CS 519 Applied Machine Learning I
Open machine learning project

Team members:

Troy McMillan, William Baker, Aaron Hudson

Topic: Outcome Prediction for Shelter Animals

Source:

The following website presents this data set as a challenge:

https://www.kaggle.com/c/shelter-animal-outcomes/overview/description

Problem:

Succinctly stated, Animal rights advocates from the Austin Animal Center and the ASPCA have put together data for shelter animals. It is the desire of these organizations, and our goal, to use this data to try and accurately predict the likely outcomes for animals when they leave the shelter.

Solution:

Machine Learning can be implemented to determine if a shelter animal has a high chance of being adopted or not. This will require a high amount of data preprocessing, as the initial data is highly categorical. Fortunately, it's also distinct without too many levels, so a labeler can be used to get numerical values for use. The key traits that will be used for training will be AnimalType, AgeuponOutcome, SexuponOutcome, and OutcomeType. OutcomeType will be the training target.

Since this outcome relies on so many traits, a simple classifier will likely not give good results. Thus, the group will use ensemble methods for predictions, though the inclusion of a more basic learner such as a support vector machine may be worth trying simply for comparison.

Solution: Walkthrough

Preparing the data:

Current data can be found in: https://github.com/ghost8472/CS519_2020S_PS/tree/master/scratch/data

The source data consists of 10 features, 6 of which are labeled classes (including our target feature of OutcomeType), two of which are identification features, and two time-related features. The data centers around the Outcome faced by the animal at the end of their stay at the shelter. The features are:

A. AnimalID – a unique identifier for each animal
B. Name – an identifier, with no guarantee of uniqueness
C. DateTime – the date/time of when the Outcome occurred
D. OutcomeType – the action that occurred for the Outcome

E. OutcomeSubType – extra information about the Outcome action
F. AnimalType – label class of either Dog or Cat
G. SexuponOutcome – label class identifying the gender/sterilization of the animal
H. AgeuponOutcome – the age (in days, weeks, months, or years) of the animal
I. Breed – label class of the breed(s) of the animal, possibly including "Mix"
J. Color – label class of the primary/secondary color(s) of the animal

Each type of feature (identification, time, and label classes) require some special attention, in order to make them useful for feeding to the machine. Used heavily are binary features, which are features where it is either existent (1) or non-existent (0), which are meant to be easy for classifiers to partition. For example, Name is converted to a binary feature by: if they have a Name it is a 1, and if they didn't it is a 0. This is important for splitting out the label classes of Breed and Color, as each may represent two classes, and important information becomes lost if each combination is merely treated with the standard labeling technique of assigning a single number to each combination.

A. AnimalID – this unique identifier was removed, as it had no discernable numeric significance.
B. Name – this identifier was converted into a binary feature, of whether they have a name or not.
C. DateTime – in addition to converting the date/time into a timestamp, it also spawns into new binary features, one for each month, one for each day of the week, and ones for morning, midday, and night, for a total of 22 new features. The times of day features were based on a manual pre-analysis that their business hours are from 7am to 7pm, thereby making morning be from 7am to noon, afternoon from noon to 7pm, and night from 7pm to 7am.
D. OutcomeType – this was moved to the end of the feature list, and left in its original form of a label class. This is the target we are using, so the standard label handling is acceptable.
E. OutcomeSubType – this feature was removed, as it is part of the outcome, and not included in the "test" data from the source's challenge.
F. AnimalType – this feature was turned into two binary features: one for Cat, and one for Dog. This handling keeps it consistent with the handling of the other label class features. Standard labeling technique is also maintained.
G. SexuponOutcome – this feature was turned into four binary features: fertile/sterilized, males, females, and unknown. Standard labeling technique is also maintained.
H. AgeuponOutcome – this feature is initially written with different time units (days, weeks, months, years), so this was standardized into a numeric value of years, for example "3 months" becomes 0.25
I. Breed – this label class contains many instances of an animal belonging to two breeds, or a breed with a "Mix" afterward. To represent this as binary features, each breed becomes a feature, and a "Mix" feature was added as well. When an animal belongs to two breeds, the features for both breeds will be 1, and "Mix" will be 1. When an animal belongs to one breed, but is a "Mix," then the feature for that breed will be 1, and "Mix" will be 1. If of an animal is of a single breed, only the one breed will be set to 1. Standard labeling technique is also maintained.
J. Color – this label class contains many instances of an animal possessing two colors. Each color became a binary feature, as well as a "Mix" feature. If an animal has two colors, then the features for both colors will be 1, as will the "Mix" feature. Standard labeling technique is also retained.

Initial comparison of classifiers

Initial comparisons of the classifiers were conducted using train_test_split on the converted data. The data, without the extra binary features described above, was also tested, to see the impact of adding the binary features.

