Figure 1: *San Francisco 5* (2015)

Kaggle-Challenge: San Francisco Crime Classification

Nagaoka University of Technology

Date: January 9, 2019
Instructor: Professor Yukawa

Raphael Emberger

Contents

1. Preface

Firstly, I want to express my gratitude to Professor Yukawa for guiding me in this project and to the Kokusaika staff members to arrange my stay here at the Nagaoka University of Technology(subsequently referred to as ”NUT”). I was given the generous opportunity to study at the NUT for one semester, for which I am very grateful. During that time I could choose from the following six Kaggle challenges to work on as project work:

- Toxic Comment Classification Challenge (Kaggle 2017*b*)
- TalkingData AdTracking Fraud Detection Challenge (Kaggle 2018*b*)
- Quora Question Pairs (Kaggle 2017*a*)
- Expedia Hotel Recommendations (Kaggle 2016)
- San Francisco Crime Classification (Kaggle 2015)
- Inclusive Images Challenge (Kaggle 2018*a*)

Of those, I was most interested in the classification of reported crimes (Kaggle 2015), as in my opinion this was an interesting challenge, given the dataset to be only consisting of time and spatial data. As such, this report is dedicated to take on this challenge.

2. Abstract

The first attempt to classify crimes based on date-time, district and coordinates was to build a neural network using Keras (n.d.). This approach failed by remaining on the same level as always guessing the most prominent label(”Larceny/Theft”) - 20%. This reached rank 1058 out of 2335.

The second attempt reached better results. This time, finished projects for the same challenge were used as reference to find problems with the first approach. This time, a Bernoulli Naïve Bayes classifier was used on the binarized dataset and reached a log-loss of 2.464 or 26.02%, which raised the rank up to 675.

The third and last attempt consisted of integrating the first attempt of using Keras into the second attempt. After some adjustment, the rank could be slightly improved once more: With a log-loss of 2.456(26.39%), rank 664 could be claimed.

The time for more improvement was not available because too much time was wasted on the first attempt, but the next steps would have included data enrichment and data manipulation.

3. Introduction

3.1. Initial situation

The challenge has been out since roughly 3 years and since then, many teams have participated and submitted their results. This lead the leaderboard to fill up with 2335 submissions which were ranked and their results displayed online(see ”Leaderboard” at Kaggle (2015)). The results vary from 34.53877 up to 1.95936, where the sample submission with a value of 32.89183 reaches rank 2241(see 4.1 for the ranking principle).

When searching on the internet for documents about that challenge, there are multiple such projects to be found. For example:

- A paper from Darekar et al. (2016). 2 Naïve Bayes, Decision Tree, Random Forest and Support Vector Machines classifiers were used. Reached highest accuracy of 23.16% with a Decision Tree.
- A blog post from Ramunno-Johnson (2015). In that project, a Bernoulli Naïves Bayes classifier was used. Reached a log-loss score of 2.58.
- A blog post from Murray (n.d.). AdaBoost, Bagging, Extra Trees, Gradient Boosting, K-Nearest Neighbors, Random Forest classifiers and Logistic Regresson were used in this project. The dataset was enriched greatly by adding 9 other datasets (features like house prices, income, police and public transportation stations, healthcare center and homeless shelter locations, altitudes). Highest accuracy achieved with Gradient Boosted Trees, resulting in 45.7%.

3.2. Objective

The objective of this project is to produce a system that is capable of classifiying the type of crime based off of the provided data consisting of date time stamps, the name of the district and street as well as the coordinates of the registered report. To quote Kaggle (2015):

> From 1934 to 1963, San Francisco was infamous for housing some of the world’s most notorious criminals on the inescapable island of Alcatraz.
>
> Today, the city is known more for its tech scene than its criminal past. But, with rising wealth inequality, housing shortages, and a proliferation of expensive digital toys riding BART to work, there is no scarcity of crime in the city by the bay.
>
> From Sunset to SOMA, and Marina to Excelsior, this competition’s dataset provides nearly 12 years of crime reports from across all

> of San Francisco's neighborhoods. Given time and location, you must predict the category of crime that occurred.
>
> We're also encouraging you to explore the dataset visually. What can we learn about the city through visualizations like this Top Crimes Map? The top most up-voted scripts from this competition will receive official Kaggle swag as prizes.

Although the Kaggle challenge includes submitting an softmax array of the predictions of the test data, this objectives shifts towards self evaluation on the training set. The reason for this is that the challenge is already over and self evaluation was considered an easier approach to measure the success of the system.

4. Theoretical Principles

4.1. Loss Function

The ranking of the results on the Kaggle leader board(Kaggle 2015) are based on the multi-class logarithmic loss function[1]:

$$loss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log\left(p_{ij}\right) \tag{I}$$

N : Number of cases in dataset.
M : Number of classes.
y_{ij} : Label for class. 1 if i is in j. Otherwise 0.
p_{ij} : Predicted probability that i belongs to j.

This basically boils down to a format as follows:

Class 1	Class 2	Class 3
0.24	0.48	0.38

With the labels being:

Class 1	Class 2	Class 3
0.00	1.00	0.00

When those values are applied to I, we get a value of 0.49548. Of course, the closer the prediction is to the actual labels, the smaller the loss value will be.

To calculate examples quickly on the python console, the following code can be used:

```
1 import numpy as np
2 from sklearn.metrics import log_loss
3 labels = np.array([0.0, 1.0, 0.0])
4 prediction = np.array([0.04, 0.78, 0.18])
5 print(log_loss(labels, prediction))
```

Listing 1: Quick Log Loss Calculation in python

[1] The framework of Keras (n.d.) refers to the multi-class logarithmic loss function also as ”categorical cross-entropy”.

5. Methods

5.1. Dataset

The Kaggle challenge (Kaggle 2015) provides 3 files on their site under ”Data”:

- **sampleSubmission.csv**(884k x 40): A sample file, demonstrating the expected format for submissions to the challenge. Consists of an array of the softmax prediction of each sample in the **test.csv**.
- **test.csv**(884k x 7): The unlabeled sample subset of the data.
- **train.csv**(878k x 9): The labeled sample subset of the data.

