A Comparison of Supervised ML Algorithms Across Multiple Metric Scores

Lulu Ricketts
UC San Diego, COGS Department
`lrickett@ucsd.edu`

Abstract

Supervised machine learning algorithms have been used widely across many applications, from stock price prediction to image segmentation and many others. Though they've proven to be useful solutions to their applications, it is often difficult to gauge which algorithms will work best for what types of problems, and little work has been done to showcase the relative effectiveness of different algorithms. In this paper, I compare performance of multiple machine learning algorithms - Logistic Regression, SVM, Random Forests, and Artificial Neural Networks - on different datasets. Through analyzing performances based on different metrics and results from 5-fold cross validation grid search through hyperparameter space, I find which algorithms are best suited for certain problem types, as well as the metrics they perform best at.

1 Introduction

For the past couple decades, supervised machine learning has made impressive strides in industry and research. The ability of these algorithms to separate classes in distinct ways revolutionized many fields by streamlining processes and problems from those that could have taken weeks to those that could take only hours. Instead of the need for hard coding all edge cases and rules for a given program, it is now possible, thanks to machine learning and supervised learning algorithms, to simply give the computer an input and desired output, allowing it to learn patterns on its own [1]. By doing so, data scientists and other data analysts are given more time to make conclusions and insights that have significant impact on societal processes. These new methodologies for training algorithms are further expedited by the shift towards increasing numbers of publicly available datasets, allowing scientists to combine information to draw new, meaningful conclusions that were previously impossible. For instance, combining data from two vastly different industries to find similarities. However, with this emergence of new data and techniques for analyzing them, uncertainty within methods themselves arise.

The field of data science has been rapidly transforming into a complex industry, where the choices of data used, cleaning, processing, model selection, hyperparameter tuning, and error metrics are highly variable. It is common to see that most data scientists choose different approaches to the same problem, often landing them at different conclusions. But which approach is really best? One issue, as described by Handelman et al [2], is that some data analysts are not fully equipped with the knowledge on how to evaluate the goodness of their model, or even lack knowledge of what metrics are best used for which situations. They describe instances where researches will throw in p values to claim significance, without really understanding what that entails. In the reality of some situations, other error metrics can prove significance of a study much better. It is thus critical to ensure data scientists fully understand their approach, and what their results mean in context of their study or industry. For example, achieving a higher precision score in one industry, such as medicine, may not be as important as achieving a higher precision in another industry, such as in self-driving vehicles.

Scientists not only be able to then maximize the score for the problem they need, but must understand which algorithms and techniques will help them to achieve that score.

To gain this understanding, there must be a way to compare and contrast these algorithms across multiple industry problems. Unfortunately, running an exhaustive comparison of machine learning algorithms is extremely costly and time consuming, so few have been able to truly extract the fundamental differences in each model. Among those that have ran comparisons is Sathya and Abraham [3], whose comparisons of both supervised and unsupervised algorithms in the context of higher education. However, their comparison was limited to that of artificial neural networks (supervised) and KSOM (unsupervised), and is ungeneralizable to other machine learning architectures. On the other hand, Caruana et al [4] ran a large scale study of supervised learning architectures across multiple datasets and metrics, offering one of the most comprehensive comparison of how each model performed in different tasks. In this paper, I seek to replicate the methods used in their paper (on a slightly smaller scale) in hopes that I will be able to draw conclusions to determine the optimality of different models.

2 Methodology

2.1 Learning Algorithms

Supervised learning algorithms are highly dependent on the hyperparameters used to train them. For instance, too high of a learning rate for neural networks may ensure they never converge to the optimal values, while too small of a learning rate may cause the algorithm to train much longer before convergence. To avoid suboptimal performance in the comparisons for this paper, I conduct a grid search throughout the hyperparameter space for each model type to ensure the results obtained are the best possible for each model. This allows for a more accurate comparison across models. This section describes each model tested in this paper, as well as the hyperparameters associated with them. All models used are from sci-kit learn's library.

Logistic Regression I train multiple logistic regression models with regularization, modifying the level of regularization (C) from 10^{-8} to 10^{4}

SVM For support vector machines, all training instances were limited to 2000 iterations to decrease training time. I train using linear, poly, and radial basis (rbf) kernels, varying the degree of poly kernels to degree 2 and 3. Regularization strength was trained from values of 10^{-7} to 10^{4}, and gamma parameter for rbf and poly kernels was trained on values of $10^{-}6$ to 10^{-1}.

Random Forest In training random forest models, the number of estimators (number of trees) was kept constant at 100. The hyperparameter search consisted of varying the max depth $[2, 4, 6, 8, 16, 20, 30, None]$ and max features $[1, 2, 4, 6, 8, 12, 16, 20]$.

Artificial Neural Network For ANNs, constant hyperparameters during all training iterations were the solver (stochastic gradient descent) and the learning rate ("adaptive"). I vary the number of hidden layers $[1, 2, 5, 7]$ and hidden layer sizes $[50, 100, 200]$. I also varied the momentum for gradient descent $[0, 0.2, 0.5, 0.9]$.

2.2 Performance Metrics

It is important to clearly understand the decipher the differences in evaluation metrics of models. Simply achieving a high training accuracy does not necessarily mean the model is generalizable to outside data [2], so it is therefore imperitive to not only handle model selection well, but to evaluate the model's performance meaninfully.

In this paper, I test each model on five performance metrics: accuracy, F1 score, ROC auc, precision, and recall. In the context of the classification methods I am using, these metrics, while related, provide distinctly different information. The accuracy score, perhaps the most common metric, simply gives the number of samples predicted correctly in proportion to the number of samples total. While useful, this metric does not give any information on class-specific performance. If data samples are unbalanced, for instance if class 0 has 100 samples and class 1 has 10, an accuracy score of 90% may seem great, but in reality it could be only predicting class 0 for every sample.

Precision and recall scores, however, can give class specific performance information by measuring the number of true positives in relation to all positive samples and the proportion of false positives to negative samples, respectively. Because these two can be highly different and represent opposite sides of the metric spectrum, the F1 score is used as a middle ground, which combines the two scores. Finally, the ROC auc metric computes the area under the curve comparing the false positive to true positive rates. Used together, I believe these five metrics will provide a holistic evaluation of the models.