	Input file: traditional label encoding train_conversion_lowfeat_sansheader.csv				Input file: with added features train_conversion_extfeat_sansheader.csv			
	Training		Testing		Training		Testing	
	Time	F1	Time	F1	Time	F1	Time	F1
Perceptron	5.261	.320	.426	.324	70.166	.363	.575	.354
SVM	12.759	.365	9.390	.365	215.047	.307	191.495	.308
Decision Tree	.039	.407	.440	.414	.385	.407	.546	.414
KNN	.074	.528	1.900	.400	3.062	.456	219.731	.323
Random Forrest	.162	.295	.530	.297	.405	.269	.570	.271
Ada Boost	1.232	.388	.691	.383	10.240	.387	1.621	.381
Bagging	1.633	.407	.523	.414	5.289	.407	2.929	.414

Observations:

- The best performance was the K Nearest Neighbors classifier
- The extra features did help some classifiers, such as the Perceptron, but not the KNN classifier
- Decision Tree and Bagging were about equal, but neither was as good as KNN

Seeing that KNN is the best classifier, and the dataset utilizing standard labeling gets better results, next is comparing this combination with different parameters for the KNN classifier.

	Input file: traditional label encoding train_conversion_lowfeat_sansheader.csv											
	Testing											
	F1											
neighbors	1	2	3	4	5	6	7	8	9	10	20	30
KNN	.376	.372	.385	.400	.400	.402	.402	.408	.395	.400	.401	.400

Observations:

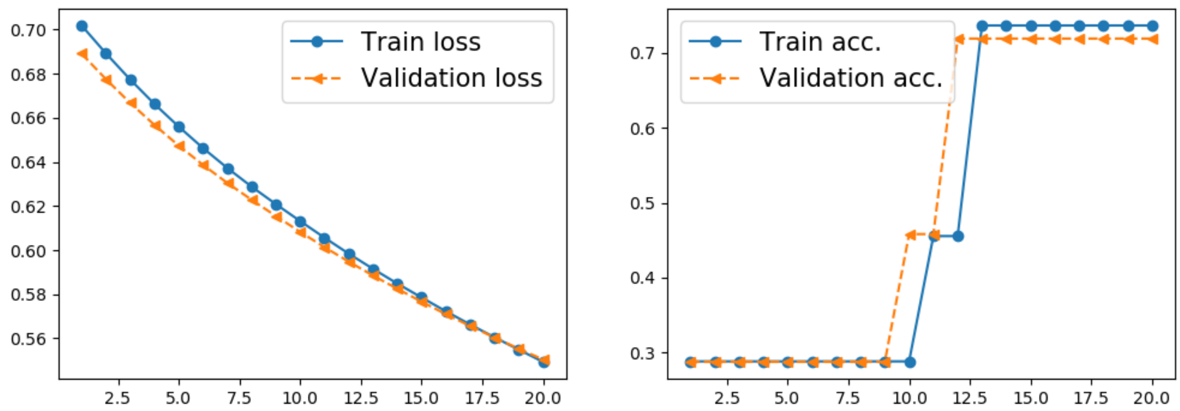
- The highest F1 for testing data occurred when neighbors was set to 8.
- Increasing the number of neighbors does not keep increasing the F1.
- The accuracy for the test data when neighbors is 8: 0.623

Classification using Neural Network

Employing a Neural Network resulted in much greater accuracy.

	Input file: with added features train_conversion_extfeat_sansheader.csv		
	Training Time (s)	Training Accuracy	Validation Accuracy
Neural Network	8.59	.734	.731

The initial epochs had accuracy similar to the simpler classifiers, with a sudden jump to the final accuracy.



Appendix A: Command sequence “readme.txt”

The following commands were used to generate the data in the report:

```
python main.py perceptron data\train_conversion_lowfeat_sansheader.csv
python main.py svm data\train_conversion_lowfeat_sansheader.csv
python main.py dtree data\train_conversion_lowfeat_sansheader.csv
python main.py knn data\train_conversion_lowfeat_sansheader.csv
python main.py randforest data\train_conversion_lowfeat_sansheader.csv
python main.py adaboost data\train_conversion_lowfeat_sansheader.csv
python main.py bagging data\train_conversion_lowfeat_sansheader.csv
python main.py perceptron data\train_conversion_extfeat_sansheader.csv
python main.py svm data\train_conversion_extfeat_sansheader.csv
python main.py dtree data\train_conversion_extfeat_sansheader.csv
python main.py knn data\train_conversion_extfeat_sansheader.csv
python main.py randforest data\train_conversion_extfeat_sansheader.csv
python main.py adaboost data\train_conversion_extfeat_sansheader.csv
python main.py bagging data\train_conversion_extfeat_sansheader.csv
```

```
python main.py knn data\train_conversion_lowfeat_sansheader.csv –neighbors=1
python main.py knn data\train_conversion_lowfeat_sansheader.csv –neighbors=2
python main.py knn data\train_conversion_lowfeat_sansheader.csv –neighbors=3
python main.py knn data\train_conversion_lowfeat_sansheader.csv –neighbors=4
python main.py knn data\train_conversion_lowfeat_sansheader.csv –neighbors=5
python main.py knn data\train_conversion_lowfeat_sansheader.csv –neighbors=6
python main.py knn data\train_conversion_lowfeat_sansheader.csv –neighbors=7
python main.py knn data\train_conversion_lowfeat_sansheader.csv –neighbors=8
python main.py knn data\train_conversion_lowfeat_sansheader.csv –neighbors=9
python main.py knn data\train_conversion_lowfeat_sansheader.csv –neighbors=10
python main.py knn data\train_conversion_lowfeat_sansheader.csv –neighbors=20
python main.py knn data\train_conversion_lowfeat_sansheader.csv –neighbors=30
```