The data itself consists of the gathered crime reports of the San Francisco Police Department from January 1st 2003 through May 13th 2015, where the odd weeks belong to the **test.csv** and the even weeks to **train.csv**.

Here are the 10 first rows of the respective data files:

Id	*one column per class*
0	*zeros, except the second last columns being all ones*
1	⋮
2	⋮
⋮	⋮

Table 1: sampleSubmission.csv(first 10 rows)

The two datasets differ slightly in their columns. The training dataset has added the labels(Category) but also Descript and Resolution, which will be ignored for this project.

Id	Dates	DayOfWeek	PdDistrict	Address	X	Y
0	2015-05-10 23:59:00	Sunday	BAYVIEW	2000 Block of THOMAS AV	-122.39958770418998	37.7350510103906
1	2015-05-10 23:51:00	Sunday	BAYVIEW	3RD ST / REVERE AV	-122.391522893042	37.7324323864471
2	2015-05-10 23:50:00	Sunday	NORTHERN	2000 Block of GOUGH ST	-122.426001954961	37.7922124386284
3	2015-05-10 23:45:00	Sunday	INGLESIDE	4700 Block of MISSION ST	-122.437393972517	37.7214120621391
4	2015-05-10 23:45:00	Sunday	INGLESIDE	4700 Block of MISSION ST	-122.437393972517	37.7214120621391
5	2015-05-10 23:40:00	Sunday	TARAVAL	BROAD ST / CAPITOL AV	-122.45902362242902	37.7131719025215
6	2015-05-10 23:30:00	Sunday	INGLESIDE	100 Block of CHENERY ST	-122.42561645123001	37.73935051446279
7	2015-05-10 23:30:00	Sunday	INGLESIDE	200 Block of BANKS ST	-122.41265203979201	37.739750156312105
8	2015-05-10 23:10:00	Sunday	MISSION	2900 Block of 16TH ST	-122.418700097043	37.7651649409646
9	2015-05-10 23:10:00	Sunday	CENTRAL	TAYLOR ST / GREEN ST	-122.413934584561	37.798886450641604

Table 2: test.csv(first 10 rows)

Dates	Category	Descript	DayOfWeek	PdDistrict	Resolution	Address	X	Y
2015-05-13 23:53:00	WARRANTS	WARRANT ARREST	Wednesday	NORTHERN	”ARREST, BOOKED”	OAK ST / LAGUNA ST	-122.425891675136	37.7745985956747
2015-05-13 23:53:00	OTHER OFFENSES	TRAFFIC VIOLATION ARREST	Wednesday	NORTHERN	”ARREST, BOOKED”	OAK ST / LAGUNA ST	-122.425891675136	37.7745985956747
2015-05-13 23:33:00	OTHER OFFENSES	TRAFFIC VIOLATION ARREST	Wednesday	NORTHERN	”ARREST, BOOKED”	VANNESS AV / GREENWICH ST	-122.42436302145	37.8004143219856
2015-05-13 23:30:00	LARCENY/THEFT	GRAND THEFT FROM LOCKED AUTO	Wednesday	NORTHERN	NONE	1500 Block of LOMBARD ST	-122.42699532676599	37.80087263276921
2015-05-13 23:30:00	LARCENY/THEFT	GRAND THEFT FROM LOCKED AUTO	Wednesday	PARK	NONE	100 Block of BRODERICK ST	-122.438737622757	37.771541172057795
2015-05-13 23:30:00	LARCENY/THEFT	GRAND THEFT FROM UNLOCKED AUTO	Wednesday	INGLESIDE	NONE	0 Block of TEDDY AV	-122.40325236121201	37.713430704116
2015-05-13 23:30:00	VEHICLE THEFT	STOLEN AUTOMOBILE	Wednesday	INGLESIDE	NONE	AVALON AV / PERU AV	-122.423326976668	37.7251380403778
2015-05-13 23:30:00	VEHICLE THEFT	STOLEN AUTOMOBILE	Wednesday	BAYVIEW	NONE	KIRKWOOD AV / DONAHUE ST	-122.371274317441	37.7275640719518
2015-05-13 23:00:00	LARCENY/THEFT	GRAND THEFT FROM LOCKED AUTO	Wednesday	RICHMOND	NONE	600 Block of 47TH AV	-122.508194031117	37.776601260681204
2015-05-13 23:00:00	LARCENY/THEFT	GRAND THEFT FROM LOCKED AUTO	Wednesday	CENTRAL	NONE	JEFFERSON ST / LEAVENWORTH ST	-122.419087676747	37.8078015516515

Table 3: train.csv(first 10 rows)

ARSON	ASSAULT	BAD CHECKS
BRIBERY	BURGLARY	DISORDERLY CONDUCT
DRIVING UNDER THE INFLUENCE	DRUG/NARCOTIC	DRUNKENNESS
EMBEZZLEMENT	EXTORTION	FAMILY OFFENSES
FORGERY/COUNTERFEITING	FRAUD	GAMBLING
KIDNAPPING	LARCENY/THEFT	LIQUOR LAWS
LOITERING	MISSING PERSON	NON-CRIMINAL
OTHER OFFENSES	PORNOGRAPHY/OBSCENE MAT	PROSTITUTION
RECOVERED VEHICLE	ROBBERY	RUNAWAY
SECONDARY CODES	SEX OFFENSES FORCIBLE	SEX OFFENSES NON FORCIBLE
STOLEN PROPERTY	SUICIDE	SUSPICIOUS OCC
TREA	TRESPASS	VANDALISM
VEHICLE THEFT	WARRANTS	WEAPON LAWS

Table 4: Crime classes

The classes occur in an unbalanced matter: ”Larceny/Theft” is the most predominant recorded crime, taking up nearly 19.92% of the dataset. For this reason, 19.92% is considered the bottom line of accuracy.

5.2. First Approach

For the first approach a Keras (n.d.) model on top of Tensorflow (n.d.) was chosen. For this, the first step was to pre-process the dataset to standardize it and properly feed it to the neural network.

