be visualized in two dimensions based on the first two principal components. 2-component PCA plot for the data used in this analysis is illustrated in figure (4).

Before feeding the data into the machine learning models, each feature and the target label were mapped from text into numerical data. For example, the "age group" variable was encoded from "year1 - year2" to an integer between 0 and 9, and the rest of the features from "yes" or "no" to 1 or 0. Then, the whole dataset was split into training and testing sets with proportions 4/5 and 1/5, respectively. Next, the values in the data are standardized according to (1):

$$x'_{ij} = \frac{x_{ij} - \mu}{\sigma}, \tag{1}$$

Where x'_{ij} is the new value for the feature j of a datapoint i, x_{ij} is the original value, μ is the mean of feature j, and σ is the standard deviation of said feature. This calculation is applied to both training and testing sets using μ and σ values of the training set; this is to avoid data leakage from testing set into the training set.

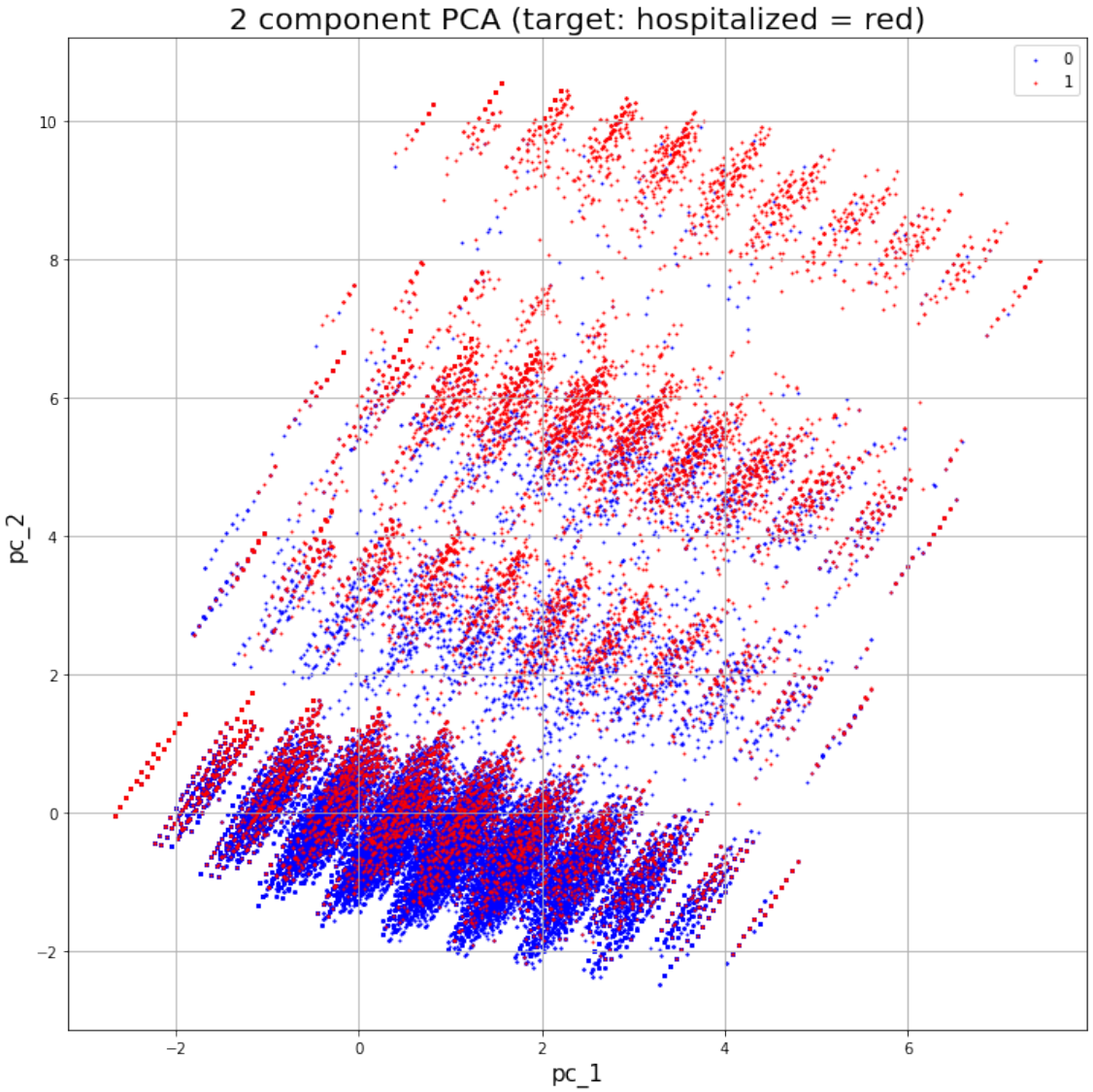


Figure 4: 2-component PCA visualization of the dataset, where hospitalized patients are marked with red points and non-hospitalized with blue. Age groups of the data form the diagonal line-like clusters, while the effect of the rest of the features remains more ambiguous. While the top-left region of the plot is predominantly red (hospitalized patient) indicating easier classification, the rest of red and blue points are not easily separated into clusters of their own.

Standardization is especially important for machine learning models that evaluate the similarity of datapoints based on euclidean distance from one to another. If a feature would have values with much higher magnitudes than others (in this work, the age group feature), it would have elevated influence on the model.

4.2 Utilized Machine Learning Models

In the analysis, three different machine learning models were trained to predict the label of interest: logistic regression, linear support vector machine (SVM), and gradient boosting. The implementations for the logistic regression and the SVM are provided by the Scikit-Learn Python package [11], while the gradient boosting algorithm utilized in the analysis is implemented in the open-source Python library XGBoost [12].

Logistic regression

unlike the name suggests, logistic regression is a classification model most commonly utilized in binary classification problems. However, it is in certain ways analogous to linear regression models; it aims to learn a hypothesis in the same linear hypothesis space, as defined in [15]:

$$\mathcal{H}^n := \{h^{\mathbf{w}} : \mathbb{R}^n \to \mathbb{R} : h^{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T\mathbf{x} \text{ with some weight vector } \mathbf{w} \in \mathbb{R}^n\} \quad (2)$$

In contrast to linear regression models, the conditional distribution of the target label $(y \mid \mathbf{x})$ is bernoulli distributed rather than gaussian, and instead of actual measurement values, the prediction values for y should be interpreted as probabilities. In order to obtain the prediction model, logistic regression minimizes the following loss function

$$\frac{1}{n}\sum_{i=1}^{n} \log(1 + exp(-y_i\mathbf{w}^T\mathbf{x}_i)) \quad (3)$$

where n is the number of data points, y_i is the label of i'th data point, $\mathbf{x}_i$ is the feature vector of i'th data point, and $\mathbf{w}$ is a vector containing the feature weights. Upon minimizing the function (3), that is, finding the optimal weight vector $\mathbf{w}$, the model can predict the label y of a datapoint with feature vector $\mathbf{x}$ as follows:

$$y = \begin{cases} 1, & \mathbf{w^T x} \geq 0 \\ 0, & \text{otherwise.} \end{cases}$$

Support Vector Machines (SVM)

Support vector machines (SVM) are another class of widely used classification methods in machine learning. Similarly to logistic regression, SVM draws a hypothesis map h from the linear hypothesis space $\mathcal{H}^n$. The two models differ in the choice of loss function, which for SVM is called the hinge loss[1]:

$$max\{0, 1 - y \cdot \mathbf{w}^T\mathbf{x}\} + \lambda\|\mathbf{w}\|^2 \quad (4)$$

where λ is a positive tuning parameter. The classifier constructed by a SVM is a hyperplane in the n dimensional feature space, dividing the datapoints into classes.

[1] To be precise, the equation 4 is the soft-margin version of hinge loss, used when the classes are not linearly separable

To minimize the hinge loss is to maximize the distance ξ from the decision boundary $h^{\mathbf{w}}$ to the closest data points, i.e., the support vectors (Figure 5).