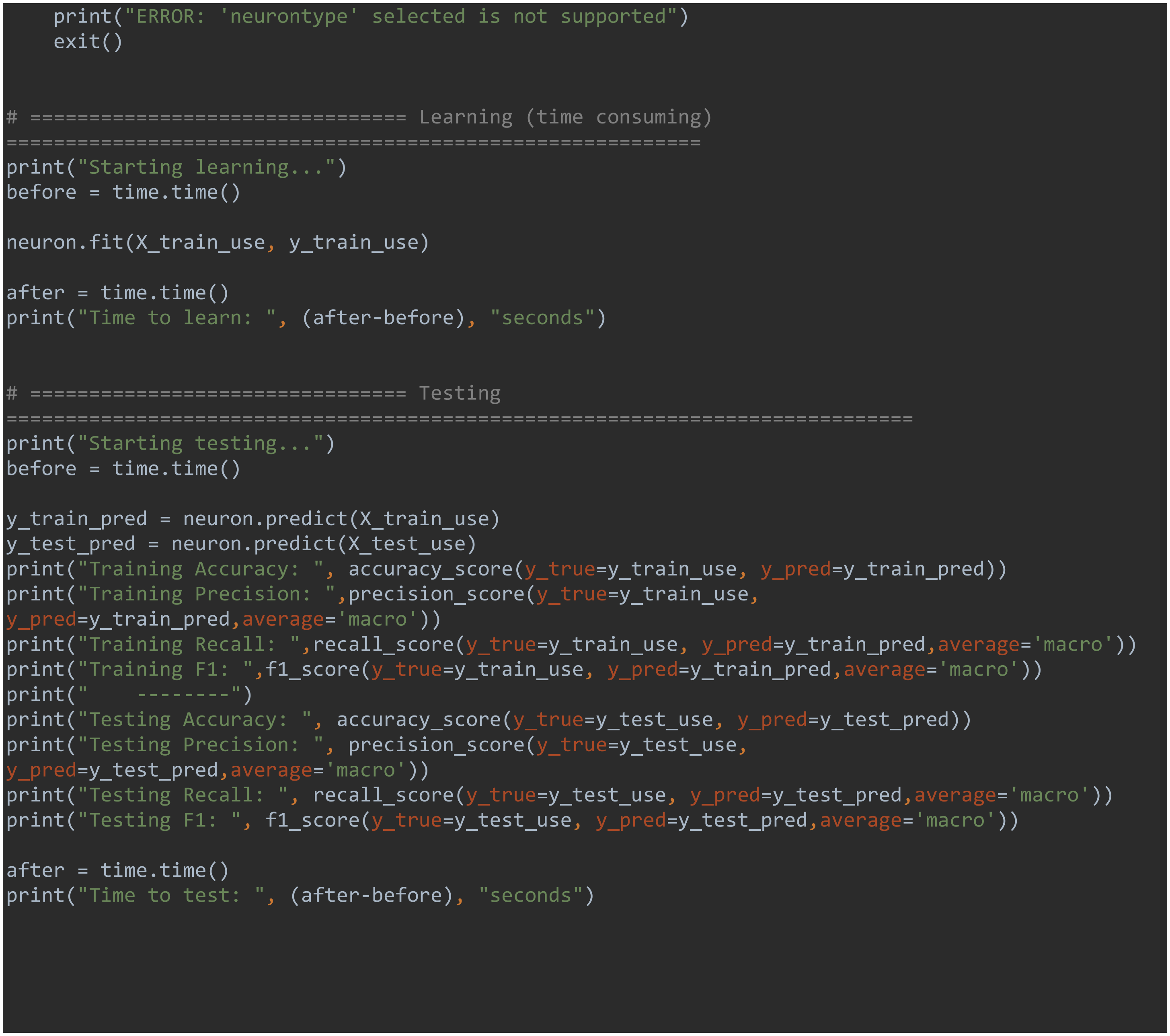
```
python nn.py -f data\train_conversion_lowfeat_sansheader.csv -e 20
```