5.2.1. Pre-Processing

To handle CSV files properly, a `CsvFile` class was created that represents a single csv file. When instantiated, it loads the csv file as a Pandas `DataFrame`. Apart from an abstract `def parse(self)`, it implements per-data-field functions that prepare the respective column for a conversion to a numerical representation(i.e. `def _prepare_date(self, date: datetime)`). It also defines an export function `def toNpArray(self) -> ndarray`, which allows to access the data as a `numpy` array.

From this basic class, three other classes were derived:

- `class TestDataCsvFile`
  This class represents the `test.csv` file. It implements the missing `parse(self)` method.
- `class TrainDataCsvFile`
  This class represents the sample part of the `train.csv` file. It implements the `parse(self)` method.
- `class TrainLabelsCsvFile`
  This class represents the label part of the `train.csv` file. When instantiating, it can make a link to an already existing `TrainDataCsvFile` class, to prevent loading the same csv file a second time. It implements the `parse(self)` method.

All the `CsvFile` derived classes support loading their already preprocessed data from the disk. This was implemented to reduce calculation time when performing multiple runs.

Feature Scaling and Mean Normalization Apart from converting the data into integers or floats, the `_prepare_X` functions also perform feature scaling(usually to a $[-1, 1]$ set - sometimes $[0, 1]$) and mean normalization(i.e. the coordinates X and Y).

5.2.2. Keras Model

To build the model, a dedicated `Model` class was created. This class operates as a Keras model factory, using the `get_model` to either create and

train a model or load it's weights and parameters from a file and return the model.

The layers of the model changed greatly over time. This is the the last version of the model:

```
28          model = keras.Sequential([
29              keras.layers.Dense(16, input_shape=(train_data.shape[1],),
    ↪ activation='relu'),
30              keras.layers.Dense(64, activation='relu'),
31              keras.layers.Dense(39, activation='softmax')
32          ])
33          self.log.info("Constructed model")
34          optimizer = keras.optimizers.Adam(lr=0.04)
35          model.compile(optimizer=optimizer,
36                        loss='sparse_categorical_crossentropy',
37                        metrics=['accuracy'])
38          self.log.info("Compiled model")
39
40          model.fit(train_data, train_labels, epochs=5, batch_size=200)
41          self.log.info("Trained model")
```

Listing 2: Keras model - model.py

5.2.3. Classification process

The classification process was written using the classes created in 5.2.1 and 2(5.2.2):

```
34 trainsamplesfile = TrainDataCsvFile()
35 trainlabelfile = TrainLabelsCsvFile(trainsamplesfile)
36 testsamplesfile = TestDataCsvFile()
37 for file in [trainsamplesfile, trainlabelfile, testsamplesfile]:
38     if args.prep_data or not file.prep_file_exists():
39         file.parse()
40         file.save()
41     else:
42         file.load()
43
44 if args.train:
45     mdl = Model().get_model(
46         trainsamplesfile.toNpArray(),
47         trainlabelfile.toNpArray()
48     )
49 else:
50     mdl = Model().get_model()
51
52 predictions = mdl.predict(trainsamplesfile.toNpArray())
53 print("LogLoss: {}".format(log_loss(trainlabelfile.toNpArray(),
    ↪ predictions)))
54 predicted_crime = np.argmax(predictions, axis=1)
55 print("Accuracy: {}%".format(accuracy_score(trainlabelfile.toNpArray(),
    ↪ predicted_crime) * 100))
```

```
56
57 for i in range(0, 19):
58     predicted = trainlabelfile.CATEGORIES[np.argmax(predictions[i])]
59     actual = trainlabelfile.get(i)
60     logging.info("{} ?= {}".format(actual, predicted))
```

Listing 3: Classification process(first approach) - main.py

5.2.4. Results

When using this setup, various settings did not raise the accuracy in any way. The accuracy value usually hovered barely below 20%, which coincides with the bottom line described in 5.1 and could at best get to rank 1058. It was suspected that bad preprocessing of the dataset as well as the lack of knowledge and experience in neural networks was the cause for the lack of progress.

After various unfruitful tries, this approach had to be abandoned because of the lack in progress due to lack of knowledge and experience with Keras.

5.3. Second Approach

After an input from Prof. Yukawa, a new approach was established. This step consisted of using finished projects for this challenge as reference. For this, the project of Ramunno-Johnson (2015) was chosen.

The system now relies on a Bernoulli Naïve Bayes classifier, which in contrast to the first approach 5.2 does not require the system to build up layers to construct. It also uses a Logistic Regression algorithm to have a bar for comparison to be able to determine progress.

5.3.1. Pre-processing

In this approach, the data was processed differently. To prevent the algorithm to search for patterns according to the principle of locality and simplify the data by binarization, columns were split up into several different dummy columns, namely:

- Minutes(m0 - m59)
- Hours(H1 - H12)
- Days(D1 - D31)
- Months(M1 - M12)
- Years(Y2003 - Y2015)
- Weekdays(Monday - Sunday)
- Districts(Bayview - Tenderloin)

```
30 def preprocess_dataframe(data: pd.DataFrame) -> Tuple[pd.DataFrame, list]:
31     print("Binarize data")
32     minute =
       ↪ pd.get_dummies(data.Dates.dt.minute).rename(columns=MINUTECOLUMNS)
33     hour = pd.get_dummies(data.Dates.dt.hour).rename(columns=HOURCOLUMNS)
34     day = pd.get_dummies(data.Dates.dt.day).rename(columns=DAYCOLUMNS)
35     month =
       ↪ pd.get_dummies(data.Dates.dt.month).rename(columns=MONTHCOLUMNS)
36     year = pd.get_dummies(data.Dates.dt.year).rename(columns=YEARCOLUMNS)
37     weekdays = pd.get_dummies(data.DayOfWeek)
38     districts = pd.get_dummies(data.PdDistrict)
39     x = data.X
40     y = data.Y
41     print("Assemble new array")
42     new_data = pd.concat([minute, hour, day, month, year, weekdays,
       ↪ districts, x, y], axis=1)
43     columns = new_data.keys().tolist()
44     return new_data, columns
```