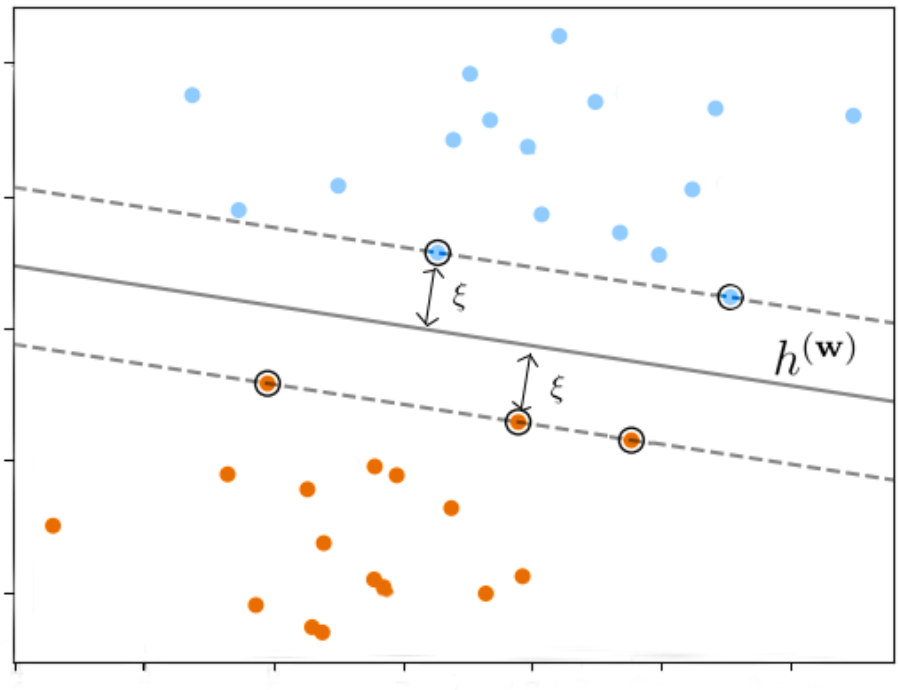


Figure 5: An example of an optimal decision boundary $h^{\mathbf{w}}$ in feature space $\mathbb{R}^2$. In the figure, blue and orange dots represent the data points belonging to different classes.

For both of the above described models, the optimization method used in the analysis was stochastic gradient descent (SGD). Unlike regular gradient descent optimization, SGD can be also applied to non-differentiable functions such as the hinge loss (4). SGD can be viewed as a stochastic approximation of true gradient descent: instead of calculating the gradient for each data point in every iteration step, SGD operates with a randomly selected subset of the entire data set. In addition to mathematical viability, SGD is more computationally efficient than regular gradient descent algorithm, which is especially true for large data sets [16].

Gradient Boosting

A gradient boosting model was selected as the third machine learning model for the analysis. The model utilized in the analysis, XGBoost, builds a decision tree ensemble

as the foundation and learns the optimal tree structure with the boosting method [12].

A decision tree is a simple classification tool used frequently in different machine learning tasks. A decision tree is formed by decision nodes, each representing a feature in the dataset, and leaf nodes which correspond to a prediction value of the target label. One may think of the classification process of a single datapoint as descending down in the tree, selecting the path based on the datapoint's features, and finally arriving to a leaf node that assigns the predicted value for the datapoint. However, a single decision tree is usually not a sufficient predicting model by itself.

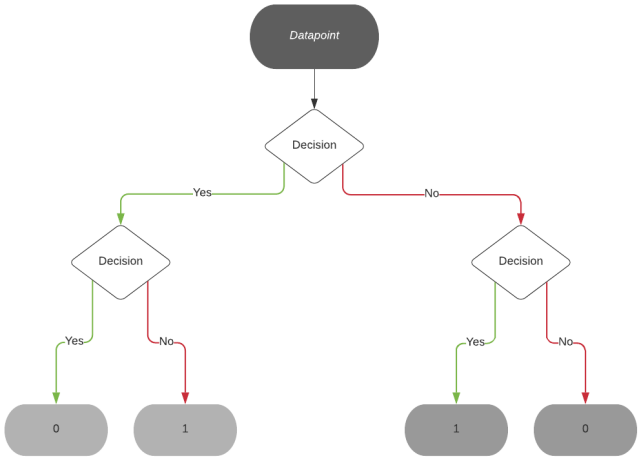


Figure 6: An example of a simple decision tree.

Hence, multiple trees, or in other words, the ensemble, are considered when assigning the final prediction value. Formally, the prediction value $\hat{y}_i$ for a datapoint i can be written as

$$\hat{y}_i = \sum_{k=1}^{K} f_k(\mathbf{x}_i), f_k \in \mathcal{F} \tag{5}$$

where K is the number of trees in the ensemble, f_k is the function of the k'th tree belonging to $\mathcal{F}$, which is a function space containing all possible decision trees. The

objective of the model is to find optimal trees f_k that minimize the objective function, as defined in the XGBoost documentation [12]:

$$obj = \sum_{i=1}^{n} l(y_i, \hat{y_i}^{(t)}) + \sum_{k=1}^{K} \Omega(f_k) \tag{6}$$

In (6), n is the number of datapoints, l is an arbitrary differentiable loss function, for example mean squared error or logistic loss (3), and Ω is a regularization term affecting the model complexity.

4.3 Model performance Evaluation

The prediction results of a binary classifier such as the ones in this thesis can be divided into four categories: true positives (TP, in this analysis correctly predicted to be hospitalized), false positives (FP, incorrectly predicted to be hospitalized), false negatives (FN, incorrectly predicted to not need hospitalization), and true negatives (TN, correctly predicted to not need hospitalization). These four metrics are conventionally arranged into a table called the confusion matrix.

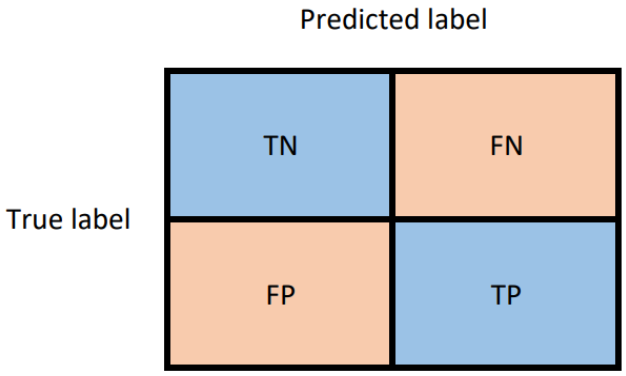


Figure 7: Confusion matrix example.

Numerous different evaluation metrics can be derived from these four values. In machine learning, common metrics in evaluating classification algorithms are the precision and recall values, defined as

$$precision = \frac{TP}{TP + FP} \tag{7}$$

$$recall = \frac{TP}{TP + FN} \tag{8}$$

Together, precision and recall values can be used to evaluate model performance as is. However, additional, single value metrics can be useful, especially during the model optimization process. Two widely used single value metrics are accuracy and F_1 scores:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{9}$$

$$F_1 = 2 * \frac{precision * recall}{precision + recall} \tag{10}$$

Accuracy is the fraction of correct classifications of all datapoints, while the F_1 score is the harmonic mean of recall and precision.

According to the statistics, only a small percentage of COVID-19 infected patients need hospital care [17]. This is visible in the dataset considered in this work as well, as visualized below:

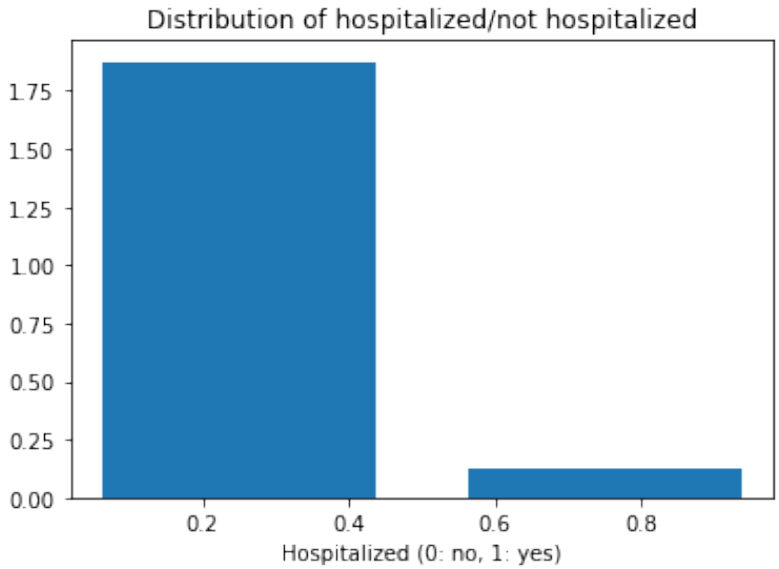


Figure 8: Distribution of hospitalized and non-hospitalized patients in the dataset, non-hospitalized on the left.