Appendix B: Machine learning script – Basic Classifiers

This script was used for the data processing discussed in this report. This script was run with the following command sequence:

```
#!/bin/python3
#written using Python 3.8.1, by William Baker, February-March 2020, for NMSU's  CS-519 course

import argparse
import pandas as pd
import numpy as np
import time
from sklearn import datasets
from sklearn.linear_model import Perceptron
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, BaggingClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.decomposition import KernelPCA, PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

aparser = argparse.ArgumentParser()
aparser.add_argument('neurontype', type=str,
choices=['perceptron','svm','dtree','knn','randforest','adaboost','bagging'])
aparser.add_argument('datafile', type=str, help="file containing the data, in CSV format (last
column is class)")
aparser.add_argument('--builtin', action='store_true', help="use flag to treat the 'datafile' as a
scikit-learn data source instead of a file")
aparser.add_argument('--fetch', action='store_true', help="use flag to treat the 'datafile' as a
scikit-learn data fetch from OpenML instead of a file")
aparser.add_argument('--nosplit', action='store_true', help="do not split the data between train
and test")
aparser.add_argument('--maxrows', default=20000, help='if rows exceed this max, it will crop
them')
aparser.add_argument('--splitseed', type=int, default=1, help="the random seed used to
shuffle/split the data into train/test")
aparser.add_argument('--splitsize', type=float, default=0.3, help="the amount of data cases used
for test cases")
aparser.add_argument('--nostandard', action='store_true', help="do not standardize the data")
aparser.add_argument('--reduce', type=str, default="none", choices=["none","pca","lda","kpca"],
help="feature/dimension reduction algorithm to use")
aparser.add_argument('--redcomps', type=int, default=2, help="Feature reduction - number of
components/eigenvectors/features to keep")
aparser.add_argument('--kpca_kernel', type=str, default="rbf", choices=["linear", "poly", "rbf",
"sigmoid", "cosine"], help="Feature reduction - KPCA - kernel to use")
aparser.add_argument('--randseed', type=int, default=1, help="All neurons - set the random seed
used for initial state values")
aparser.add_argument('--eta', type=float, default=0.1, help="Perceptron & AdaBoost option - the
learning factor to pass to the Perceptron")
aparser.add_argument('--iter', type=int, default=1000, help="Perceptron option - how many epochs
of learning")
```

```
aparser.add_argument('--C', type=float, default=1.0, help="SVM option - the penalty for
miscalculation")
aparser.add_argument('--kernel', type=str, default='rbf',
choices=['linear','rbf','poly','sigmoid'], help="SVM option - whether to use linear, or RBF
kernel")
aparser.add_argument('--criterion', type=str, default='gini', choices=['gini','entropy'],
help="Decision Tree & Random Forest option - what impurity criterion to use")
aparser.add_argument('--maxdepth', type=int, default=4, help="Decision Tree & Random Forest option
- maximum depth")
aparser.add_argument('--neighbors', type=int, default=5, help="K Nearest Neighbors option - number
of neighbors to find ('k')")
aparser.add_argument('--knnmetric', type=str, default='minkowski', choices=['minkowski'], help="K
Nearest Neighbors option - distance calculation metric")
aparser.add_argument('--knnmetricp', type=int, default=2, choices=[1,2], help="K Nearest Neighbors
option - when using minkowski, 1=Manhattan distance, 2=Euclidean")
aparser.add_argument('--n_est', type=int, default=25, help="Random Forest, AdaBoost, & Bagging
option - number of estimators")
aparser.add_argument('--n_jobs', type=int, default=2, help="Random Forest option - number of jobs
to run in parallel")
aparser.add_argument('--min_samples_split', type=int, default=2, help="Random Forest option -
minimum samples req. for split")
aparser.add_argument('--min_samples_leaf', type=int, default=2, help="Random Forest option -
minimum samples leaf")
aparser.add_argument('--bootstrap', action='store_true', help="Random Forest & Bagging option -
whether to bootstrap")
aparser.add_argument('--max_features', type=float, default=1.0, help="Bagging option - maximum
features (1.0 means all)")
aparser.add_argument('--max_samples', type=float, default=1.0, help="Bagging option - maximum
samples (1.0 means all)")

args = aparser.parse_args()


# =========================== File reading
==========================================================================
print("Accessing data...")

X = None
y = None

data = pd.read_csv(args.datafile, header=None, encoding='utf-8')

X = data.values[:, 0:-1]
y = data.values[:, -1]

# Note: using "median" so that values are known valid.  consider an int enumerated type, a value
of 1.3 makes no sense.
imr = SimpleImputer(missing_values=np.nan, strategy='median')
imr.fit(X)
X = imr.transform(X)

print("Shape of data", X.shape)

# split the data into train and test data sets
if args.nosplit:
    X_train = X
    X_test = X
    y_train = y
    y_test = y
else:
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=args.splitsize,
random_state=args.splitseed, stratify=y)

# standardize the inputs
if args.nostandard:
    X_train_std = X_train
    X_test_std = X_test
else:
    sc = StandardScaler()
    sc.fit(X_train)
    X_train_std = sc.transform(X_train)
    X_test_std = sc.transform(X_test)


# use the latest X, y values from above transformations/standardizations/reductions
X_train_use = X_train_std
X_test_use = X_test_std
y_train_use = y_train
y_test_use = y_test

# ============================== Neuron initialization
================================================================
print("Initializing neuron...")

base_est = None
if args.neurontype == 'adaboost' or args.neurontype == 'bagging':
    base_est = DecisionTreeClassifier(criterion=args.criterion, max_depth=args.maxdepth,
random_state=args.randseed)

neuron = None
if args.neurontype == 'perceptron':
    neuron = Perceptron(eta0=args.eta, tol=None, max_iter=args.iter, random_state=args.randseed)
elif args.neurontype == 'svm':
    neuron = SVC(kernel=args.kernel, C=args.eta, random_state=args.randseed)
elif args.neurontype == 'dtree':
    neuron = DecisionTreeClassifier(criterion=args.criterion, max_depth=args.maxdepth,
random_state=args.randseed)
elif args.neurontype == 'knn':
    neuron = KNeighborsClassifier(n_neighbors=args.neighbors, metric=args.knnmetric,
p=args.knnmetricp)
elif args.neurontype == 'randforest':
    neuron = RandomForestClassifier(criterion=args.criterion, n_estimators=args.n_est,
n_jobs=args.n_jobs,
                                    min_samples_split=args.min_samples_split,
min_samples_leaf=args.min_samples_leaf,
                                    bootstrap=args.bootstrap, max_depth=args.maxdepth,
random_state=args.randseed)
elif args.neurontype == 'adaboost':
    neuron = AdaBoostClassifier(base_estimator=base_est, n_estimators=args.n_est,
learning_rate=args.eta, random_state=args.randseed)
elif args.neurontype == 'bagging':
    neuron = BaggingClassifier(base_estimator=base_est, n_estimators=args.n_est,
n_jobs=args.n_jobs,
                               max_samples=args.max_samples, max_features=args.max_features
                               , bootstrap=args.bootstrap, random_state=args.randseed)
else:
```