2.3 Data Sets

Each algorithm, along with their sets of hyperparameters, was trained on a binary classification problem using four different datasets downloaded from Kaggle (see appendix A for links to data sources). I chose to stray from Caruana's [4] choice of datasets in their comparison because I wanted to further explore novel datasets that had not been tested in the past. Given Kaggle's reputability and abundance of large public datasets, I believe it was a good platform to borrow data from. The datasets chosen were airline satisfaction, adult income prediction, phishing website detector, and a surgical complication prediction, henceafter referred to as AIRLINE, INCOME, PHISH, SURG. It is worth noting that the INCOME dataset is very similar to that of ADULT used in Caruana's paper [4], where the targets are whether an adult makes <50k or >50k a year. Furthermore, I chose these four datasets because I believe they each represent different applications of data science methodology, including surveyed data (AIRLINE), anomaly detection (PHISH), and medical (SURG) and economical (INCOME) applications. See table 1 for additional information on each dataset.

In terms of preprocessing, most attributes/features for each dataset was used. Only columns that were unmeaninful (ie. ID) were dropped, with the exception of the column "native country" dropped in INCOME for the reason that values were overwhelmingly (>90%) "USA". For all other categorical features, they were one hot encoded using pandas get dummies function.

PROBLEM	NUM ATTR	TRAIN SIZE	TEST SIZE	%POZ
AIRLINE	22	5000	98594	56%
INCOME	63	5000	36451	75%
PHISH	30	5000	6054	44%
SURG	24	5000	9635	74%

Table 1. Description of problems

3 Experimental Procedure

3.1 Procedure

All procedural code can be found in Appendix C

For each dataset, training and testing sets were split the same way for each. Each training set comprised of 5000 samples, while the test set consisted of all the samples that were not in the training set. This was done by bootstrapping (random sampling) 5000 samples with replacement. Because of this, the training set size was always constant, but the test set size could vary slightly depending on how many samples were repeated in the training set. However, this wasn't much of an issue because each dataset had well over (>5000) samples allocated to the test set each time.

A function was created for each one of the algorithms to run it. For these functions, the parameters they took in were X and Y, the complete set of samples and labels prior to splitting the data. Five trials were run for each call of the function, where the procedure was as follows. 5000 samples were bootstrapped to be the training set, while the rest were allocated to the test set. For each trial, a grid search through the hyperparameters was run with sci-kit learn's built in GridSearchCV function, which automatically performs 5-fold cross validation on the training set, returning all results. During the grid search, the GridSearchCV function was given the five metrics to evaluate (accuracy, f1, roc, precision, recall), which were returned at the end of the grid search. Then, for each metric, I collected the set of hyperparameters that resulted in the highest value for that metric only. Those hyperparameters were used to get predictions for both the training and testing sets, and saved to a list.

Note that it is highly possible that different metrics had different hyperparameters that optimized their values, but this was unimportant for this study because I am comparing the algorithms themselves, not their properties. At the end of all five trials, The mean of each metric across all five trials was taken and returned for both the training and testing performances.

Additionally, the functions described above also saved the hyperparameters for a single trial of grid search for each algorithm/dataset combination. This was done for all algorithms except for SVM, which I left out because of the complicated nature of the hyperparameter space. While other algorithms had at most 2 hyperparameters that varied, SVM had more with more complications (for instance, degree was only used with poly kernel), which I decided would be too difficult to visualize in an effective manner. Through visualizing the grid search process and their successive results during 5-fold cross validation, we are able to see clearly how increasing or decreasing one of the parameters has an effect on the algorithm itself. These visualizations can be found in Appendix B.

These functions were run for each dataset in a notebook, saving results from all datasets for each algorithm in a single .csv file.

3.2 Performance by Problem

For each dataset/algorithm combination, the mean training and testing scores from running the model across all five trials of k-fold cross validation grid search was saved. Table 2 shows the mean performance of each dataset for each algorithm across all five metrics, where Table 2.1 shows the performance of the testing set, whilst 2.2 for the training set. The mean performance of each algorithm across all datasets is also calculated.

In these tables, **bold faced** values represent the algorithm that performed the best for each dataset. Curiously, random forests performed the best across the board. Values marked with a * represent values that are not significantly different than the best-performing algorithm over the metrics at a significance set at $p = 0.05$. Unmarked values represent those that are significantly different than the best-performing algorithm.

	AIRLINE	INCOME	PHISH	SURG	MEAN
LogReg	0.7843	0.5547*	0.9421*	0.5630	0.7111
SVM	0.7009	0.5203	0.9601*	0.7443*	0.7314
RF	**0.9385**	**0.7770**	**0.9673**	**0.8170**	**0.8749**
ANN	0.6977	0.7003*	0.9526*	0.5780	0.7322

Table 2.1. Test set performance by dataset for each algorithm

	AIRLINE	INCOME	PHISH	SURG	MEAN
LogReg	0.7853	0.5618	0.9452	0.5811	0.71839
SVM	0.8003	0.7333*	0.9893*	0.8539*	0.8442
RF	**0.9982**	**0.9809**	**0.9926**	**0.9961**	**0.9919**
ANN	0.6996	0.7115	0.9913*	0.6160	0.7546

Table 2.2. Training set performance by dataset for each algorithm

3.3 Performance by Metric

For each metric/algorithm combination, the mean training and testing scores from running the model across all five trials of k-fold cross validation grid search was saved. Table 3 shows the mean performance of each metric for each algorithm across all four datasets, where Table 3.1 shows the performance of the testing set, whilst 3.2 for the training set. The mean performance of each algorithm across all metrics is also calculated.

In these tables, **bold faced** values represent the metric that performed the best for each algorithm. As before, values marked with a * represent values that are not significantly different than the best-performing algorithm over the datasets at a significance set at $p = 0.05$. Unmarked values represent those that are significantly different than the best-performing algorithm.

	ACC	F1	ROC	PRECISION	RECALL	MEAN
LogReg	0.8259*	0.626*	0.7396*	0.7696	0.5942*	0.7111
SVM	0.8294*	0.654*	0.6735*	0.7713*	0.729*	0.7314
RF	**0.9125**	**0.8326**	**0.87**	**0.9574**	**0.8023**	**0.8749**
ANN	0.826*	0.6805*	0.7641*	0.7672	0.6229*	0.7322

Table 3.1. Test set performance by metric for each algorithm

	ACC	F1	ROC	PRECISION	RECALL	MEAN
LogReg	0.8289	0.6309	0.7426	0.7926	0.597	0.7184
SVM	0.9296*	0.7212*	0.7704*	**0.9975**	0.8025*	0.8442
RF	**0.9931**	**0.9982**	**0.9791**	0.9948*	**0.9947**	**0.992**
ANN	0.8428	0.7088	0.7854	0.7908	0.6455	0.7546

Table 3.2. Training set performance by metric for each algorithm

4 Discussion

In this paper, I conducted a study of four different algorithms on four different datasets whilst evaluating their performance on five different metrics. The results overwhelmingly suggest that random forest classifier outperformed all other classifiers on all training and testing sets. On the one instance where random forest did not perform the best and was outperformed by SVM, the resulting training set performance is not statistically different than that of SVM. This suggests that there is something about random forests that makes it an optimal classifier for binary classification problems across the board. Random forests differs from the other algorithms used - logistic regression, support vector machine, and neural networks - in that it is an ensemble classifier. While the other algorithms aim to find a linear or nonlinear boundary, random forests is unique in that it can find the most complex boundaries between classes. It does so by averaging the results of many mini trees trained on only a few of the features. This perhaps is why it is a powerful classifier (wisdom of the crowd). By taking the maximum class predicted from the forest, it has the ability to generalize to many different problems and performs well on all metrics. In the future, it could be worth exploring how the other algorithms, when used in ensemble classification, can improve testing metrics.