Listing 4: Pre-processing method - newmain.py

```
print("Load Data with pandas, and parse the first column into datetime")
train = pd.read_csv('train.csv', parse_dates=['Dates'])
test = pd.read_csv('test.csv', parse_dates=['Dates'])

print("Convert crime labels to numbers")
le_crime = preprocessing.LabelEncoder()
crime = le_crime.fit_transform(train.Category)

print("Build training data")
train_data, features = preprocess_dataframe(train)
train_data['crime'] = crime

print("Features[{}]: {}".format(len(features), np.array(features)))

print("Split up training data")
# training, validation = train_test_split(train_data, test_size=.20)
training = train_data
validation = train_data


# Bernoulli Naïve Bayes
print("Train Bernoulli Naïve Bayes classifier")
air_bnb = BernoulliNB()
air_bnb.fit(training[features], training['crime'])

print("Predict labels")
predicted = air_bnb.predict_proba(validation[features])

print("Validate prediction")
evaluate(predicted, validation['crime'])
```

Listing 5: Bernoulli Naïve Bayes fitting- newmain.py

```
# Logistic Regression
print("Train Logistic Regression for comparison")
lr = LogisticRegression(C=0.1, solver='lbfgs', multi_class='multinomial')
lr.fit(training[features], training['crime'])

print("Predict labels")
predicted = np.array(lr.predict_proba(validation[features]))

print("Validate prediction")
evaluate(predicted, validation['crime'])
```

Listing 6: Logistic Regression fitting - newmain.py

This lead to an expansion of the number of columns from 6 to 159, dropping column "Address" and binarizing all the other columns except for the coordinates, which remain floats.

5.3.2. Results

This new classification system reached a value of 2.464 with the Bernoulli Naïve Bayes classifier. This log-loss value corresponds to an accuracy of 26.02%. For comparison, the Logistic Regression reached 2.591 with an accuracy of 24.43%. This result would correspond to rank 675 on the Kaggle leaderboard.

To finalize this project, it was decided to combine the findings of those two approaches to further advance in this challenge.

5.4. Third approach

Although the second approach 5.3 was at an acceptable level, the failed first attempt was decided to be merged into the second one as comparison and to draw conclusions about what went wrong the first time. For this the classes written in the first approach were completely abandoned and only the core code of the first approach was amended and integrated into the script of 5.3.

5.4.1. Results

The amended Keras model showed results on the same level as the Bernoulli Naïves Bays classifier from 5.2.3. After a few tries, occasionally even slightly better results were reached:

```
47 def evaluate(prediction, labels):
48     print("LogLoss: {}".format(log_loss(labels, prediction)))
49     predicted_crime = np.argmax(prediction, axis=1)
50     print("Accuracy: {}%".format(accuracy_score(labels, predicted_crime)
       ↪ * 100))
```

Listing 7: Evaluation method - newmain.py

```
95 print("Train Keras")
96 model = keras.Sequential([
97     keras.layers.Dense(80, input_dim=len(features), activation='relu'),
98     keras.layers.Dense(118, activation='relu'),
99     keras.layers.Dense(39, activation='softmax')
100 ])
101 optimizer = keras.optimizers.Adam(lr=0.01)
102 model.compile(optimizer=optimizer,
103               loss='sparse_categorical_crossentropy',
104               metrics=['sparse_categorical_accuracy'])
105 model.fit(training[features], training['crime'], epochs=5,
       ↪ batch_size=1024)
106
107 print("Predict labels")
108 predicted = model.predict_proba(validation[features])
109
110 print("Validate prediction")
111 evaluate(predicted, validation['crime'])
```

Listing 8: Keras model integrated - newmain.py

With this configuration the model reached a log-loss of 2.456 (accuracy: 26.39%), which was equivalent to rank 664. Which satisfies our definition of Done.

6. Results

Although the first approach lead to some measurable results(rank 1058), there were still many improvements to be made. Some of them were successfully implemented in the second approach and lead to an advance in rank to 675(log-loss: 2.464). After applying those findings into the first approach by merging both attempts, a further - although small - improvement could be made: The rank improved to 664(log-loss: 2.456).

7. Conclusion

During this project, a number of mistakes have been committed. The biggest one was investing too much time into the first attempt, instead of looking up finished projects for reference. However, this is only clear now in hindsight, as at the point in time, no disadvantage could be identified. Nevertheless did this loss in time cause the project to halt at rank 664(log-loss: 2.456), instead of advancing further due to further research.

Another problem was the lack of knowledge on the part of constructing neural networks with APIs like Keras. This lead to a number of unnecessary attempts on improvement on the wrong front-line. Much time was spent on implementing the conversion from data into integer representation, feature scaling and mean normalization, even though later in the process(see 5.3.1) it turned out that binarizing the data was much faster in both programming and execution and brought better results.

7.1. Future Considerations

For future projects of similar nature, a better theoretical foundation is required. However, the mistakes made in this project were very insightful and helped in understanding general processes of creating neural networks.

If more time was available, the next step would have been enriching the dataset by relying on additional statistics and data from the internet. Other cited projects have relied on this method as well and just careless perusal makes this step seem very promising. The provided dataset by Kaggle after all only included date-time, district name and coordinates(address as well, but this was ignored in this project).

Another point to consider would be to process the data more: For example evening out distributions that otherwise are heavily uneven could lead to the neural network to not just abandon those samples in favour of sample types that are more prominent.