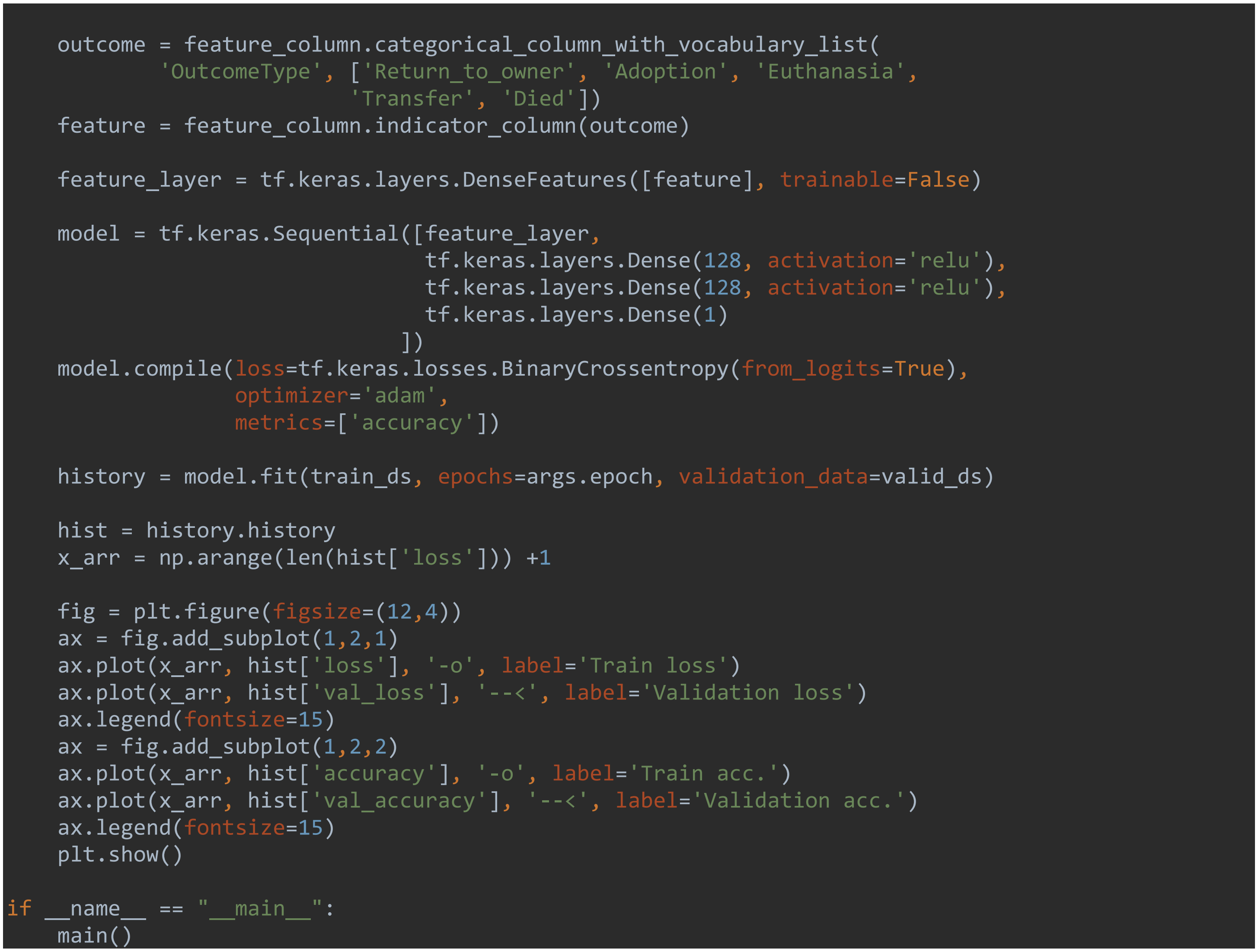


```
    print("ERROR: 'neurontype' selected is not supported")
    exit()


# ================================ Learning (time consuming)
===========================================================
print("Starting learning...")
before = time.time()

neuron.fit(X_train_use, y_train_use)

after = time.time()
print("Time to learn: ", (after-before), "seconds")


# ================================ Testing
=============================================================================
print("Starting testing...")
before = time.time()

y_train_pred = neuron.predict(X_train_use)
y_test_pred = neuron.predict(X_test_use)
print("Training Accuracy: ", accuracy_score(y_true=y_train_use, y_pred=y_train_pred))
print("Training Precision: ",precision_score(y_true=y_train_use,
y_pred=y_train_pred,average='macro'))
print("Training Recall: ",recall_score(y_true=y_train_use, y_pred=y_train_pred,average='macro'))
print("Training F1: ",f1_score(y_true=y_train_use, y_pred=y_train_pred,average='macro'))
print("    --------")
print("Testing Accuracy: ", accuracy_score(y_true=y_test_use, y_pred=y_test_pred))
print("Testing Precision: ", precision_score(y_true=y_test_use,
y_pred=y_test_pred,average='macro'))
print("Testing Recall: ", recall_score(y_true=y_test_use, y_pred=y_test_pred,average='macro'))
print("Testing F1: ", f1_score(y_true=y_test_use, y_pred=y_test_pred,average='macro'))

after = time.time()
print("Time to test: ", (after-before), "seconds")
```