The distribution of the target variable is heavily imbalanced, which makes accuracy a poor evaluation metric for this analysis: a classifier that predicted every datapoint as "not hospitalized" would obtain a seemingly decent accuracy of 93%. The F_1 score, however, does not have a similar flaw and hence is more suitable evaluation metric for the analysis of this thesis.

4.4 Cross validation and model parameter tuning

When a machine learning model's predictions match the training set extremely well but fails to make predictions for unseen data, it is said to overfit. To test for overfitting, one should compare the training error and testing error calculated from testing the model with respective splits of the dataset. However, a single split into training and testing and then validating based on them alone may lead to inaccurate estimation results. This is especially true if the dataset is small or imbalanced [15]. To overcome these issues, cross validation techniques can be employed. Cross validation is a resampling technique in which a model is trained for multiple different samples from the same dataset and tested with another sample from the data.

In this analysis, k-fold cross validation was used. In k-fold cross validation, the data is first split into k sets. Then for each k sets, a model is trained using the other k-1 sets and tested with the remaining one. By collecting the results from all k models and calculating the average, a more robust performance estimator is obtained compared to a single split [15].

In addition to validation, optimal prediction results require manual tuning of the model. In machine learning, this process is called hyperparameter tuning. Hyperparameters are parameters in the machine learning model that cannot be optimized directly in the training process [18]. While the simplest models may have only a few or no hyperparameters, the more complicated ones such as XGBoost may have dozens of them. Optimal hyperparameters cannot be solved analytically in practice, and therefore optimizing them requires trial and error, that is, training a model and evaluating its performance.

Exhaustive search for an optimal set of hyperparameters from the whole hyperparameter space quickly becomes computationally unfeasible as the number of hyperparameters increase. In this thesis, a Bayesian optimization method for selecting the hyperparameters was applied. Founded on the famous Bayes theorem (11),

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}, \tag{11}$$

where A and B are events and $P(A \mid B)$ denotes the conditional probability of A given B, the method optimizes an unknown objective function by creating a probabilistic model starting from a prior guess, and iteratively updating the model based on evidence to form the posterior distribution of the objective. Bayesian optimization of machine learning hyperparameters can be summarized by the following, as described by Agrawal in [19]:

1. Build the prior model
2. Start with a guess of hyperparameters

3. Evaluate the objective function score

4. Update the prior model based on the previous evaluation

5. Repeat 3. and 4. for a set number of iterations

By utilizing prior knowledge of already tested sets of hyperparameters, Bayesian hyperparameter optimization spends less time testing poorly performing hyperparameters, thus resulting in a shorter search time.

In this analysis, the objective score to evaluate in the hyperparameter selection process was the mean F_1 score of the positive class (hospitalized) in 3-fold cross validation. The parameter search space was a predefined matrix of hyperparameter values varying in their respective general use ranges. The optimization was applied to the best performing initial model of the three candidate models previously introduced.

5 Results

This section will present the prediction results of the three machine learning models. The results are presented in terms of the evaluation metrics discussed in the previous section: Accuracy, precision, recall and F_1 scores for both of the classes (hospitalized and not hospitalized) as well as the resulting confusion matrices. As previously stated, the target label (hospitalized or not) was encoded as 0 (not hospitalized) and 1 (hospitalized). For each model, prediction results for the testing dataset and the mean F_1 scores from cross validation are presented.

5.1 Logistic Regression

Mean F_1-score for the positive class in 10-fold cross validation for the training set was 0.504 with variance of 0.051.

Results for the logistic regression classifier for the testing set:

Class	Precision	Recall	F_1-score
0	0.9600	0.9921	0.9758
1	0.7631	0.3821	0.5092

Confusion matrix:

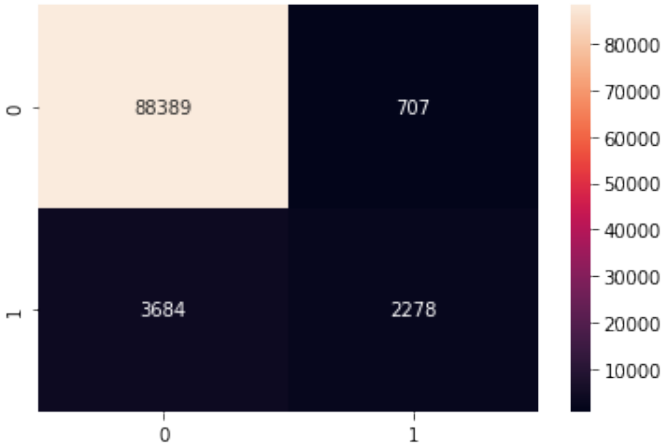


Figure 9: Confusion matrix for logistic regression

5.2 Support Vector Machines

Mean F_1-score for the positive class in 10-fold cross validation for the training set was 0.523 with variance of 0.025.

Results for the SVM classifier for the testing set:

Class	Precision	Recall	F_1-score
0	0.9638	0.9838	0.9737
1	0.6497	0.4477	0.5301

Confusion matrix:

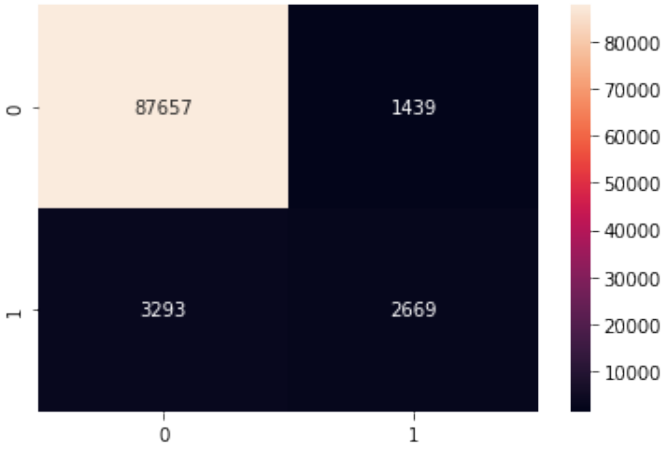


Figure 10: Confusion matrix for support vector machines

5.3 Gradient Boosting

Mean F_1-score for the positive class in 10-fold cross validation for the training set was 0.532 with variance of 0.014.

Results for the gradient boosting classifier for the testing set:

Class	Precision	Recall	F_1-score
0	0.9618	0.9907	0.9760
1	0.7465	0.4114	0.5305

Confusion matrix:

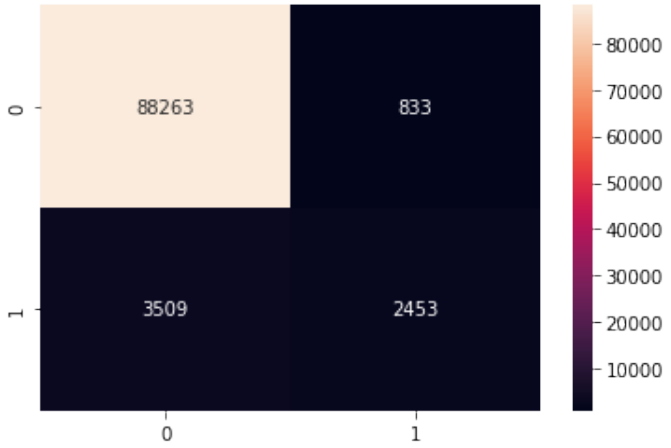


Figure 11: Confusion matrix for gradient boosting

Gradient boosting model had the best performance in terms of F_1 score for both of the classes. Hyperparameter tuning was performed for the gradient boosting model. In bayesian optimization search for the best 3-fold cross validation score, a model with mean F_1 score for the class 1 of 0.5716 was found.