8. Listings

References

Darekar, R., Dandona, R. & Sureshbabu, V. (2016), 'Predicting and Analysis of Crime in San Francisco'.
URL: *https://www.slideshare.net/SameerDarekar1/san-francisco-crime-analysis-classification-kaggle-contest* 5

Kaggle (2015), 'San Francisco Crime Classification'.
URL: *https://www.kaggle.com/c/sf-crime* 3, 5, 7, 8

Kaggle (2016), 'Expedia Hotel Recommendations'.
URL: *https://www.kaggle.com/c/expedia-hotel-recommendations* 3

Kaggle (2017*a*), 'Quora Question Pairs'.
URL: *https://www.kaggle.com/c/quora-question-pairs* 3

Kaggle (2017*b*), 'Toxic Comment Classification Challenge'.
URL: *https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge* 3

Kaggle (2018*a*), 'Inclusive Images Challenge'.
URL: *https://www.kaggle.com/c/inclusive-images-challenge* 3

Kaggle (2018*b*), 'TalkingData AdTracking Fraud Detection Challenge'.
URL: *https://www.kaggle.com/c/talkingdata-adtracking-fraud-detection* 3

Keras (n.d.), 'The Python Deep Learning library'.
URL: *https://keras.io/* 4, 7, 11

Murray, M. (n.d.), 'Classifying San Francisco Crime Incidents'.
URL: *http://mattmurray.net/classifying-san-francisco-crime-incidents/* 5

Ramunno-Johnson, D. (2015), 'Machine learning to predict San Francisco crime'.
URL: *http://efavdb.com/predicting-san-francisco-crimes/* 5, 14

San Francisco 5 (2015).
URL: *https://allhdwallpapers.com/san-francisco-new-hd-wallpapers-high-resolution/* 1, 21

Tensorflow (n.d.), ‘An open source machine learning framework for everyone’.
URL: *https://www.tensorflow.org/* 11

List of Figures

List of Tables

Listings

A. Appendix

```
#!/bin/env python3

import argparse
import logging

import tensorflow as tf
from sklearn.metrics import log_loss, accuracy_score
from tensorflow import keras

from model import Model
from preprocessor import *

logging.basicConfig(
    format='%(asctime)s %(levelname)-8s %(name)20s: %(message)s',
    datefmt='%Y-%m-%d %H:%M:%S',
    level=logging.DEBUG
)

logging.info("Tensorflow: {}".format(tf.__version__))
logging.info("Keras: {}".format(keras.__version__))
logging.info("Numpy: {}".format(np.version.full_version))

parser = argparse.ArgumentParser(description="Classifies crimes of San
    ↪ Francisco")
parser.add_argument('-t', '--train', action='store_true',
                    help="Create and train the model(default: load model
    ↪ from disk).")
parser.add_argument('-p', '--prep-data', action='store_true',
                    help="Preprocess data files even if the preprocessed
    ↪ data already exists.")
args = parser.parse_args()
if args.prep_data:
    logging.debug("Preparing data from csv files")
if args.train:
    logging.debug("Training model")

trainsamplesfile = TrainDataCsvFile()
trainlabelfile = TrainLabelsCsvFile(trainsamplesfile)
testsamplesfile = TestDataCsvFile()
for file in [trainsamplesfile, trainlabelfile, testsamplesfile]:
    if args.prep_data or not file.prep_file_exists():
        file.parse()
        file.save()
    else:
        file.load()

if args.train:
    mdl = Model().get_model(
        trainsamplesfile.toNpArray(),
        trainlabelfile.toNpArray()
```

```
    )
else:
    mdl = Model().get_model()

predictions = mdl.predict(trainsamplesfile.toNpArray())
print("LogLoss: {}".format(log_loss(trainlabelfile.toNpArray(),
    ↪ predictions)))
predicted_crime = np.argmax(predictions, axis=1)
print("Accuracy: {}%".format(accuracy_score(trainlabelfile.toNpArray(),
    ↪ predicted_crime) * 100))

for i in range(0, 19):
    predicted = trainlabelfile.CATEGORIES[np.argmax(predictions[i])]
    actual = trainlabelfile.get(i)
    logging.info("{} ?= {}".format(actual, predicted))

# for key in trainlabelfile.stats:
#     trainlabelfile.stats[key] /= trainlabelfile.count
#     print("{} : {}%".format(trainlabelfile.stats[key] * 100,
    ↪ trainlabelfile.CATEGORIES[key]))

# for i in range(0, 10):
#     # (date, _, _, address, lat, long) = data.get(i)
#     print(trainfile.get(i), trainlabelfile.get(i))
#     webbrowser.open(
#         "https://www.google.ch/maps/"
#         "@{},{},58m/data=!3m1!1e3".format(lat, long)
#     )
```

Listing 9: main.py

```
import logging as log

# from tensorflow import keras
import keras
from numpy import ndarray


class Model:
    file = "model.h5"
    log = None

    def __init__(self):
        self.log = log.getLogger(self.__class__.__name__)

    def _train(self, train_data, train_labels, test_data=None,
    ↪ test_labels=None):
        r"""Construct and train model

        :param train_data: The training data
        :type train_data: ndarray
        :param train_labels: The training labels
        :type train_labels: ndarray
```

```
        :param test_data: The test data for verification
        :type test_data: ndarray
        :param test_labels: The test labels for verification
        :type test_labels: ndarray
        :return:
        """
        model = keras.Sequential([
            keras.layers.Dense(16, input_shape=(train_data.shape[1],),
    ↪ activation='relu'),
            keras.layers.Dense(64, activation='relu'),
            keras.layers.Dense(39, activation='softmax')
        ])
        self.log.info("Constructed model")
        optimizer = keras.optimizers.Adam(lr=0.04)
        model.compile(optimizer=optimizer,
                      loss='sparse_categorical_crossentropy',
                      metrics=['accuracy'])
        self.log.info("Compiled model")

        model.fit(train_data, train_labels, epochs=5, batch_size=200)
        self.log.info("Trained model")

        if test_data is not None and test_labels is not None:
            test_loss, test_acc = model.evaluate(test_data, test_labels)
            self.log.info("Tested model")
            self.log.info("Test accuracy: {}".format(test_acc))

        model.save(self.file)
        self.log.info("Saved model")
        return model

    def get_model(self, train_data=None, train_labels=None,
    ↪ test_data=None, test_labels=None):
        if train_data is not None and train_labels is not None:
            return self._train(train_data, train_labels, test_data,
    ↪ test_labels)
        else:
            self.log.debug("Loading model")
            return keras.models.load_model(self.file, compile=False)
```