Both training and testing set metrics are included in tables 2 and 3, for the reason being that we can view how each model is able to generalize to the test set after being trained on the training set. In general, training set performance exceeds test set performance, which is to be expected. The algorithms that generalize the best are logistic regression and neural networks, while SVM generalizes slightly worse and random forest is highly variable in its generalizability. It is interesting to note that although random forests attains the highest metrics in most cases, it may be more worthwhile to work on improving metrics on other algorithms that do generalize well.

In my calculation of p values for significance testing, I was unable to collect raw scores from each trial, as I had already completed running all algorithms while thinking that p values would be calculated after taking the mean across trials. Thus, the significance tests star values in which the mean across metrics (table 2) or datasets (table 3) are significantly different. This methodology differs from Caruana's [4] paper and thus may yield the exact same significance of metric scores.

In appendix B, I include heatmap visualizations of the hyperparameter grid search results for logistic regression, random forest, and neural networks. In logistic regression, three metrics (accuracy, f1, and roc) are included, whereas for random forest and neural networks, only accuracy heatmaps are shown. For logistic regression, we can see the general trend that all metrics tend to increase for all datasets as the parameter C increases as well, suggesting stronger regularization yields better results. For random forests, we see that as both the depth and features used in the forest increases, accuracy tends to increase as well. Here, the value "nan" for maximum depth refers to no maximum depth specified, in which the trees in the forest grow as large as they need to. Finally, artificial neural network's hyperparameter search seems more variable from dataset to dataset. While momentum = 0.9 tends to yield the highest result, the hidden layer sizes are more variable and thus we can theorize that the number of optimal hidden layers and their sizes vary from problem to problem. Therefore, neural networks most likely need more hands on fine tuning than other models might.

5 Bonus Points

I feel that my analysis in this paper deserves bonus points because I included metrics on both the training and testing sets, as well as offered comparisons as to why this is important. I also included heatmap visualizations of the hyperparameter grid search in the appendix and explained results in the discussion portion.

6 References

[1] Jordan, M. I., Mitchell, T. M. (2015). Machine Learning: Trends, Perspectives, and Prospects. *Science*, 349(6245), 255-260.

[2] Handelman, G. S., Kok, H. K., Chandra, R. V., Razavi, A. H., Huang, S., Brooks, M., ... Asadi, H. (2019). Peering Into the Black Box of Artificial Intelligence: Evaluation Metrics of Machine Learning Methods. *American Journal of Roentgenology*, 212(1), 38-43.

[3] Sathya, R., Abraham, A. (2013). Comparison of Supervised and Unsupervised Learning Algorithms for Pattern Classification. *International Journal of Advanced Research in Artificial Intelligence*, 2(2), 34-38.

[4] Caruana, R., Niculescu-Mizil, A. (2006, June). An Empirical Comparison of Supervised Learning Algorithms. In Proceedings of the 23rd International Conference on Machine Learning (pp. 161-168).

7 Appendix

7.1 Appendix A: Data Set Sources

AIRLINE: https://www.kaggle.com/teejmahal20/airline-passenger-satisfaction

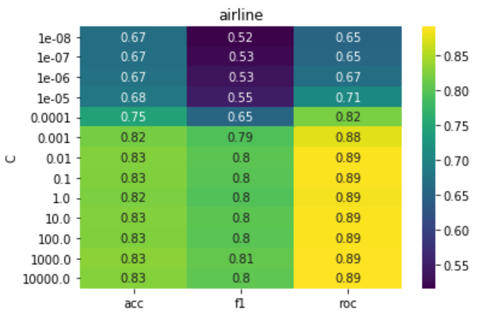
INCOME: https://www.kaggle.com/mastmustu/income/

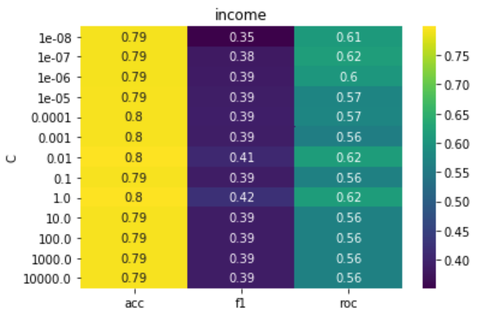
PHISH: https://www.kaggle.com/eswarchandt/phishing-website-detector

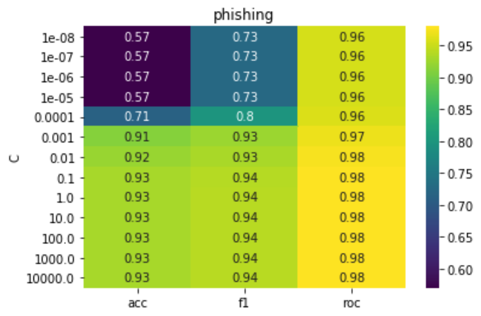
SURG: https://www.kaggle.com/omnamahshivai/surgical-dataset-binary-classification

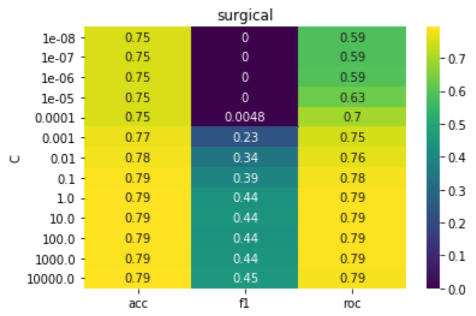
7.2 Appendix B: Grid Search Hyperparameter Results