Training results for the optimized gradient boosting model:

Class	Precision	Recall	F_1-score
0	0.9698	0.9769	0.9734
1	0.6130	0.5456	0.5773

Results for the optimized gradient boosting model for the testing set (final evaluation):

Class	Precision	Recall	F_1-score
0	0.9694	0.9774	0.9734
1	0.6145	0.5392	0.5744

Confusion matrix:

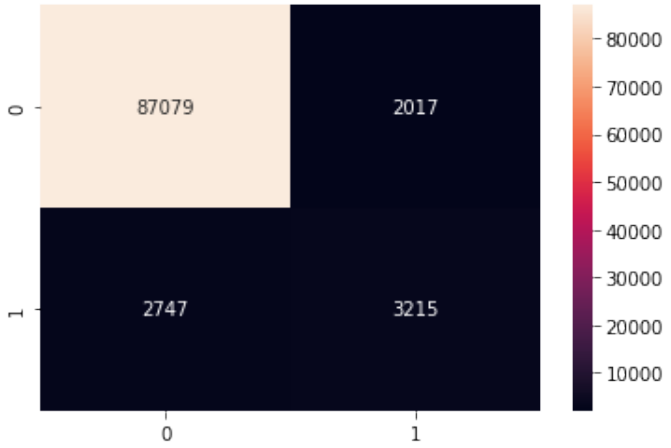


Figure 12: Confusion matrix for optimized gradient boosting (testing set).

Cross validation score variance was low for all of the models and the F_1 scores in the validation were aligned with the final testing results, indicating that none of the models overfit the training data. Optimized gradient boosting model yielded clearly best F_1-score for the positive prediction class, while also being the only one out of the three models with more correct than false positive predictions. Even though the parameter optimization for the gradient boosting resulted in a meaningful increase in the score, the model still does numerous misclassifications.

6 Summary

The objective of this thesis was to predict the need for hospitalization resulting from COVID-19 infections. For this purpose three candidate supervised learning models were selected, trained and tested. XGBoost, a gradient boosting model, was found to give the best predictions in terms of the F_1-score. Part of the reason for its superiority over the other two models may lie in the selection of the underlying model: medical symptom information as binary, often independent or intricately related features are more naturally represented as nodes of a decision tree rather than as coordinates in euclidean space.

Nonetheless, no model achieved positive class F_1-score greater than 0.5744. In other words, the models failed to find a clear pattern to correctly classify the need for hospitalization. A hint of this could already be seen in the PCA visualization, where the datapoints of different target variable class were heavily mixed together. Predicting hospitalizations emerged as a challenging machine learning problem, at least for the supervised learning models covered in this thesis. It may be, however, that the data itself does not always contain sufficient information to make the correct prediction.

Given data similar to the one used in this thesis and the aforementioned supervised learning methods, the resulting model always becomes a compromise between its specificity and sensitivity. This balancing then becomes an optimization problem of its own: an allowed level of misclassifications would have to be set for a similar model used in a health care organization. In real-world applications this would involve determining a cost for various available resources, which is out of the scope of this thesis.

While the question of this thesis proved itself a hard supervised learning task, the models, especially when optimized, still had some meaningful predictive power. Given this alongside with the examples from related studies, it can be concluded that machine learning models have the potential to be used as a predictive tool in various health care related applications.

References

[1] WHO COVID-19 Dashboard. Geneva: World Health Organization, 2021. Available online: https://covid19.who.int/ (Accessed Oct. 10, 2021)

[2] Mitchell, T., Machine Learning. New York, NY, USA: McGraw Hill, 1997. 414 p. ISBN 0-07-042807-7.

[3] Kushwaha, S., Bahl, S., Bagha, A.K., Parmar, K.S., Javaid, M., Haleem, A., Singh, R.P., "Significant Applications of Machine Learning for COVID-19 Pandemic", Journal of Industrial Integration and Management, Vol 5, No 4, pp. 453-479, Nov. 2020, doi: 10.1142/S2424862220500268.

[4] Google Flu Trends. Google.org, 2012. Available online (archived): https://web.archive.org/web/20121022154915/http://www.google.org/flutrends/about/how.html

[5] Cook, S., Conrad, C., Fowlkes, A.L., Mohebbi, M.H., "Assessing Google Flu Trends Performance in the United States during the 2009 Influenza Virus A (H1N1) Pandemic", PLOS ONE, Vol 6, No 8, Aug. 2011, doi: 10.1371/journal.pone.0023610

[6] Guo, S., Fang, F., Zhou, T., Zhang, W., Guo, Q., Zeng, R., Chen, X., Liu, J., Lu, X., "Improving Google Flu Trends for COVID-19 estimates using Weibo posts", Data Science and Management, Vol 3, pp. 13-21, Sept. 2021, doi: 10.1016/j.dsm.2021.07.001.

[7] Rustam, F., Reshi, A.A., Mehmood, A., Ullah, S., On, B., Aslam, W., Choi, G.S., "COVID-19 Future Forecasting Using Supervised Machine Learning Models", IEEE Access, Vol 8, pp. 101489-101499, 2020, doi: 10.1109/ACCESS.2020.2997311.

[8] Han, C.H., Kim, M., Kwak, J.T., "Semi-supervised learning for an improved diagnosis of COVID-19 in CT images", PLOS ONE, Vol 16, No 4, 2021, doi: 10.1371/journal.pone.0249450

[9] Muhammad, L.J., Algehyne, E.A., Usman, S.S., Ahmad, A., Chakraborty, C., Mohammed, I. A., "Supervised Machine Learning Models for Prediction of COVID-19 Infection using Epidemiology Dataset", SN COMPUT. SCI., Vol 2, No 11, 2021, doi: 10.1007/s42979-020-00394-7

[10] Centers for Disease Control and Prevention, COVID-19 Response. COVID-19 Case Surveillance Restricted Data Access, Summary, and Limitations. Available online: https://data.cdc.gov/Case-Surveillance/COVID-19-Case-Surveillance-Restricted-Access-Detai/mbd7-r32t (Accessed Oct.12, 2021)

[11] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R. and Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., "Scikit-learn: Machine Learning in Python", Journal of Machine Learning Research, Vol 12, pp. 2825-2830, 2011.

[12] Chen, T., Guestrin, C., "XGBoost: A Scalable Tree Boosting System", In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA: ACM, 2016. pp. 785–794. doi: 10.1145/2939672.2939785.

[13] Rubin, D. B., "Multiple Imputation for Nonresponse in Surveys" in Wiley Series in Probability and Statistics. USA: Wiley, 1987. 26 p. doi: 10.1002/9780470316696

[14] Little, R. J. A., Rubin, D. B., Statistical Analysis with Missing Data, Second Edition. Hoboken, NJ, USA: Wiley, 2002. 408 p. ISBN 978-1-119-01356-3

[15] Jung, A., "Machine Learning: The Basics," under preparation, 2021. available online at https://alexjungaalto.github.io/MLBasicsBook.pdf.

[16] Bottou, L., "Tradeoffs of Large Scale Learning" in Optimization for Machine Learning, Nowozin, S., Wright, S. J., Sra, S., Ed. Cambridge: MIT Press, 2012. pp. 351–368. ISBN 978-0-262-01646-9.

[17] Ritchie, H., Mathieu, E., Rodés-Guirao, L., Appel, C., Giattino, C., Ortiz-Ospina, E., Hasell, J., Macdonald, B., Beltekian, D., Roser, M., "Coronavirus Pandemic (COVID-19)". Our World in Data, 2020. Available online: https://ourworldindata.org/coronavirus

[18] Kuhn, M., Applied predictive modeling. New York, NY, USA: Springer, 2013. 600 p. ISBN 1-4614-6848-5.