Listing 10: model.py

```
#!/bin/env python3

import logging as log
import os
import sys
from datetime import datetime

import numpy as np
import pandas as pd
import pytz

```

```
12
13 class CsvFile:
14     filename = ''
15     df: pd.DataFrame = None
16     df_orig: pd.DataFrame = None
17     log = None
18     COLS = [
19         'Id',
20         'Dates',
21         'Year',
22         'Month',
23         'Day',
24         'Hour',
25         'Minute',
26         'Weekday',
27         'Season',
28         'Daynight',
29         'DayOfWeek',
30         'PdDistrict',
31         'Address',
32         'X',
33         'Y'
34     ]
35     DAYS = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
    ↪ 'Saturday', 'Sunday']
36     DISTRICTS = ['BAYVIEW', 'NORTHERN', 'INGLESIDE', 'TARAVAL',
    ↪ 'MISSION', 'CENTRAL', 'TENDERLOIN', 'RICHMOND',
37                  'SOUTHERN', 'PARK']
38     min_date = pd.Timestamp('1/1/2003 00:00:00')
39     max_date = pd.Timestamp('1/1/2016 00:00:00')
40     sf_tz = pytz.timezone('US/Pacific')
41
42     def __init__(self, filename, csvfile=None):
43         self.log = log.getLogger(self.__class__.__name__)
44         self.filename = filename
45         file = self.filename + '.csv'
46         if csvfile is not None:
47             self.df_orig = csvfile.df_orig
48             self.log.info("Linked to data frame of '{}' with shape
    ↪ {}".format(csvfile.filename, self.df_orig.shape))
49         else:
50             self.log.debug("Reading csv file '{}'".format(file))
51             self.df_orig = self._read_file(file)
52             self.log.info("Read data frame of shape
    ↪ {}".format(self.df_orig.shape))
53
54     def _read_file(self, file):
55         raise NotImplementedError()
56
57     def _read_prep_file(self, file):
58         return pd.read_csv(
59             file,
```

```
            index_col='Id',
            keep_date_col=True,
            parse_dates=["Dates"]
        )

    def parse(self):
        raise NotImplementedError()

    def _prepare_date(self, date: datetime):
        # date = datetime.strptime(date, '%Y-%m-%d %H:%M:%S')
        # date = date.astimezone(self.sf_tz)
        max_delta = self.max_date - self.min_date
        delta = date - self.min_date
        return int(delta.total_seconds())/max_delta.total_seconds()

    def _transform_date(self):
        def get_season(month: int):
            if month < 3 or month == 12:
                return 0  # December - February: Winter
            if month < 6:
                return 1  # March - May: Spring
            if month < 9:
                return 2  # June - August: Summer
            if month < 12:
                return 3  # September - November: Autumn
            raise Exception()

        def get_day_night(hour: int):
            if hour <= 7 or hour >= 7:
                return 0  # Night
            return 1  # Day

        self.log.debug("Parsing Year")
        self.df['Year'] = self.df['Dates'].apply(lambda x: x.year)
        self.log.debug("Parsing Month")
        self.df['Month'] = self.df['Dates'].apply(lambda x: x.month)
        self.log.debug("Parsing Day")
        self.df['Day'] = self.df['Dates'].apply(lambda x: x.day)
        self.log.debug("Parsing Hour")
        self.df['Hour'] = self.df['Dates'].apply(lambda x: x.hour)
        self.log.debug("Parsing Minute")
        self.df['Minute'] = self.df['Dates'].apply(lambda x: x.minute)
        self.log.debug("Parsing Weekday")
        self.df['Weekday'] = self.df['Dates'].apply(lambda x:
    ↪ x.isoweekday())
        self.log.debug("Parsing Season")
        self.df['Season'] = self.df['Dates'].apply(lambda x:
    ↪ get_season(x.month))
        self.log.debug("Parsing Daynight")
        self.df['Daynight'] = self.df['Dates'].apply(lambda x:
    ↪ get_day_night(x.day))
        self.log.debug("Parsing Dates")
```

```
109         self.df['Dates'] = self.df['Dates'].apply(self._prepare_date)
110
111     def _prepare_day(self, daystr):
112         return self.DAYS.index(daystr)/(len(self.DAYS)/2)-1
113
114     def _prepare_district(self, daystr):
115         return self.DISTRICTS.index(daystr)/(len(self.DISTRICTS)/2)-1
116
117     @staticmethod
118     def _prepare_address(addressstr):
119         return hash(addressstr)/(sys.maxsize/2)-1
120
121     @staticmethod
122     def _prepare_latitude(latitudestr):
123         return float(latitudestr)/180
124
125     @staticmethod
126     def _prepare_longitude(longitudestr):
127         return float(longitudestr)/180
128
129     def save(self):
130         if self.df is None:
131             self.log.error("Data not yet parsed")
132             return
133         file = self._prep_file()
134         self.log.debug("Writing csv file '{}'".format(file))
135         self.df.to_csv(
136             file,
137             index_label='Id'
138         )
139         self.log.info("Wrote data frame of shape
    ↪ {}".format(self.df.shape))
140
141     def _prep_file(self):
142         return self.filename + '_prep.csv'
143
144     def prep_file_exists(self):
145         return os.path.isfile(self._prep_file())
146
147     def load(self):
148         file = self._prep_file()
149         self.log.debug("Reading csv file '{}'".format(file))
150         self.df = self._read_prep_file(file)
151         self.log.info("Read data frame of shape
    ↪ {}".format(self.df_orig.shape))
152
153     def get(self, index):
154         if self.df is None:
155             self.log.error("Data not yet parsed")
156             return
157         date = self.df_orig['Dates'][index]
158         # date = datetime.strptime(self.df_old.at[index, 'Dates'],
```

```
    ↪ '%Y-%m-%d %H:%M:%S')
159         day = self.df_orig['DayOfWeek'][index]
160         # day = self.df_old.at[index, 'DayOfWeek']
161         district = self.df_orig['PdDistrict'][index]
162         # district = self.df_old.at[index, 'DayOfWeek']
163         address = self.df_orig['Address'][index]
164         latitude = float(self.df_orig['Y'][index])
165         longitude = float(self.df_orig['X'][index])
166         return date, day, district, address, latitude, longitude
167 
168     def toNpArray(self):
169         return np.reshape(self.df.values, self.df.shape)