Logistic Regression

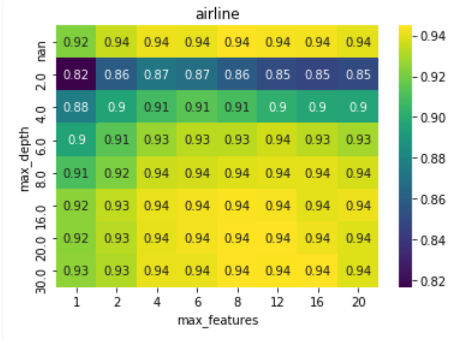


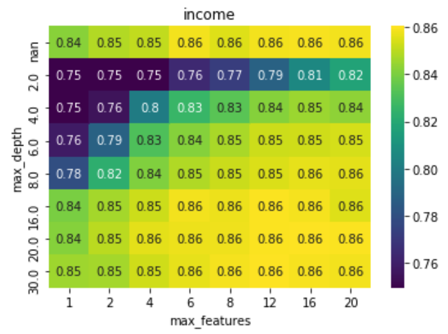


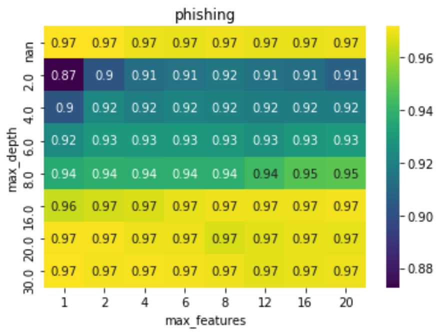


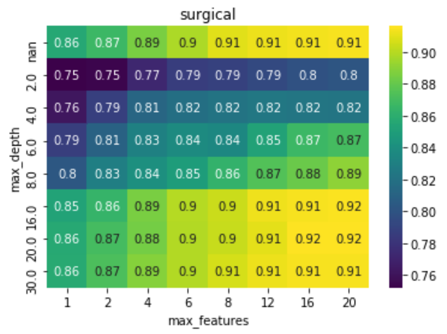


Random Forest

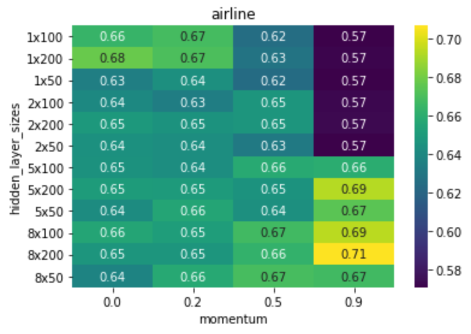


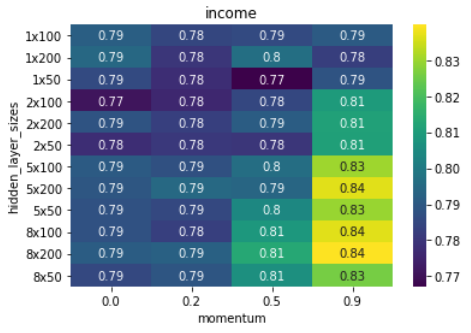


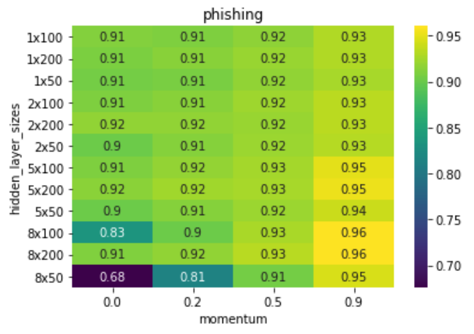


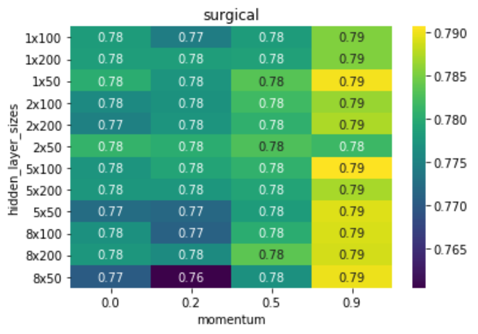


Artificial Neural Network









7.3 Appendix C: Code

```python
# Gets the feature and target vectors for each dataset

# imports
import pandas as pd
import numpy as np

# get airlines features/targets
def prep_airlines():

    data_dir = '../data/airline_satisfaction/'
    df = pd.read_csv(data_dir + 'train.csv', index_col=0)
    df.drop('id', axis=1, inplace=True)
    df.dropna(inplace=True)

    to_encode = ['Gender', 'Customer Type', 'Type of Travel', 'Class', 
'satisfaction']
    cleanup = dict()
    for col in to_encode:
        if col == 'satisfaction':
            cleanup[col] = {k:v for k,v in zip(df[col].unique(), [0,1])}
        elif len(df[col].unique()) == 2:
            cleanup[col] = {k:v for k,v in zip(df[col].unique(), [-1,1])}
        elif len(df[col].unique()) == 3:
            cleanup[col] = {k:v for k,v in zip(df[col].unique(), [-1,0,1])}

    df = df.replace(cleanup)

    X = np.array(df[df.columns[:-1]])
    y = np.array(df[df.columns[-1]])

    return X,y

# get incomes features/targets
def prep_income():

    data_dir = '../data/income/'
    df = pd.read_csv(data_dir + 'train.csv')

    drop = ['native-country']
    df.drop(drop, axis=1, inplace=True)
    df.dropna(inplace=True)

    onehot_cols = [col for col in df.columns if df[col].dtype == 'object']
    df = pd.get_dummies(data=df, columns=onehot_cols)

    X = np.array(df[df.columns[df.columns != 'income_>50K']])
    y = np.array(df['income_>50K'])

    return X,y
```

```python
# get phishing websites features/targets
def prep_phishing():

    data_dir = '../data/phishing_website/'
    df = pd.read_csv(data_dir + 'phishing.csv.xls', index_col=0)

    df['class'] = df['class'].map({-1:0, 1:1})

    X = np.array(df[df.columns[:-1]])
    y = np.array(df[df.columns[-1]])

    return X,y

# get surgical complications features/targets
def prep_surgical():

    data_dir = '../data/surgical_complications/'
    df = pd.read_csv(data_dir + 'Surgical-deepnet.csv')

    X = np.array(df[df.columns[:-1]])
    y = np.array(df[df.columns[-1]])

    return X,y
```

```python
# Bootstraps 5000 samples for training set and rest to test set

import numpy as np

def bootstrap(X, y, n_train=5000):
    """
    Creates training and testing sets with 5000 training examples
    sampled with replacement

    parameters
    ----------
    X: feature list
    y: targets
    n_train: number

    returns:
    --------
    X_train, X_test, y_train, y_test numpy arrays
    """

    n_samples = X.shape[0]
    train_inds = np.random.randint(n_samples, size=n_train)

    X_train = np.take(X, train_inds, axis=0)
    y_train = np.take(y, train_inds, axis=0)

    X_test = np.take(X, [i for i in range(n_samples) if i not in train_inds], axis=0)
    y_test = np.take(y, [i for i in range(n_samples) if i not in train_inds], axis=0)

    return X_train, X_test, y_train, y_test
```

```python
# imports
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score,
precision_score, recall_score
from bootstrap import bootstrap

# runs 5 trials of logistic regression on given dataset
def run_logistic_regression(X, y, n_trials=5):
    """
    Runs logistic regression for a dataset
    5 trials of LogReg with 5-fold cross validation and a gridsearch over
hyperparameter: C
    and computes mean accuracy over metrics
        accuracy, f1, roc, precision, recall

    parameters
    ----------
    X: feature vector
    y: target vector
    n_trials: number of trials to run

    returns
    --------
    train_metrics: average of each metric on training set across 5 trials
    test_metrics: average of each metric on test set across 5 trials
    hyperp: dataframe of hyperparameters tried during hyperparameter search,
            of metrics (acc, f1, roc) and their mean performance during
cross-validation
    """

    # hyperparameters
    C_list = [1e-8,1e-7,1e-6,1e-5,1e-4,1e-3,1e-2,1e-1,1e0,1e1,1e2,1e3,1e4]
    params = {'C': C_list}

    # metric evaluation scores
    scores = ['accuracy', 'f1', 'roc_auc', 'precision', 'recall']

    # to hold calculated metric performances
    train_metrics = []
    test_metrics = []

    for trial in range(n_trials):

        # initialize model for cross validation grid search
        LogReg = LogisticRegression(max_iter=200)
        GS = GridSearchCV(LogReg, params, scoring=scores, refit=False)
        # bootstrap training and testing sets
        X_train, X_test, y_train, y_test = bootstrap(X,y)
        GS.fit(X_train, y_train)

        # collect results and get dataframe to visualize hyperparameter
search
        res = GS.cv_results_
        hyperp = pd.DataFrame(res['params'])
        hyperp['acc'] = res['mean_test_accuracy']
        hyperp['f1'] = res['mean_test_f1']
        hyperp['roc'] = res['mean_test_roc_auc']