Appendix C: Machine learning script – Neural Network

This script was used for the data processing discussed in this report. This script was run with the following command sequence:

```
#!/usr/bin/env python3
import argparse, csv, os
import numpy as np
import tensorflow as tf
import pandas as pd
from argparse import RawTextHelpFormatter
from sklearn.model_selection import train_test_split
from tensorflow import feature_column
import matplotlib.pyplot as plt

# Taken from Tensorflow documentation
def df_to_dataset(dataframe, shuffle=True, batch_size=32, name='Name'):
  dataframe = dataframe.copy()
  labels = dataframe.pop(name).values.reshape(-1,1)
  ds = tf.data.Dataset.from_tensor_slices((dict(dataframe), labels))
  if shuffle:
    ds = ds.shuffle(buffer_size=len(dataframe))
  ds = ds.batch(batch_size)
  return ds

def main():
    parser = argparse.ArgumentParser(formatter_class=RawTextHelpFormatter)
    parser.add_argument('-f', dest='csv', type=str, required=True,
            help='specify *relative* location of input data. (Expects CSV)\n')
    parser.add_argument('-b', dest='batch', type=int, required=False,
            default=100,
            help='specify batch size, default is 100\n')
    parser.add_argument('-s', dest='size', type=int, required=False,
            default=10000,
            help='specify buffer size, default is 10,000\n')
    parser.add_argument('-e', dest='epoch', type=int, required=False,
            default=10,
            help='specify the epoch, default is 10\n')
    args = parser.parse_args()

    df = pd.read_csv(args.csv, usecols=['AgeuponOutcome', 'AnimalType-Dog',
                                        'Name', 'SexuponOutcome-Female',
                                        'OutcomeType'])

    train, test = train_test_split(df, test_size=0.4)
    train, valid = train_test_split(train, test_size=0.4)

    '''
    le = LabelEncoder().fit(['Return_to_owner', 'Adoption', 'Euthanasia',
                           'Transfer', 'Died'])
    y_train = le.transform(y_train)
    y_test  = le.transform(y_test)
    '''


    train_ds = df_to_dataset(train, batch_size=args.size)
    valid_ds = df_to_dataset(valid, batch_size=args.size)
    test_ds  = df_to_dataset(test, batch_size=args.size)
```



```
    outcome = feature_column.categorical_column_with_vocabulary_list(
            'OutcomeType', ['Return_to_owner', 'Adoption', 'Euthanasia',
                           'Transfer', 'Died'])
    feature = feature_column.indicator_column(outcome)

    feature_layer = tf.keras.layers.DenseFeatures([feature], trainable=False)

    model = tf.keras.Sequential([feature_layer,
                                 tf.keras.layers.Dense(128, activation='relu'),
                                 tf.keras.layers.Dense(128, activation='relu'),
                                 tf.keras.layers.Dense(1)
                               ])
    model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
                  optimizer='adam',
                  metrics=['accuracy'])

    history = model.fit(train_ds, epochs=args.epoch, validation_data=valid_ds)

    hist = history.history
    x_arr = np.arange(len(hist['loss'])) +1

    fig = plt.figure(figsize=(12,4))
    ax = fig.add_subplot(1,2,1)
    ax.plot(x_arr, hist['loss'], '-o', label='Train loss')
    ax.plot(x_arr, hist['val_loss'], '--<', label='Validation loss')
    ax.legend(fontsize=15)
    ax = fig.add_subplot(1,2,2)
    ax.plot(x_arr, hist['accuracy'], '-o', label='Train acc.')
    ax.plot(x_arr, hist['val_accuracy'], '--<', label='Validation acc.')
    ax.legend(fontsize=15)
    plt.show()

if __name__ == "__main__":
    main()
```

Appendix D: Inputs conversion script

This script will take the downloaded “train.csv” and convert it into the various formats discussed in this report.

The results of this script can be downloaded from:
https://github.com/ghost8472/CS519_2020S_PS/tree/master/scratch/data