170 
171 
172 class TestDataCsvFile(CsvFile):
173 
174     def __init__(self):
175         super().__init__("test")
176 
177     def _prep_file(self):
178         return self.filename + '_samples.csv'
179 
180     def _read_file(self, file):
181         return pd.read_csv(
182             file,
183             index_col='Id',
184             keep_date_col=True,
185             parse_dates=["Dates"]
186         )
187 
188     def parse(self):
189         self.df = self.df_orig.copy()
190         self.log.debug('Parsing Dates')
191         self._transform_date()
192         self.log.debug('Parsing Day of the week')
193         self.df['DayOfWeek'] =
    ↪ self.df['DayOfWeek'].apply(self._prepare_day)
194         self.log.debug('Parsing District')
195         self.df['PdDistrict'] =
    ↪ self.df['PdDistrict'].apply(self._prepare_district)
196         self.log.debug('Parsing Address')
197         self.df['Address'] =
    ↪ self.df['Address'].apply(self._prepare_address)
198         self.log.debug('Parsing Longitude')
199         self.df['X'] = self.df['X'].apply(self._prepare_longitude)
200         self.log.debug('Parsing Latitude')
201         self.df['Y'] = self.df['Y'].apply(self._prepare_latitude)
202         self.log.info('Parsed dataframe')
203 
204 
205 class TrainDataCsvFile(CsvFile):
206 
```

```
207     def __init__(self, csvfile=None):
208         super().__init__("train", csvfile)
209 
210     def _prep_file(self):
211         return self.filename + '_samples.csv'
212 
213     def _read_file(self, file):
214         return pd.read_csv(
215             file,
216             keep_date_col=True,
217             parse_dates=["Dates"]
218         )
219 
220     def parse(self):
221         self.df = self.df_orig.copy()
222         # print(self.df.shape)
223         # print(self.df['Dates'].shape)
224         # print(self.df[['Dates']].shape)
225         # print(self.df.at[1, 'Dates'])
226         # print(self.df.at[1, 'Dates'].minute)
227         # exit(0)
228         self.log.debug('Parsing Dates')
229         self._transform_date()
230         self.log.debug('Deleting Category')
231         self.df = self.df.drop('Category', axis=1)
232         # self.log.debug('Parsing Category')
233         # self.df['Category'] =
    ↪ self.df['Category'].apply(self._prepare_category)
234         self.log.debug('Deleting Descript')
235         self.df = self.df.drop('Descript', axis=1)
236         self.log.debug('Parsing Day of the week')
237         self.df['DayOfWeek'] =
    ↪ self.df['DayOfWeek'].apply(self._prepare_day)
238         self.log.debug('Parsing District')
239         self.df['PdDistrict'] =
    ↪ self.df['PdDistrict'].apply(self._prepare_district)
240         self.log.debug('Deleting Resolution')
241         self.df = self.df.drop('Resolution', axis=1)
242         self.log.debug('Parsing Address')
243         self.df['Address'] =
    ↪ self.df['Address'].apply(self._prepare_address)
244         self.log.debug('Parsing Longitude')
245         self.df['X'] = self.df['X'].apply(self._prepare_longitude)
246         self.log.debug('Parsing Latitude')
247         self.df['Y'] = self.df['Y'].apply(self._prepare_latitude)
248         self.log.info('Parsed dataframe')
249 
250 
251 class TrainLabelsCsvFile(CsvFile):
252     CATEGORIES = ['ARSON', 'ASSAULT', 'BAD CHECKS', 'BRIBERY',
    ↪ 'BURGLARY', 'DISORDERLY CONDUCT',
253                   'DRIVING UNDER THE INFLUENCE', 'DRUG/NARCOTIC',
```

```
    ↪ 'DRUNKENNESS', 'EMBEZZLEMENT', 'EXTORTION',
254                 'FAMILY OFFENSES', 'FORGERY/COUNTERFEITING', 'FRAUD',
    ↪ 'GAMBLING', 'KIDNAPPING', 'LARCENY/THEFT',
255                 'LIQUOR LAWS', 'LOITERING', 'MISSING PERSON',
    ↪ 'NON-CRIMINAL', 'OTHER OFFENSES',
256                 'PORNOGRAPHY/OBSCENE MAT', 'PROSTITUTION', 'RECOVERED
    ↪ VEHICLE', 'ROBBERY', 'RUNAWAY',
257                 'SECONDARY CODES', 'SEX OFFENSES FORCIBLE', 'SEX
    ↪ OFFENSES NON FORCIBLE', 'STOLEN PROPERTY', 'SUICIDE',
258                 'SUSPICIOUS OCC', 'TREA', 'TRESPASS', 'VANDALISM',
    ↪ 'VEHICLE THEFT', 'WARRANTS', 'WEAPON LAWS']
259     stats = {}
260     count = 0
261
262     def __init__(self, csvfile=None):
263         super().__init__("train", csvfile)
264
265     def _prep_file(self):
266         return self.filename + '_labels.csv'
267
268     def _read_file(self, file):
269         return pd.read_csv(file)
270
271     def _read_prep_file(self, file):
272         return pd.read_csv(
273             file,
274             index_col='Id'
275         )
276
277     def _prepare_category(self, category):
278         cat = self.CATEGORIES.index(category)
279         if not cat in self.stats:
280             self.stats[cat] = 0.0
281         self.stats[cat] += 1.0
282         self.count += 1
283         return cat
284
285     def parse(self):
286         self.df = self.df_orig.copy()
287         # self.log.debug('Deleting Id')
288         # self.df = self.df.drop('Id', axis=1)
289         self.log.debug('Deleting Dates')
290         self.df = self.df.drop('Dates', axis=1)
291         self.log.debug('Parsing Category')
292         self.df['Category'] =
    ↪ self.df['Category'].apply(self._prepare_category)
293         self.log.debug('Deleting Descript')
294         self.df = self.df.drop('Descript', axis=1)
295         self.log.debug('Deleting DayOfWeek')
296         self.df = self.df.drop('DayOfWeek', axis=1)
297         self.log.debug('Deleting PdDistrict')
298         self.df = self.df.drop('PdDistrict', axis=1)
```

```
299         self.log.debug('Deleting Resolution')
300         self.df = self.df.drop('Resolution', axis=1)
301         self.log.debug('Deleting Address')
302         self.df = self.df.drop('Address', axis=1)
303         self.log.debug('Deleting Longitude')
304         self.df = self.df.drop('X', axis=1)
305         self.log.debug('Deleting Latitude')
306         self.df = self.df.drop('Y', axis=1)
307         self.log.debug('Parsed dataframe')
308 
309     def get(self, index):
310         if self.df is None:
311             self.log.error("Data not yet parsed")
312             return
313         district = self.CATEGORIES[self.df.at[index, 'Category']]
314         return district
```