        hyperp.set_index('C', inplace=True)

        test_per = [] # test set performances
        train_per = [] # train set performances

        # get best hyperparameters for each metric and use on test set
        for s in scores:

            # train logreg with best hyperparameters for metric
            best_C = res['params'][np.argmax(res['mean_test_{}'.format(s)])]
            LR = LogisticRegression(C=best_C['C'])
            LR.fit(X_train, y_train)

            # predictions for train and test sets
            y_pred = LR.predict(X_test)
            y_pred_train = LR.predict(X_train)

            # evaluate metric on test set
            if s == 'accuracy':
                test_per.append(accuracy_score(y_test, y_pred))
                train_per.append(accuracy_score(y_train, y_pred_train))
            elif s == 'f1':
                test_per.append(f1_score(y_test, y_pred))
                train_per.append(f1_score(y_train, y_pred_train))
            elif s == 'roc_auc':
                test_per.append(roc_auc_score(y_test, y_pred))
                train_per.append(roc_auc_score(y_train, y_pred_train))
            elif s == 'precision':
                test_per.append(precision_score(y_test, y_pred))
                train_per.append(precision_score(y_train, y_pred_train))
            elif s == 'recall':
                test_per.append(recall_score(y_test, y_pred))
                train_per.append(recall_score(y_train, y_pred_train))

        train_metrics.append(train_per)
        test_metrics.append(test_per)

        print('Trial {} done'.format(trial+1))
```

```python

    # take mean of each metric across 5 trials
    train_metrics = np.mean(np.array(train_metrics), axis=0)
    test_metrics = np.mean(np.array(test_metrics), axis=0)

    return train_metrics, test_metrics, hyperp
```

```python
# imports
import pandas as pd
import numpy as np
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score,
precision_score, recall_score
from bootstrap import bootstrap

def run_svm(X, y, n_trials=5):
    """
    Runs svm for a dataset
    5 trials of SVM with 5-fold cross validation and a gridsearch over
hyperparameter: C
    and computes mean accuracy over metrics
        accuracy, f1, roc, precision, recall
    No cross-val hyperparameters will be returned because of the high
dimensionality of
    hyperparameters tried here

    parameters
    ----------
    X: feature vector
    y: target vector
    n_trials: number of trials to run

    returns
    --------
    train_metrics: average of each metric on training set across 5 trials
    test_metrics: average of each metric on test set across 5 trials

    """

    # hyperparameters
    C_list = [1e-7,1e-6,1e-5,1e-4,1e-3,1e-2,1e-1,1e0,1e1,1e2,1e3,1e4]
    kernel_list = ('rbf', 'linear', 'poly')
    degree_list = [2,3]
    gamma_list = [1e-6,1e-5,1e-4,1e-3,1e-2,1e-1]
    params = {'C': C_list, 'gamma':gamma_list, 'kernel':kernel_list,
'degree':degree_list}

    # metric evaluation scores
    scores = ['accuracy', 'f1', 'roc_auc', 'precision', 'recall']

    # to hold calculated metric performances
    train_metrics = []
    test_metrics = []

    for trial in range(n_trials):

        # initialize model for cross validation grid search
        SVM = SVC(max_iter=2000)
        GS = GridSearchCV(SVM, params, scoring=scores, refit=False)

        # bootstrap training and testing sets
        X_train, X_test, y_train, y_test = bootstrap(X,y)
        GS.fit(X_train, y_train)

        # collect results
        res = GS.cv_results_

        test_per = [] # test set performances
        train_per = [] # train set performances

        # get best hyperparameters for each metric and use on test set
        for s in scores:

            # train rf with best hyperparameters for metric
            best_p = res['params'][np.argmax(res['mean_test_{}'.format(s)])]
            SVM = SVC(max_iter=2000, kernel=best_p['kernel'], C=best_p['C'],
degree=best_p['degree'], gamma=best_p['gamma'])
            SVM.fit(X_train, y_train)

            # predictions for train and test sets
            y_pred = SVM.predict(X_test)
            y_pred_train = SVM.predict(X_train)

            # evaluate metric on test set
            if s == 'accuracy':
                test_per.append(accuracy_score(y_test, y_pred))
                train_per.append(accuracy_score(y_train, y_pred_train))
            elif s == 'f1':
                test_per.append(f1_score(y_test, y_pred))
                train_per.append(f1_score(y_train, y_pred_train))
            elif s == 'roc_auc':
                test_per.append(roc_auc_score(y_test, y_pred))
                train_per.append(roc_auc_score(y_train, y_pred_train))
            elif s == 'precision':
                test_per.append(precision_score(y_test, y_pred))
                train_per.append(precision_score(y_train, y_pred_train))
            elif s == 'recall':
                test_per.append(recall_score(y_test, y_pred))
                train_per.append(recall_score(y_train, y_pred_train))

        train_metrics.append(train_per)
        test_metrics.append(test_per)

        print('Trial {} done'.format(trial+1))