[19] Agrawal, T., Hyperparameter Optimization in Machine Learning. Apress, 2021. 177 p. ISBN 1-4842-6579-3.

A Jupyter notebook of the analysis

The empirical analysis for this thesis was written entirely in Python. Multiple free, open-source libraries, such as Scikit-Learn, XGBoost, and Seaborn were used. The original data used in the analysis may not be provided due to CDC's terms of use. The CDC does not take responsibility for the scientific validity or accuracy of methodology, results, statistical analyses, or conclusions presented.

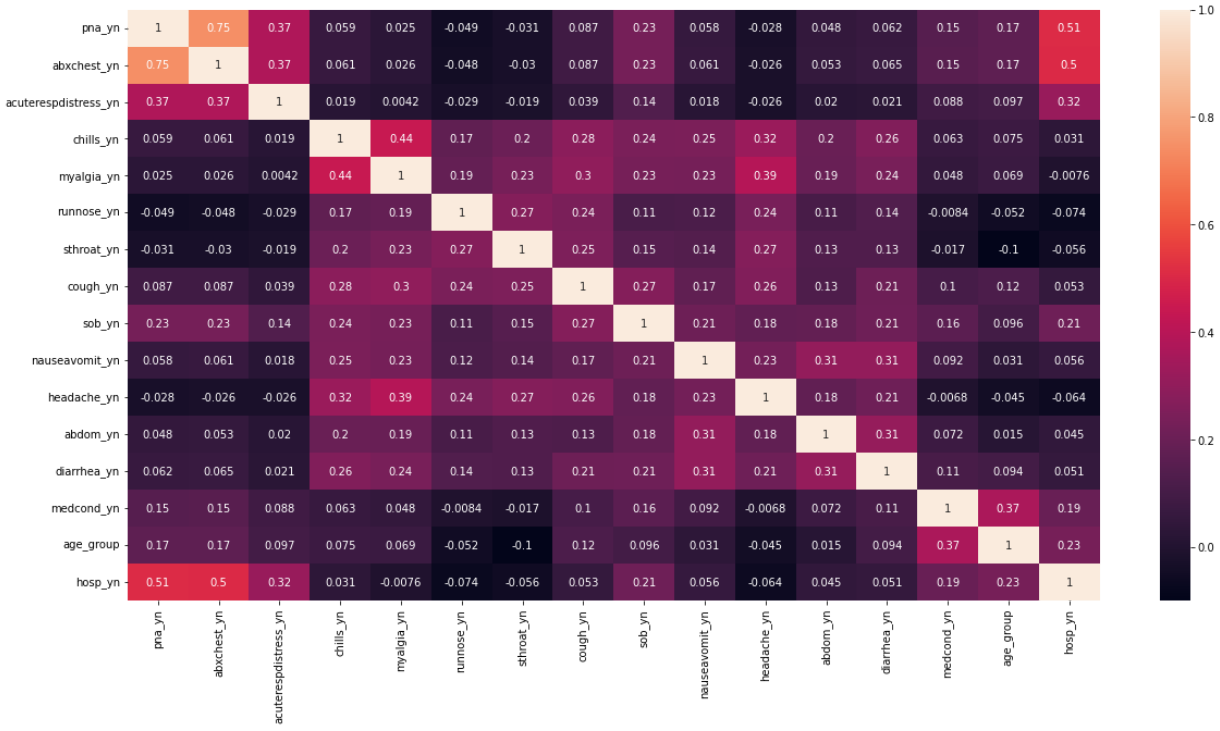
Initial data preprocessing and model fitting

0. Visualisation

Correlation matrix:

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
import numpy as np
from sklearn.linear_model import SGDClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
from xgboost import XGBClassifier
from sklearn.decomposition import PCA
from sklearn.model_selection import cross_val_score
from skopt import BayesSearchCV
df = pd.read_csv('preprocessed_data.csv')
#dropping the index column
df = df.iloc[: , 1:]

#Correlation matrix:
fig1 = plt.figure(1)
fig1.set_size_inches(20,10)
sb.heatmap(df.corr(), annot= True)
```

Out[1]:
```
<AxesSubplot:>
```



No major multicollinearity in the dataset, only pneumonia and abnormal chest x-ray have 0.75 correlation with one another.

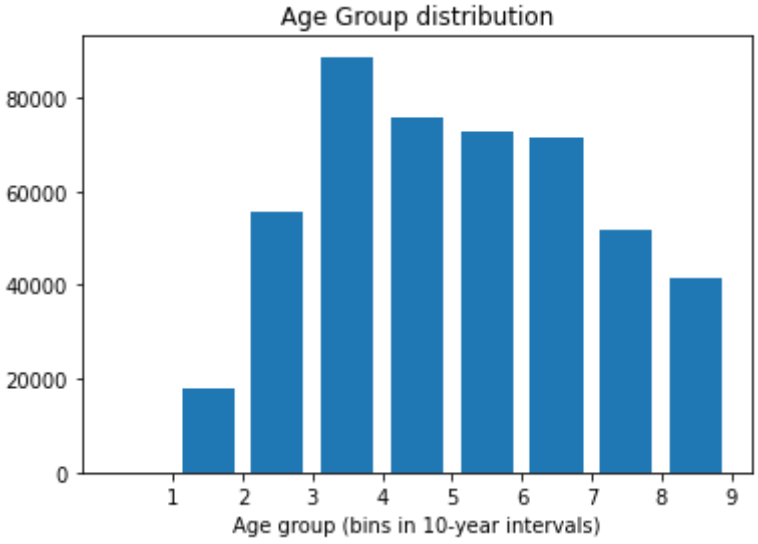
Age distribution in the dataset:

In [2]:
```
fig5 = plt.figure()
plt.title('Age Group distribution')
```

```
ax = plt.hist(df['age_group'], bins=np.arange((10)),rwidth=0.75)
plt.xlabel('Age group (bins in 10-year intervals)')
plt.xticks(range(1,10))
```

Out[2]:

```
([<matplotlib.axis.XTick at 0x2a6acb56820>,
  <matplotlib.axis.XTick at 0x2a6acb56160>,
  <matplotlib.axis.XTick at 0x2a6acb502e0>,
  <matplotlib.axis.XTick at 0x2a6ac4e04f0>,
  <matplotlib.axis.XTick at 0x2a6ac4e0c40>,
  <matplotlib.axis.XTick at 0x2a6ac4ea3d0>,
  <matplotlib.axis.XTick at 0x2a6ac4eab20>,
  <matplotlib.axis.XTick at 0x2a6ac4e07c0>,
  <matplotlib.axis.XTick at 0x2a6ac4ea430>],
 [Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, '')])
```

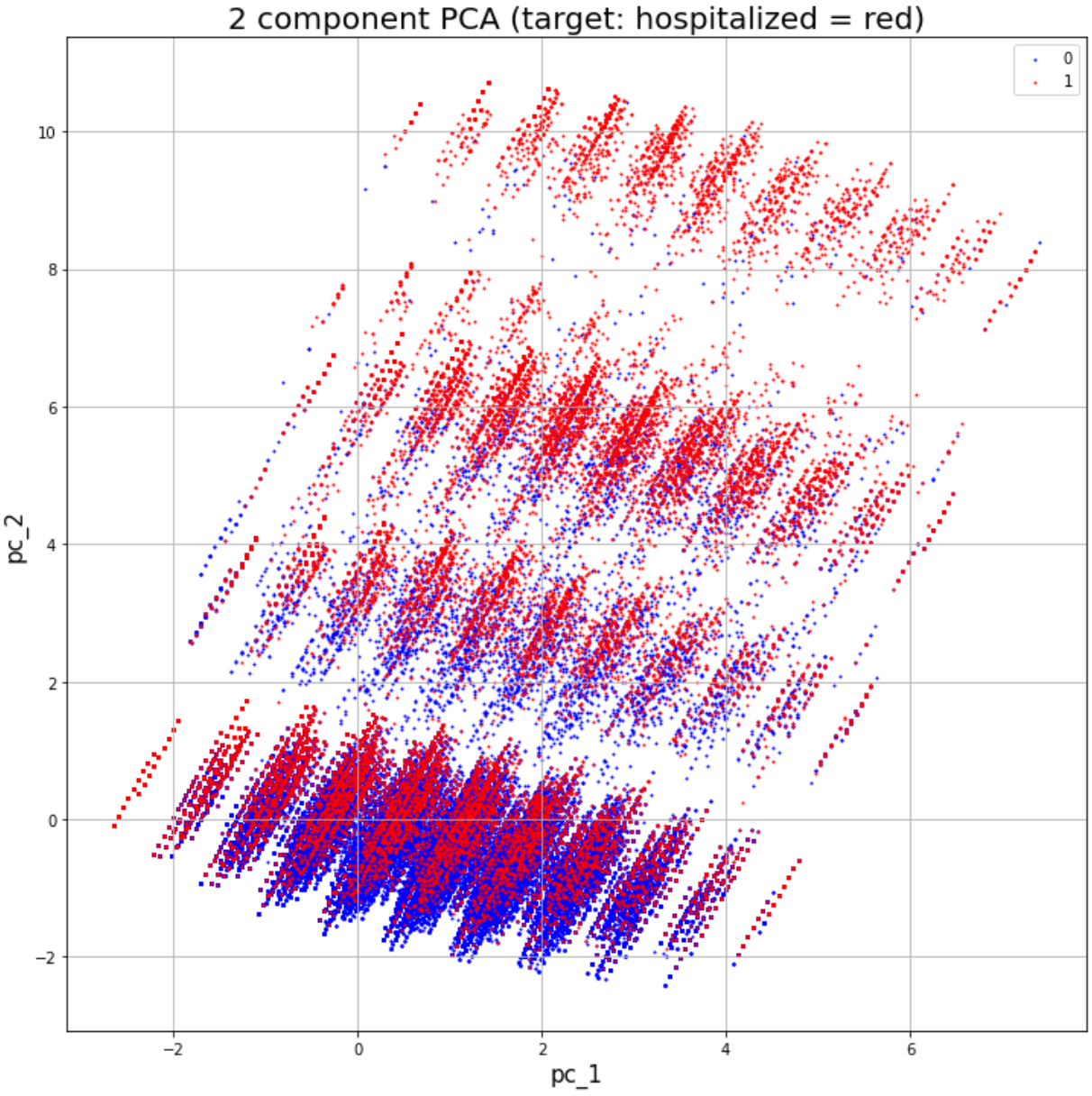


Visualisation with PCA:

In [3]:

```
columns = [
                        'pna_yn',
                        'abxchest_yn',
                        'acuterespdistress_yn',
                        'chills_yn',
                        'myalgia_yn',
                        'runnose_yn',
                        'sthroat_yn',
                        'cough_yn',
                        'sob_yn',
                        'nauseavomit_yn',
                        'headache_yn',
                        'abdom_yn',
                        'diarrhea_yn',
                        'medcond_yn',
                        'age_group'
]
x = df[columns]
y = df['hosp_yn']
x = StandardScaler().fit_transform(x)
pca = PCA(n_components=2)
```

```
principalComponents = pca.fit_transform(x)
principaldf = pd.DataFrame(data = principalComponents, columns = ['pc_1', 'pc_2'])
finaldf = pd.concat([principaldf, y], axis = 1)

fig = plt.figure(figsize = (12,12))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('pc_1', fontsize = 15)
ax.set_ylabel('pc_2', fontsize = 15)
ax.set_title('2 component PCA (target: hospitalized = red)', fontsize = 20)
targets = [0,1]
colors = ['b', 'r']
for target, color in zip(targets,colors):
    indicesToKeep = finaldf['hosp_yn'] == target
    ax.scatter(finaldf.loc[indicesToKeep, 'pc_1']
               , finaldf.loc[indicesToKeep, 'pc_2']
               , c = color
               , s = 1)
ax.legend(targets)
ax.grid()
```



Visualising data imbalance:

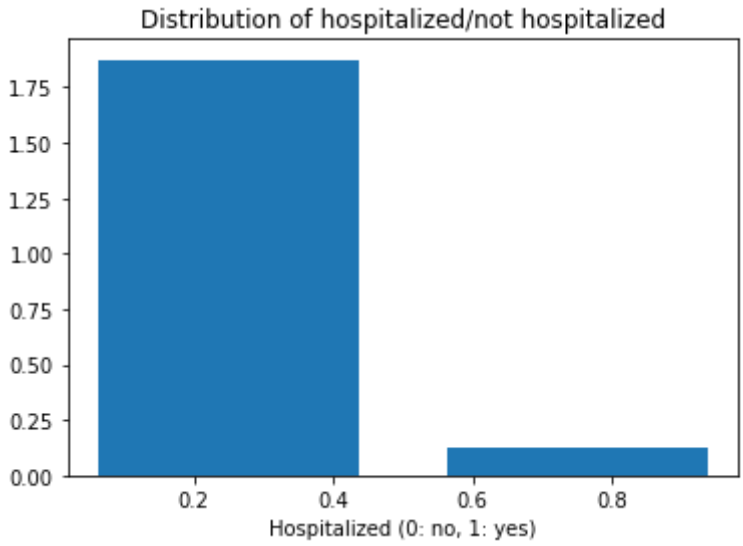
In [4]:

```
print('n_datapoints:',len(df.index))
print('n_hospitalized:',len(df[df['hosp_yn'] == 1].index))
```

```
fig6 = plt.figure()
plt.title('Distribution of hospitalized/not hospitalized')
ax = plt.hist(df['hosp_yn'],bins=2, rwidth=0.75,density=True)
plt.xlabel('Hospitalized (0: no, 1: yes)')
```

Out[4]:

```
n_datapoints: 475290
n_hospitalized: 29812
Text(0.5, 0, 'Hospitalized (0: no, 1: yes)')
```



Dealing with data imbalance: undersampling

Undersampling only makes the results worse...

In [5]:

```
#Splitting the data

features_all = df.drop(['hosp_yn'],1)
target_all = df['hosp_yn']

features_train, features_test, target_train, target_test = train_test_split(features
                                                    test_size = 0.2,
                                                    random_state = 2,
                                                    stratify = target_all)


# Standardization

scaler = StandardScaler()
#fit the scaler and transform the training set
features_train = scaler.fit_transform(features_train)

#transform the testing set with the same parameters used to transform the training s
features_test = scaler.transform(features_test)


''' Near miss undersampling algorithm implemented in the Imbalanced-learn library
#nm = imblearn.under_sampling.NearMiss(version=1)
#features_train, target_train = nm.fit_resample(features_train, target_train)

fig6 = plt.figure()
plt.title('Distribution after near miss undersampling (training set)')
ax = plt.hist(target_train,bins=2, rwidth=0.75,density=False)
plt.xlabel('Hospitalized (0: no, 1: yes)')
'''
```

```
C:\Users\Juho\AppData\Local\Temp/ipykernel_12056/973114146.py:3: FutureWarning: In a
future version of pandas all arguments of DataFrame.drop except for the argument 'la
bels' will be keyword-only
  features_all = df.drop(['hosp_yn'],1)
```

Out[5]:

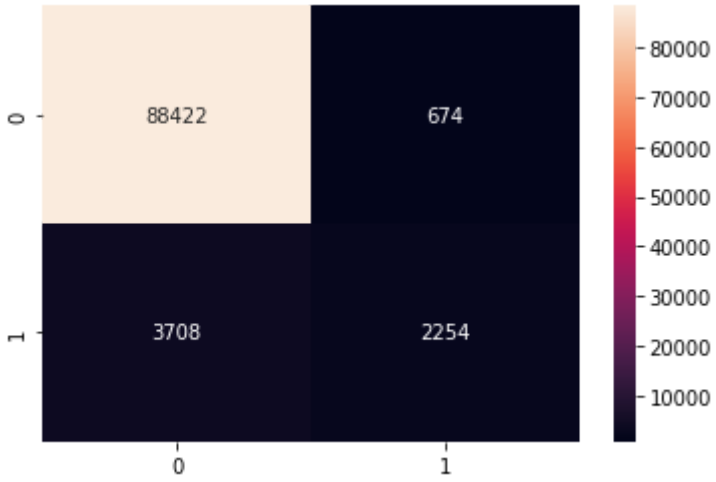
```
" Near miss undersampling algorithm implemented in the Imbalanced-learn library\n#nm
= imblearn.under_sampling.NearMiss(version=1)\n#features_train, target_train = nm.fi
t_resample(features_train, target_train)\n\nfig6 = plt.figure()\nplt.title('Distribu
tion after near miss undersampling (training set)')\nax = plt.hist(target_train,bins
=2, rwidth=0.75,density=False)\nplt.xlabel('Hospitalized (0: no, 1: yes)')\n"
```

Logistic Regression:

In [10]:

```
classifier = SGDClassifier(loss='log' ,max_iter=1000, tol=1e-3)

classifier.fit(features_train,target_train)

predicted = classifier.predict(features_test)

cm = confusion_matrix(target_test, predicted)

fig2 = plt.figure(2)
sb.heatmap(cm, annot=True,fmt='d')

print("Logistic Regression:")
print(classification_report(target_test, predicted,digits=4))


print(features_all.columns)
print("logistic regression feature weights:")
print(classifier.coef_)
```

```
Logistic Regression:
              precision    recall  f1-score   support

           0     0.9598    0.9924    0.9758     89096
           1     0.7698    0.3781    0.5071      5962

    accuracy                         0.9539     95058
   macro avg     0.8648    0.6852    0.7415     95058
weighted avg     0.9478    0.9539    0.9464     95058

Index(['pna_yn', 'abxchest_yn', 'acuterespdistress_yn', 'chills_yn',
       'myalgia_yn', 'runnose_yn', 'sthroat_yn', 'cough_yn', 'sob_yn',
       'nauseavomit_yn', 'headache_yn', 'abdom_yn', 'diarrhea_yn',
       'medcond_yn', 'age_group'],
      dtype='object')
logistic regression feature weights:
[[ 0.31563506  0.31284857  0.12305752  0.08274817 -0.07940118 -0.27378042
  -0.1236367  -0.09671626  0.44026245  0.09387491 -0.19395069  0.10690336
  -0.02968889  0.39965639  0.73141558]]
```



10-fold Cross validation for logistic regression:

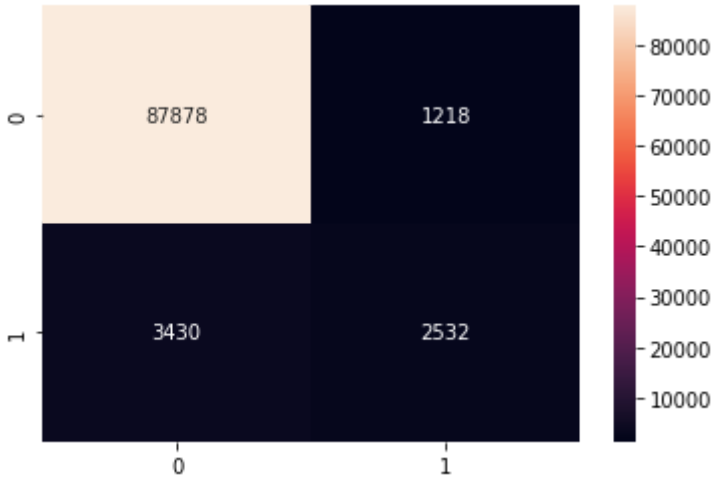
In [7]:
```
scores = cross_val_score(classifier, features_train, target_train, cv=10, scoring='f
print('The mean and variance of F_1 score in 10-fold CV:')
print("F_1 score: %0.3f, variance %0.5f" % (scores.mean(), scores.std() * 2))
```

```
The mean and variance of F_1 score in 10-fold CV:
F_1 score: 0.504, variance 0.05112
```

2. SVM:

In [8]:
```
classifier = SGDClassifier(loss='hinge' ,max_iter=1000, tol=1e-3)

classifier.fit(features_train,target_train)

predicted = classifier.predict(features_test)

cm = confusion_matrix(target_test, predicted)

fig3 = plt.figure(3)
sb.heatmap(cm, annot=True,fmt='d')
print("Support vector machines:")
print(classification_report(target_test, predicted,digits=4))
```

```
Support vector machines:
              precision    recall  f1-score   support

           0     0.9624    0.9863    0.9742     89096
           1     0.6752    0.4247    0.5214      5962

    accuracy                         0.9511     95058
   macro avg     0.8188    0.7055    0.7478     95058
weighted avg     0.9444    0.9511    0.9458     95058
```



10-fold CV for SVM classifier:

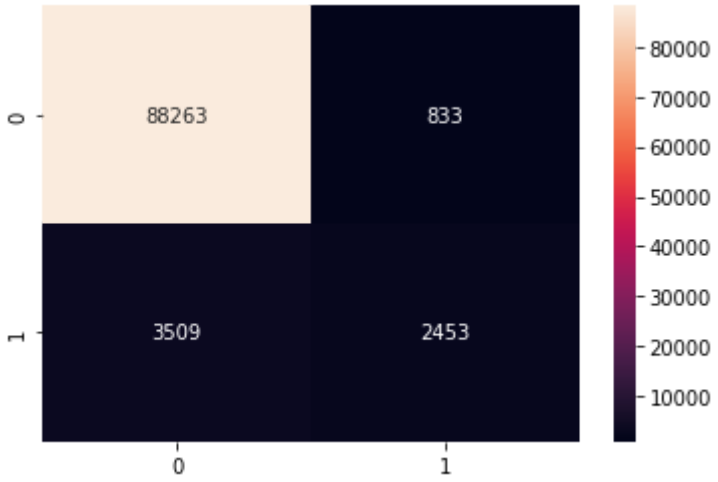
In [9]:
```
scores = cross_val_score(classifier, features_train, target_train, cv=10, scoring='f
print('The mean and variance of F_1 score in 10-fold CV:')
print("F_1 score: %0.3f, variance %0.5f" % (scores.mean(), scores.std() * 2))
```

```
The mean and variance of F_1 score in 10-fold CV:
F_1 score: 0.523, variance 0.02473
```

3. XGBoost

In [10]:
```
classifier = XGBClassifier(objective='binary:logistic',
                           booster='gbtree',
                           eval_metric='logloss',
                           tree_method='hist',
                           grow_policy='lossguide',
                           use_label_encoder=False)

classifier.fit(features_train,target_train)

predicted = classifier.predict(features_test)

cm = confusion_matrix(target_test, predicted)

fig4 = plt.figure(4)
sb.heatmap(cm, annot=True,fmt='d')
print("Gradient Boosting:")
print(classification_report(target_test, predicted,digits=4))
```

```
Gradient Boosting:
              precision    recall  f1-score   support

           0     0.9618    0.9907    0.9760     89096
           1     0.7465    0.4114    0.5305      5962

    accuracy                         0.9543     95058
   macro avg     0.8541    0.7010    0.7532     95058
weighted avg     0.9483    0.9543    0.9481     95058
```



10-fold CV for XGBoost:

In [11]:
```
scores = cross_val_score(classifier, features_train, target_train, cv=10, scoring='f
print('The mean and variance of F_1 score in 10-fold CV:')
print("F_1 score: %0.3f, variance %0.5f" % (scores.mean(), scores.std() * 2))
```