Listing 11: preprocessor.py

```
1  #!/bin/env python3
2  
3  import keras
4  import numpy as np
5  import pandas as pd
6  from sklearn import preprocessing
7  from sklearn.linear_model import LogisticRegression
8  from sklearn.metrics import log_loss, accuracy_score
9  from sklearn.model_selection import train_test_split
10 from sklearn.naive_bayes import BernoulliNB
11 from typing import Tuple
12 
13 MINUTECOLUMNS = {}
14 for min_int in range(0, 60):
15     MINUTECOLUMNS[min_int] = "m{}".format(min_int)
16 HOURCOLUMNS = {}
17 for hour_int in range(0, 24):
18     HOURCOLUMNS[hour_int] = "H{}".format(hour_int)
19 DAYCOLUMNS = {}
20 for day_int in range(0, 30):
21     DAYCOLUMNS[day_int] = "D{}".format(day_int + 1)
22 MONTHCOLUMNS = {}
23 for month_int in range(0, 11):
24     MONTHCOLUMNS[month_int] = "M{}".format(month_int + 1)
25 YEARCOLUMNS = {}
26 for year_int in range(2003, 2015):
27     YEARCOLUMNS[year_int] = "Y{}".format(year_int)
28 
29 
30 def preprocess_dataframe(data: pd.DataFrame) -> Tuple[pd.DataFrame, list]:
31     print("Binarize data")
32     minute =
    ↪ pd.get_dummies(data.Dates.dt.minute).rename(columns=MINUTECOLUMNS)
33     hour = pd.get_dummies(data.Dates.dt.hour).rename(columns=HOURCOLUMNS)
```

```
    day = pd.get_dummies(data.Dates.dt.day).rename(columns=DAYCOLUMNS)
    month =
    ↪ pd.get_dummies(data.Dates.dt.month).rename(columns=MONTHCOLUMNS)
    year = pd.get_dummies(data.Dates.dt.year).rename(columns=YEARCOLUMNS)
    weekdays = pd.get_dummies(data.DayOfWeek)
    districts = pd.get_dummies(data.PdDistrict)
    x = data.X
    y = data.Y
    print("Assemble new array")
    new_data = pd.concat([minute, hour, day, month, year, weekdays,
    ↪ districts, x, y], axis=1)
    columns = new_data.keys().tolist()
    return new_data, columns


def evaluate(prediction, labels):
    print("LogLoss: {}".format(log_loss(labels, prediction)))
    predicted_crime = np.argmax(prediction, axis=1)
    print("Accuracy: {}%".format(accuracy_score(labels, predicted_crime)
    ↪ * 100))


print("Load Data with pandas, and parse the first column into datetime")
train = pd.read_csv('train.csv', parse_dates=['Dates'])
test = pd.read_csv('test.csv', parse_dates=['Dates'])

print("Convert crime labels to numbers")
le_crime = preprocessing.LabelEncoder()
crime = le_crime.fit_transform(train.Category)

print("Build training data")
train_data, features = preprocess_dataframe(train)
train_data['crime'] = crime

print("Features[{}]: {}".format(len(features), np.array(features)))

print("Split up training data")
# training, validation = train_test_split(train_data, test_size=.20)
training = train_data
validation = train_data


# Bernoulli Naïve Bayes
print("Train Bernoulli Naïve Bayes classifier")
air_bnb = BernoulliNB()
air_bnb.fit(training[features], training['crime'])

print("Predict labels")
predicted = air_bnb.predict_proba(validation[features])

print("Validate prediction")
evaluate(predicted, validation['crime'])
```

```
83
84 # Predict crimes of test dataset
85 print("Build test data")
86 test_data, _ = preprocess_dataframe(test)
87
88 print("Predict test labels")
89 predicted = air_bnb.predict_proba(test_data[features])
90
91 print("Write results")
92 result = pd.DataFrame(predicted, columns=le_crime.classes_)
93 result.to_csv('testResult.csv', index=True, index_label='Id')
94
95 print("Train Keras")
96 model = keras.Sequential([
97     keras.layers.Dense(80, input_dim=len(features), activation='relu'),
98     keras.layers.Dense(118, activation='relu'),
99     keras.layers.Dense(39, activation='softmax')
100 ])
101 optimizer = keras.optimizers.Adam(lr=0.01)
102 model.compile(optimizer=optimizer,
103               loss='sparse_categorical_crossentropy',
104               metrics=['sparse_categorical_accuracy'])
105 model.fit(training[features], training['crime'], epochs=5,
    ↪ batch_size=1024)
106
107 print("Predict labels")
108 predicted = model.predict_proba(validation[features])
109
110 print("Validate prediction")
111 evaluate(predicted, validation['crime'])
112
113
114 # Logistic Regression
115 print("Train Logistic Regression for comparison")
116 lr = LogisticRegression(C=0.1, solver='lbfgs', multi_class='multinomial')
117 lr.fit(training[features], training['crime'])
118
119 print("Predict labels")
120 predicted = np.array(lr.predict_proba(validation[features]))
121
122 print("Validate prediction")
123 evaluate(predicted, validation['crime'])
```

Listing 12: newmain.py