```

```
    # take mean of each metric across 5 trials
    train_metrics = np.mean(np.array(train_metrics), axis=0)
    test_metrics = np.mean(np.array(test_metrics), axis=0)

    return train_metrics, test_metrics
```

```python
# imports
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score, precision_score, recall_score
from bootstrap import bootstrap

def run_random_forests(X, y, n_trials=5):
    """
    Runs random forests for a dataset
    5 trials of random forests with 5-fold cross validation and a gridsearch over hyperparameters:
        max_depth, max_features
    and computes mean accuracy over metrics
        accuracy, f1, roc, precision, recall

    parameters
    ----------
    X: feature vector
    y: target vector
    n_trials: number of trials to run

    returns
    --------
    train_metrics: average of each metric on training set across 5 trials
    test_metrics: average of each metric on test set across 5 trials
    hyperp: dataframe of hyperparameters tried during hyperparameter search,
            of metrics (acc, f1, roc) and their mean performance during cross-validation
    """

    # hyperparameters
    depth_list = [2,4,6,8,16,20,30,None]
    feature_list = [1,2,4,6,8,12,16,20]
    params = {'max_depth':depth_list, 'max_features':feature_list}

    # metric evaluation scores
    scores = ['accuracy', 'f1', 'roc_auc', 'precision', 'recall']

    # to hold calculated metric performances
    train_metrics = []
    test_metrics = []

    for trial in range(n_trials):

        # initialize model for cross validation grid search
        RF = RandomForestClassifier()
        GS = GridSearchCV(RF, params, scoring=scores, refit=False)
        GS = GridSearchCV(RF, params, scoring=scores, refit=False)

        # bootstrap training and testing sets
        X_train, X_test, y_train, y_test = bootstrap(X,y)
        GS.fit(X_train, y_train)

        # collect results and get dataframe to visualize hyperparameter search
        res = GS.cv_results_
        hyperp = pd.DataFrame(res['params'])
        hyperp['acc'] = res['mean_test_accuracy']
        hyperp['f1'] = res['mean_test_f1']
        hyperp['roc'] = res['mean_test_roc_auc']

        test_per = [] # test set performances
        train_per = [] # train set performances

        # get best hyperparameters for each metric and use on test set
        for s in scores:

            # train rf with best hyperparameters for metric
            best_p = res['params'][np.argmax(res['mean_test_{}'.format(s)])]
            RF = RandomForestClassifier(max_depth=best_p['max_depth'], max_features=best_p['max_features'])
            RF.fit(X_train, y_train)

            # predictions for train and test sets
            y_pred = RF.predict(X_test)
            y_pred_train = RF.predict(X_train)

            # evaluate metric on test set
            if s == 'accuracy':
                test_per.append(accuracy_score(y_test, y_pred))
                train_per.append(accuracy_score(y_train, y_pred_train))
            elif s == 'f1':
                test_per.append(f1_score(y_test, y_pred))
                train_per.append(f1_score(y_train, y_pred_train))
            elif s == 'roc_auc':
                test_per.append(roc_auc_score(y_test, y_pred))
                train_per.append(roc_auc_score(y_train, y_pred_train))
            elif s == 'precision':
                test_per.append(precision_score(y_test, y_pred))
                train_per.append(precision_score(y_train, y_pred_train))
            elif s == 'recall':
                test_per.append(recall_score(y_test, y_pred))
                train_per.append(recall_score(y_train, y_pred_train))

        train_metrics.append(train_per)
        test_metrics.append(test_per)

        print('Trial {} done'.format(trial+1))

    # take mean of each metric across 5 trials
    train_metrics = np.mean(np.array(train_metrics), axis=0)
    test_metrics = np.mean(np.array(test_metrics), axis=0)

    return train_metrics, test_metrics, hyperp
```

```python
import pandas as pd
import numpy as np
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score, precision_score, recall_score
from bootstrap import bootstrap

def run_ann(X, y, n_trials=5):
    """
    Runs artifical neural network for a dataset
    5 trials of ANN with 5-fold cross validation and a gridsearch over hyperparameters:
        hidden_layer_sizes, momentum
    and computes mean accuracy over metrics
        accuracy, f1, roc, precision, recall

    parameters
    ----------
    X: feature vector
    y: target vector
    n_trials: number of trials to run

    returns
    --------
    train_metrics: average of each metric on training set across 5 trials
    test_metrics: average of each metric on test set across 5 trials

    """

    # hyperparameters
    layers = [[50],[50,50],[50,50,50,50,50],[50,50,50,50,50,50,50,50],[100],[100,100],[100,100,100,100,100],
          [100,100,100,100,100,100,100,100],[200],[200,200],[200,200,200,200,200],
          [200,200,200,200,200,200,200,200]]
    momentums = [0, 0.2, 0.5, 0.9]
    params = {'hidden_layer_sizes':layers, 'momentum':momentums}

    # metric evaluation scores
    scores = ['accuracy', 'f1', 'roc_auc', 'precision', 'recall']

    # to hold calculated metric performances
    train_metrics = []
    test_metrics = []

    for trial in range(n_trials):

        # initialize model for cross validation grid search
        ANN = MLPClassifier(solver='sgd', learning_rate='adaptive')
        ANN = MLPClassifier(solver='sgd', learning_rate='adaptive')
        GS = GridSearchCV(ANN, params, scoring=scores, refit=False)

        # bootstrap training and testing sets
        X_train, X_test, y_train, y_test = bootstrap(X,y)
        GS.fit(X_train, y_train)

        # collect results
        res = GS.cv_results_

        # collect/store hyperparameters for visualization
        hyperp = pd.DataFrame(res['params'])
        hyperp['acc'] = res['mean_test_accuracy']
        hyperp['f1'] = res['mean_test_f1']
        hyperp['roc'] = res['mean_test_roc_auc']
        hidden_layers = []
        # rename hidden layer sizes column from list to str
        for i,row in hyperp.iterrows():
            if 50 in row['hidden_layer_sizes']:

 hidden_layers.append('{}x50'.format(len(row['hidden_layer_sizes'])))
            elif 100 in row['hidden_layer_sizes']:

 hidden_layers.append('{}x100'.format(len(row['hidden_layer_sizes'])))
            elif 200 in row['hidden_layer_sizes']:

 hidden_layers.append('{}x200'.format(len(row['hidden_layer_sizes'])))
        hyperp['hidden_layer_sizes'] = hidden_layers

        test_per = [] # test set performances
        train_per = [] # train set performances

        # get best hyperparameters for each metric and use on test set
        for s in scores:

            # train rf with best hyperparameters for metric
            best_p = res['params'][np.argmax(res['mean_test_{}'.format(s)])]
            ANN = MLPClassifier(solver='sgd', learning_rate='adaptive',

 hidden_layer_sizes=best_p['hidden_layer_sizes'],
momentum=best_p['momentum'])
            ANN.fit(X_train, y_train)

            # predictions for train and test sets
            y_pred = ANN.predict(X_test)
            y_pred_train = ANN.predict(X_train)

            # evaluate metric on test set
            if s == 'accuracy':
                test_per.append(accuracy_score(y_test, y_pred))
```

```python
                train_per.append(accuracy_score(y_train, y_pred_train))
            elif s == 'f1':
                test_per.append(f1_score(y_test, y_pred))
                train_per.append(f1_score(y_train, y_pred_train))
            elif s == 'roc_auc':
                test_per.append(roc_auc_score(y_test, y_pred))
                train_per.append(roc_auc_score(y_train, y_pred_train))
            elif s == 'precision':
                test_per.append(precision_score(y_test, y_pred))
                train_per.append(precision_score(y_train, y_pred_train))
            elif s == 'recall':
                test_per.append(recall_score(y_test, y_pred))
                train_per.append(recall_score(y_train, y_pred_train))

        train_metrics.append(train_per)
        test_metrics.append(test_per)

        print('Trial {} done'.format(trial+1))

    # take mean of each metric across 5 trials
    train_metrics = np.mean(np.array(train_metrics), axis=0)
    test_metrics = np.mean(np.array(test_metrics), axis=0)

    return train_metrics, test_metrics, hyperp
```