```
The mean and variance of F_1 score in 10-fold CV:
F_1 score: 0.532, variance 0.01405
```

Parameter tuning for XGboost:

In [12]:
```
param_grid = {'gamma': [0,0.1,0.2,0.4,0.8,1.6,3.2,6.4,12.8,25.6,51.2,102.4, 200],
              'learning_rate': [0.01, 0.03, 0.06, 0.1, 0.15, 0.2, 0.25, 0.3, 0.4],
              'max_depth': [6,7,8,9,10,11,12,13],
              'n_estimators': [50,65,80,100,115,130],
              'reg_alpha': [0,0.1,0.2,0.4,0.8,1.6,3.2,6.4,12.8,25.6,51.2,102.4,200],
              'reg_lambda': [0,0.1,0.2,0.4,0.8,1.6,3.2,6.4,12.8,25.6,51.2,102.4,200]
              'scale_pos_weight':[1,2,3,4,5,6]}

gcvj = np.cumsum([len(x) for x in param_grid.values()])[-1]

clf = BayesSearchCV(estimator=classifier, search_spaces=param_grid, n_iter=int(gcvj)
clf.fit(features_train, target_train)
```

```
Fitting 3 folds for each of 1 candidates, totalling 3 fits
[CV 1/3] END gamma=6.4, learning_rate=0.25, max_depth=6, n_estimators=80, reg_alpha=
12.8, reg_lambda=6.4, scale_pos_weight=2;, score=(train=0.571, test=0.566) total tim
e=   0.6s
[CV 2/3] END gamma=6.4, learning_rate=0.25, max_depth=6, n_estimators=80, reg_alpha=
12.8, reg_lambda=6.4, scale_pos_weight=2;, score=(train=0.573, test=0.565) total tim
e=   0.6s
[CV 3/3] END gamma=6.4, learning_rate=0.25, max_depth=6, n_estimators=80, reg_alpha=
12.8, reg_lambda=6.4, scale_pos_weight=2;, score=(train=0.571, test=0.571) total tim
e=   0.6s
Fitting 3 folds for each of 1 candidates, totalling 3 fits
[CV 1/3] END gamma=12.8, learning_rate=0.3, max_depth=10, n_estimators=50, reg_alpha
=102.4, reg_lambda=3.2, scale_pos_weight=4;, score=(train=0.564, test=0.565) total t
ime=   0.4s
[CV 2/3] END gamma=12.8, learning_rate=0.3, max_depth=10, n_estimators=50, reg_alpha
=102.4, reg_lambda=3.2, scale_pos_weight=4;, score=(train=0.567, test=0.562) total t
ime=   0.4s
[CV 3/3] END gamma=12.8, learning_rate=0.3, max_depth=10, n_estimators=50, reg_alpha
=102.4, reg_lambda=3.2, scale_pos_weight=4;, score=(train=0.565, test=0.564) total t
ime=   0.4s
Fitting 3 folds for each of 1 candidates, totalling 3 fits
[CV 1/3] END gamma=3.2, learning_rate=0.01, max_depth=9, n_estimators=65, reg_alpha=
```

```
=0.4, reg_lambda=0.1, scale_pos_weight=4;, score=(train=0.572, test=0.569) total tim
e=   1.0s
[CV 2/3] END gamma=0.2, learning_rate=0.03, max_depth=7, n_estimators=100, reg_alpha
=0.4, reg_lambda=0.1, scale_pos_weight=4;, score=(train=0.575, test=0.562) total tim
e=   1.1s
[CV 3/3] END gamma=0.2, learning_rate=0.03, max_depth=7, n_estimators=100, reg_alpha
=0.4, reg_lambda=0.1, scale_pos_weight=4;, score=(train=0.572, test=0.565) total tim
e=   1.1s
Fitting 3 folds for each of 1 candidates, totalling 3 fits
[CV 1/3] END gamma=200.0, learning_rate=0.06, max_depth=8, n_estimators=130, reg_alp
ha=0.1, reg_lambda=0.8, scale_pos_weight=3;, score=(train=0.568, test=0.571) total t
ime=   1.9s
[CV 2/3] END gamma=200.0, learning_rate=0.06, max_depth=8, n_estimators=130, reg_alp
ha=0.1, reg_lambda=0.8, scale_pos_weight=3;, score=(train=0.571, test=0.565) total t
ime=   1.8s
[CV 3/3] END gamma=200.0, learning_rate=0.06, max_depth=8, n_estimators=130, reg_alp
ha=0.1, reg_lambda=0.8, scale_pos_weight=3;, score=(train=0.569, test=0.569) total t
ime=   1.8s
```

Out[12]:

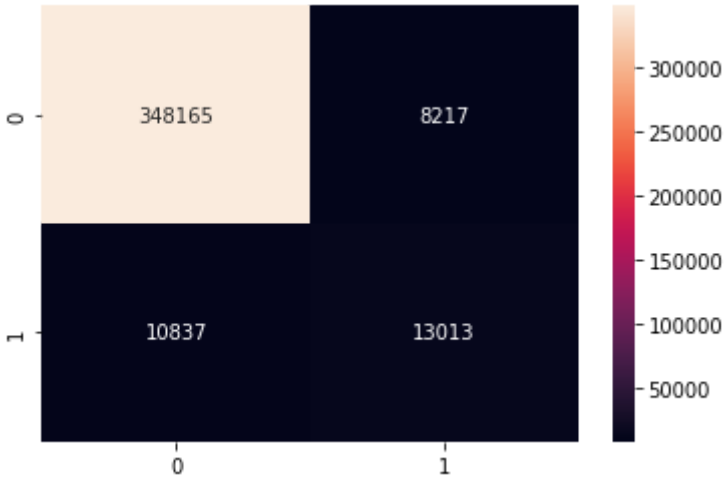
```
BayesSearchCV(cv=3,
              estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                      colsample_bylevel=1, colsample_bynode=1,
                                      colsample_bytree=1, eval_metric='logloss',
                                      gamma=0, gpu_id=-1,
                                      grow_policy='lossguide',
                                      importance_type='gain',
                                      interaction_constraints='',
                                      learning_rate=0.300000012,
                                      max_delta_step=0, max_depth=6,
                                      min_child_weight=1, missing=nan,
                                      monotone_constrai...
              search_spaces={'gamma': [0, 0.1, 0.2, 0.4, 0.8, 1.6, 3.2, 6.4,
                                       12.8, 25.6, 51.2, 102.4, 200],
                             'learning_rate': [0.01, 0.03, 0.06, 0.1, 0.15, 0.2,
                                               0.25, 0.3, 0.4],
                             'max_depth': [6, 7, 8, 9, 10, 11, 12, 13],
                             'n_estimators': [50, 65, 80, 100, 115, 130],
                             'reg_alpha': [0, 0.1, 0.2, 0.4, 0.8, 1.6, 3.2, 6.4,
                                           12.8, 25.6, 51.2, 102.4, 200],
                             'reg_lambda': [0, 0.1, 0.2, 0.4, 0.8, 1.6, 3.2,
                                            6.4, 12.8, 25.6, 51.2, 102.4, 200],
                             'scale_pos_weight': [1, 2, 3, 4, 5, 6]},
              verbose=3)
```

In [17]:

```
print("Best F_1-score in optimization: ",clf.best_score_ )

predicted = clf.predict(features_train)

cm = confusion_matrix(target_train, predicted)

fig4 = plt.figure(6)
sb.heatmap(cm, annot=True,fmt='d')
print("Gradient Boosting, optimized (training):")
print(classification_report(target_train, predicted,digits=4))
```

```
Best F_1-score in optimization:  0.5716821331432825
Gradient Boosting, optimized (training):
              precision    recall  f1-score   support

           0     0.9698    0.9769    0.9734    356382
           1     0.6130    0.5456    0.5773     23850

    accuracy                         0.9499    380232
   macro avg     0.7914    0.7613    0.7753    380232
```

```
weighted avg     0.9474    0.9499    0.9485    380232
```



In [15]:

```
predicted = clf.predict(features_test)

cm = confusion_matrix(target_test, predicted)

fig5 = plt.figure(7)
sb.heatmap(cm, annot=True,fmt='d')
print("Gradient Boosting, optimized (test):")
print(classification_report(target_test, predicted,digits=4))
```

```
Gradient Boosting, optimized (test):
              precision    recall  f1-score   support

           0     0.9694    0.9774    0.9734     89096
           1     0.6145    0.5392    0.5744      5962

    accuracy                         0.9499     95058
   macro avg     0.7920    0.7583    0.7739     95058
weighted avg     0.9472    0.9499    0.9484     95058
```