converter.php:

```
<?php

echo "Started at ".date("H:m:s");

$in_raw = file_get_contents('train.csv');
$in = explode("\n",$in_raw);

//helper function to get the Default Save Value
function DefSV($def,$obj,$key) {
        if (isset($obj[$key])) {
                return $obj[$key];
        } else {
                return $def;
        }
}

//luckily, the data is clean enough for doing dirty csv processing
$keyed = [];
$colmap = explode(",",$in[0]);
for($i = 1; $i < count($in); $i++) {
        $line = explode(",",$in[$i]);
        if (count($line) == 1) continue;

        $row = [];
        foreach ($colmap as $c=>$col) {
                $row[$col] = $line[$c];
        }
        $keyed[] = $row;
}
//echo "<pre>".var_export($keyed,true)."</pre>";

//Now, find all values, for enumeration, and/or column layout
$alls = [];
$splits = [];
foreach ($keyed as $i=>$row) { foreach($row as $col=>$val) {
        $alls[$col][$val] = $val;

        if ($col == "Breed") {
                $ex = explode("/",str_replace(" Mix","",$val));
                foreach($ex as $exv) { $splits[$col.'-'.$exv] = []; }
        } else if ($col == "Color") {
                $ex = explode("/",$val);
                foreach($ex as $exv) { $splits[$col.'-'.$exv] = []; }
        } else if ($col == "AnimalType") {
                $splits[$col.'-'.$val] = [];
        } else if ($col == "SexuponOutcome") {
                // ignore, done manually below
        } else {
                $splits[$col][$val] = $val;
        }
}}
//Add a few more columns to the extended feature version
$splits["SexuponOutcome-Female"] = [];
$splits["SexuponOutcome-Male"]   = [];
$splits["SexuponOutcome-Sterilized"] = [];
$splits["SexuponOutcome-Unknown"] = [];
$splits["Breed-Mix"] = [];
$splits["Color-Mix"] = [];
for($i = 1; $i <= 12; $i++) {
        $splits["DateTime-Month{$i}"] = [];
}
for ($i = 1; $i <= 7; $i++) {
        $splits["DateTime-DOW{$i}"] = [];
```

```
}
$splits["DateTime-Morning"] = [];
$splits["DateTime-Afternoon"] = [];
$splits["DateTime-Night"] = [];

//Get rid of columns we won't be keeping
unset($alls['AnimalID']);
unset($splits['AnimalID']);
unset($alls['OutcomeSubtype']);
unset($splits['OutcomeSubtype']);

ksort($alls);
ksort($splits);

//echo "<pre>".var_export($alls,true)."</pre>";
//echo "<pre>".var_export($splits,true)."</pre>";

//Go through each row, and each data cell, and transform it.
foreach ($keyed as $i=>&$row) { foreach($row as $col=>$val) {

        if ($col == "AnimalID") {
                unset($row[$col]); //this doesn't get used as an input, so would require
        }
        if ($col == "Name") {
                $row[$col] = ($val ? 1:0);
        }
        if ($col == "DateTime") {
                $row[$col] = strtotime($val);
                $ex = explode(" ",$val);
                $exh = explode(":",$ex[1]);  $hour = intval($exh[0]);
                $exm = explode("-",$ex[0]);  $mon  = intval($exm[1]);
                $row[$col.'-Month'.$mon] = 1;
                $row[$col.'-DOW'.date('N')] = 1;
                if ($hour >= 7 && $hour < 12) {
                        $row[$col.'-Morning'] = 1;
                } else if ($hour >= 12 && $hour <= 19) {
                        $row[$col.'-Afternoon'] = 1;
                } else { // $hour < 7 || $hour > 19
                        $row[$col.'-Night'] = 1;
                }
        }
        if ($col == "AgeuponOutcome") {
                $ex = explode(" ",$val);
                $ex[0] = intval($ex[0]);
                if ($val == "") {
                        $row[$col] = 7.77; //unknown value, distinct though