algorithms

Running all algorithms on all datasets

March 14, 2021

1 Comparison of various algorithms on metrics

This notebook for running algorithms: * Logistic Regression * SVM * Random Forests * Artificial neural network

and scores them based on metrics: * Accuracy * F1 Score * ROC AUC * Precision * Recall

Runs each algorithm for each dataset across 5 trials, where GridSearch is used to find the optimal hyperparameters for each metric, then runs the classifier on training/testing sets and takes the mean over 5 trials.

Results are then stored in the results folder of this directory

[6]:
```
# import needed packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score,␣
 ↪precision_score, recall_score
from sklearn.preprocessing import StandardScaler

# import needed functions
from preprocess import prep_airlines, prep_income, prep_phishing, prep_surgical
from bootstrap import bootstrap
from logistic_regression import run_logistic_regression
from random_forest import run_random_forests
from support_vector import run_svm
from artificial_nn import run_ann

import warnings
warnings.filterwarnings('ignore')
```

1.1 Define datasets and metrics to be used

[2]:
```
# datasets and metrics
datasets = ['airline', 'income', 'phishing', 'surgical']
metrics = ['accuracy', 'f1', 'roc_auc', 'precision', 'recall']
```

2 Logistic Regression

[3]:
```
# final values
logreg_results_train = np.zeros((len(datasets), len(metrics)))
logreg_results_test = np.zeros((len(datasets), len(metrics)))
logreg_hyperparams = [] # list of dataframes

# for each dataset: run trials and add to final results

# AIRLINE
print('AIRLINE\n-----------------------------------')
X,y = prep_airlines()
train, test, hypers = run_logistic_regression(X,y)

logreg_results_train[0,:] = train
logreg_results_test[0,:] = test
logreg_hyperparams.append(hypers)

# INCOMES
print('\nINCOMES\n-----------------------------------')
X,y = prep_income()
train, test, hypers = run_logistic_regression(X,y)

logreg_results_train[1,:] = train
logreg_results_test[1,:] = test
logreg_hyperparams.append(hypers)

# PHISHING
print('\nPHISHING\n-----------------------------------')
X,y = prep_phishing()
train, test, hypers = run_logistic_regression(X,y)

logreg_results_train[2,:] = train
logreg_results_test[2,:] = test
logreg_hyperparams.append(hypers)

# SURGICAL
print('\nSURGICAL\n-----------------------------------')
X,y = prep_surgical()
train, test, hypers = run_logistic_regression(X,y)
```

```
logreg_results_train[3,:] = train
logreg_results_test[3,:] = test
logreg_hyperparams.append(hypers)
```

```
AIRLINE
----------------------------------
Trial 1 done
Trial 2 done
Trial 3 done
Trial 4 done
Trial 5 done

INCOMES
----------------------------------
Trial 1 done
Trial 2 done
Trial 3 done
Trial 4 done
Trial 5 done

PHISHING
----------------------------------
Trial 1 done
Trial 2 done
Trial 3 done
Trial 4 done
Trial 5 done

SURGICAL
----------------------------------
Trial 1 done
Trial 2 done
Trial 3 done
Trial 4 done
Trial 5 done
```

[23]:
```
# save to results folder
result_dir = './results/'

df_results_train = pd.DataFrame(logreg_results_train, columns=metrics,␣
 ↪index=datasets)
df_results_train['mean'] = np.mean(df_results_train, axis=1)
df_results_train.to_csv(result_dir+'logreg_results_train.csv')

df_results_test = pd.DataFrame(logreg_results_test, columns=metrics,␣
 ↪index=datasets)
```

```
df_results_test['mean'] = np.mean(df_results_test, axis=1)
df_results_test.to_csv(result_dir+'logreg_results_test.csv')

df_hyperparams = pd.concat(logreg_hyperparams)
df_hyperparams.to_csv(result_dir+'logreg_hyperparameters.csv')
```

[ ]:
```
# to visualize hyperparameter search results
for i,hyp in enumerate(logreg_hyperparams):
    sns.heatmap(hyp, annot=True, cmap='viridis')
    plt.title(datasets[i])
    plt.show()
```