                } else if ($ex[1] == "years" || $ex[1] == "year") {
                        $row[$col] = $ex[0];
                } else if ($ex[1] == "months" || $ex[1] == "month") {
                        $row[$col] = $ex[0]/12;
                } else if ($ex[1] == "weeks" || $ex[1] == "week") {
                        $row[$col] = $ex[0]/52;
                } else if ($ex[1] == "days" || $ex[1] == "day") {
                        $row[$col] = $ex[0]/365;
                }
        }
        if ($col == "OutcomeType") {
                //no modification, final combiner will put this last
        }
        if ($col == "OutcomeSubtype") {
                unset($row[$col]); //this is not in the "test.csv" so we can't use it for the challenge
        }
        if ($col == "AnimalType") {
                $row[$col] = array_search($val,array_keys($alls[$col]));
                $row[$col.'-'.$val] = 1;
        }
        if ($col == "SexuponOutcome") {
                if ($val == "") { $val = "Unknown"; } //one blank to fill in
                $row[$col] = array_search($val,array_keys($alls[$col]));
                $row[$col.'-Unknown'] = ($val=="Unknown"?1:0);
                $row[$col.'-Sterilized'] = (strpos($val,"Neutered") !== strpos($val,'Spayed') ? 1:0); //both
"false" if Intact and not Unkown
                $row[$col.'-Male'] = (strpos($val,"Male") !== false ? 1:0);
                $row[$col.'-Female'] = (strpos($val,"Female") !== false ? 1:0);
        }
        if ($col == "Breed") {
                $row[$col] = array_search($val,array_keys($alls[$col]));
                $mix = (strpos($val," Mix") !== false ? 1:0);
                $ex = explode("/",str_replace(" Mix","",$val));
                $mix = (count($ex) > 1 ? 1:$mix);
                foreach($ex as $exv) { $row[$col.'-'.$exv] = 1; }
                $row[$col.'-Mix'] = $mix;
        }
        if ($col == "Color") {
                $row[$col] = array_search($val,array_keys($alls[$col]));
                $ex = explode("/",$val);
                $mix = (count($ex) > 1 ? 1:0);
                foreach($ex as $exv) { $row[$col.'-'.$exv] = 1; }
                $row[$col.'-Mix'] = $mix;
        }
}} unset($row);

//echo "<pre>".var_export($keyed,true)."</pre>";

//function to convert the definition array and data array into CSV ready lines
function Transform($define, $keyed, $header) {
        $conv = "";
        if ($header) {
                $head = [];
                foreach($define as $col=>$vals) {
                        if ($col == "OutcomeType") continue;
                        $head[] = $col;
                }
                $head[] = "OutcomeType";
                $conv .= implode(",",$head)."\n";
        }
        foreach ($keyed as $i=>$row) {
                $line = ""; $d = "";
                foreach($define as $col=>$vals) {
                        if ($col == "OutcomeType") continue;

                        $line .= $d.DefSV(0,$row,$col);
                        $d = ",";
                }
                $line .= ",".DefSV("",$row,"OutcomeType");

                $conv .= $line."\n";
        }
        return $conv;
}


//write the various versions to the files
file_put_contents("train_conversion_lowfeat_withheader.csv",Transform($alls,  $keyed, true));
file_put_contents("train_conversion_lowfeat_sansheader.csv",Transform($alls,  $keyed, false));
file_put_contents("train_conversion_extfeat_withheader.csv",Transform($splits,$keyed, true));
file_put_contents("train_conversion_extfeat_sansheader.csv",Transform($splits,$keyed, false));


echo "<br/>Finished at ".date("H:m:s");
```