3 SVM

[3]:
```
# final values
svm_results_train = np.zeros((len(datasets), len(metrics)))
svm_results_test = np.zeros((len(datasets), len(metrics)))

# for each dataset: run trials and add to final results

# AIRLINE
print('AIRLINE\n----------------------------------')
X,y = prep_airlines()
train, test = run_svm(X,y)

svm_results_train[0,:] = train
svm_results_test[0,:] = test

# INCOMES
print('\nINCOMES\n----------------------------------')
X,y = prep_income()
train, test = run_svm(X,y)

svm_results_train[1,:] = train
svm_results_test[1,:] = test

# PHISHING
print('\nPHISHING\n----------------------------------')
X,y = prep_phishing()
train, test = run_svm(X,y)

svm_results_train[2,:] = train
svm_results_test[2,:] = test

# SURGICAL
print('\nSURGICAL\n----------------------------------')
```

```
X,y = prep_surgical()
train, test = run_svm(X,y)

svm_results_train[3,:] = train
svm_results_test[3,:] = test
```

```
AIRLINE
-----------------------------------
Trial 1 done
Trial 2 done
Trial 3 done
Trial 4 done
Trial 5 done

INCOMES
-----------------------------------
Trial 1 done
Trial 2 done
Trial 3 done
Trial 4 done
Trial 5 done

PHISHING
-----------------------------------
Trial 1 done
Trial 2 done
Trial 3 done
Trial 4 done
Trial 5 done

SURGICAL
-----------------------------------
Trial 1 done
Trial 2 done
Trial 3 done
Trial 4 done
Trial 5 done
```

[4]:
```
# save to results folder
result_dir = './results/'

df_results_train = pd.DataFrame(svm_results_train, columns=metrics,␣
 ↪index=datasets)
df_results_train['mean'] = np.mean(df_results_train, axis=1)
df_results_train.to_csv(result_dir+'svm_results_train.csv')
```

```
df_results_test = pd.DataFrame(svm_results_test, columns=metrics,␣
 ↪index=datasets)
df_results_test['mean'] = np.mean(df_results_test, axis=1)
df_results_test.to_csv(result_dir+'svm_results_test.csv')
```

4 Random Forests

[3]:
```
# final values
rf_results_train = np.zeros((len(datasets), len(metrics)))
rf_results_test = np.zeros((len(datasets), len(metrics)))
rf_hyperparams = [] # list of dataframes

# for each dataset: run trials and add to final results

# AIRLINE
print('AIRLINE\n----------------------------------')
X,y = prep_airlines()
train, test, hypers = run_random_forests(X,y)

rf_results_train[0,:] = train
rf_results_test[0,:] = test
rf_hyperparams.append(hypers)

# INCOMES
print('\nINCOMES\n----------------------------------')
X,y = prep_income()
train, test, hypers = run_random_forests(X,y)

rf_results_train[1,:] = train
rf_results_test[1,:] = test
rf_hyperparams.append(hypers)

# PHISHING
print('\nPHISHING\n----------------------------------')
X,y = prep_phishing()
train, test, hypers = run_random_forests(X,y)

rf_results_train[2,:] = train
rf_results_test[2,:] = test
rf_hyperparams.append(hypers)

# SURGICAL
print('\nSURGICAL\n----------------------------------')
X,y = prep_surgical()
train, test, hypers = run_random_forests(X,y)
```

```
rf_results_train[3,:] = train
rf_results_test[3,:] = test
rf_hyperparams.append(hypers)
```

```
AIRLINE
----------------------------------
Trial 1 done
Trial 2 done
Trial 3 done
Trial 4 done
Trial 5 done

INCOMES
----------------------------------
Trial 1 done
Trial 2 done
Trial 3 done
Trial 4 done
Trial 5 done

PHISHING
----------------------------------
Trial 1 done
Trial 2 done
Trial 3 done
Trial 4 done
Trial 5 done

SURGICAL
----------------------------------
Trial 1 done
Trial 2 done
Trial 3 done
Trial 4 done
Trial 5 done
```

[4]:
```
# save to results folder
result_dir = './results/'

df_results_train = pd.DataFrame(rf_results_train, columns=metrics,␣
↪index=datasets)
df_results_train['mean'] = np.mean(df_results_train, axis=1)
df_results_train.to_csv(result_dir+'rf_results_train.csv')

df_results_test = pd.DataFrame(rf_results_test, columns=metrics, index=datasets)
df_results_test['mean'] = np.mean(df_results_test, axis=1)
df_results_test.to_csv(result_dir+'rf_results_test.csv')
```

```
df_hyperparams = pd.concat(rf_hyperparams)
df_hyperparams.to_csv(result_dir+'rf_hyperparameters.csv')
```

5 Artifical Neural Networks

[7]:
```
# final values
ann_results_train = np.zeros((len(datasets), len(metrics)))
ann_results_test = np.zeros((len(datasets), len(metrics)))
ann_hyperparams = [] # list of dataframes

# for each dataset: run trials and add to final results

# AIRLINE
print('AIRLINE\n----------------------------------')
X,y = prep_airlines()
train, test, hypers = run_ann(X,y)

ann_results_train[0,:] = train
ann_results_test[0,:] = test
ann_hyperparams.append(hypers)

# INCOMES
print('\nINCOMES\n----------------------------------')
X,y = prep_income()
# doesn't work with non-scaled values of second feature
scaler = StandardScaler()
X[:,1] = scaler.fit_transform(X[:,1].reshape(-1,1)).reshape(-1)
train, test, hypers = run_ann(X,y)

ann_results_train[1,:] = train
ann_results_test[1,:] = test
ann_hyperparams.append(hypers)

# PHISHING
print('\nPHISHING\n----------------------------------')
X,y = prep_phishing()
train, test, hypers = run_ann(X,y)

ann_results_train[2,:] = train
ann_results_test[2,:] = test
ann_hyperparams.append(hypers)

# SURGICAL
print('\nSURGICAL\n----------------------------------')
X,y = prep_surgical()
```

```
train, test, hypers = run_ann(X,y)

ann_results_train[3,:] = train
ann_results_test[3,:] = test
ann_hyperparams.append(hypers)
```

```
AIRLINE
----------------------------------
Trial 1 done
Trial 2 done
Trial 3 done
Trial 4 done
Trial 5 done

INCOMES
----------------------------------
Trial 1 done
Trial 2 done
Trial 3 done
Trial 4 done
Trial 5 done

PHISHING
----------------------------------
Trial 1 done
Trial 2 done
Trial 3 done
Trial 4 done
Trial 5 done

SURGICAL
----------------------------------
Trial 1 done
Trial 2 done
Trial 3 done
Trial 4 done
Trial 5 done
```

[8]:
```
# save to results folder
result_dir = './results/'

df_results_train = pd.DataFrame(ann_results_train, columns=metrics,␣
 ↪index=datasets)
df_results_train['mean'] = np.mean(df_results_train, axis=1)
df_results_train.to_csv(result_dir+'ann_results_train.csv')
```

```
df_results_test = pd.DataFrame(ann_results_test, columns=metrics,␣
 ↪index=datasets)
df_results_test['mean'] = np.mean(df_results_test, axis=1)
df_results_test.to_csv(result_dir+'ann_results_test.csv')

df_hyperparams = pd.concat(ann_hyperparams)
df_hyperparams.to_csv(result_dir+'ann_hyperparameters.csv')
```

[ ]:

```

```