Real World Machine Learning

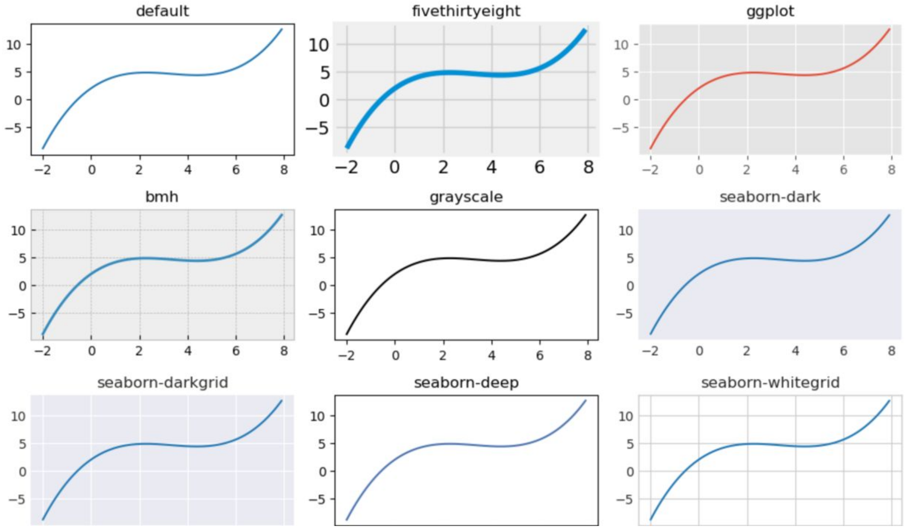
TJ Machine Learning Club

Introduction

- So far in TJML:
  - Decision Tree
  - Random Forest
  - SVM
  - KNN
  - Naive Bayes
  - Neural Network
- How do data scientists today code these?

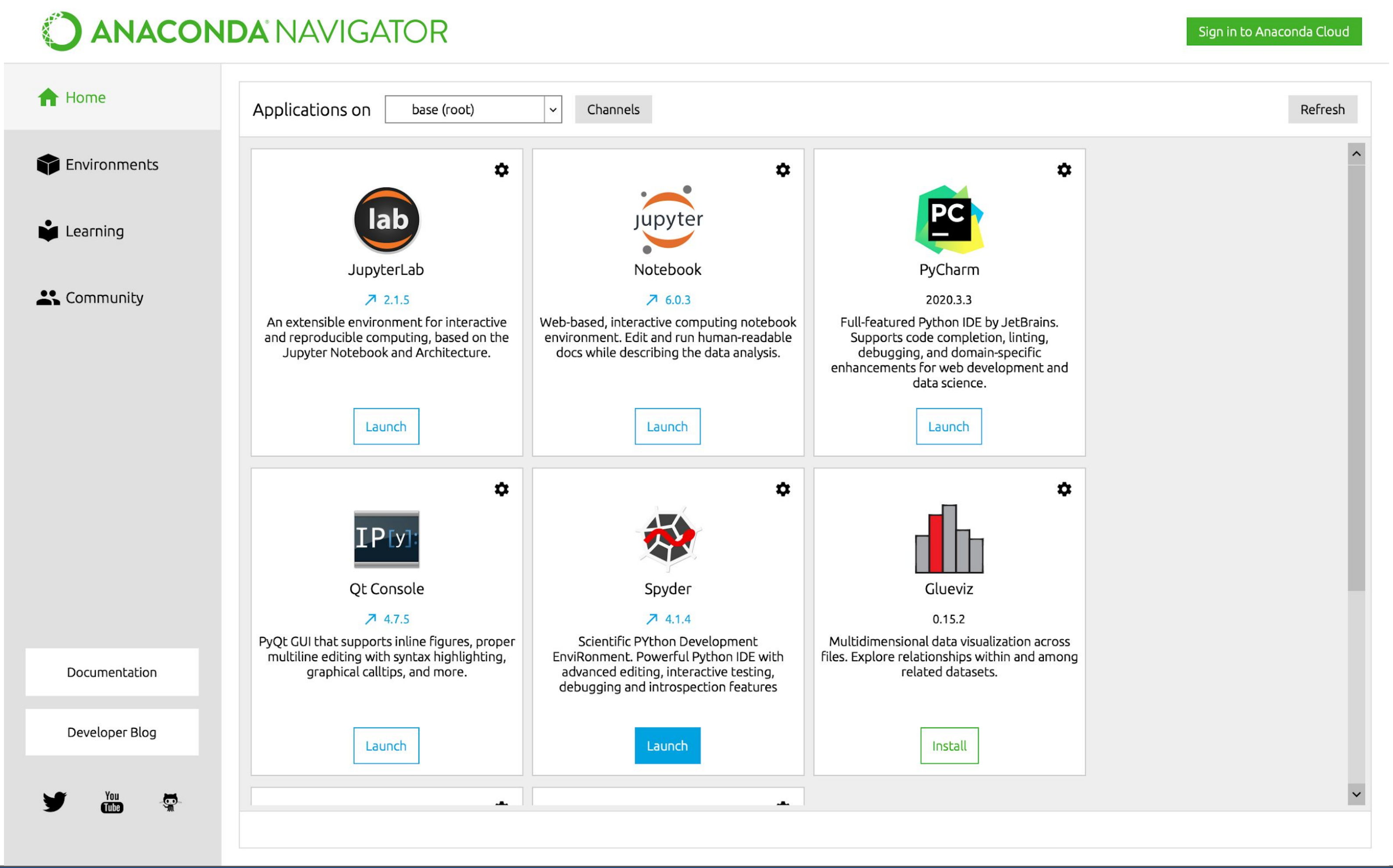
Setup

- Visualizing data:
  - Excel
  - Matplotlib
  - Seaborn
- Helpful data analysis libraries:
  - Pandas
  - Numpy



Conda

- Open-source Python package manager designed for data scientists.
- Can be installed through Anaconda, which bundles the package manager with 150+ data science packages for convenience.
    - Tensorflow, Keras, sklearn
- Another more efficient way to install Conda is with Miniconda, which will just install Conda and everything it needs to run, letting you decide what packages to install.
- Data scientists generally choose Conda over pip because it groups package installations into separate environments, which prevents conflicts between packages intended for separate projects.



JupyterLab

2.1.5

An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Launch

Notebook

6.0.3

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Launch

PyCharm

2020.3.3

Full-featured Python IDE by JetBrains. Supports code completion, linting, debugging, and domain-specific enhancements for web development and data science.

Launch

Qt Console

4.7.5

PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

Launch

Spyder

4.1.4

Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

Launch

Glueviz

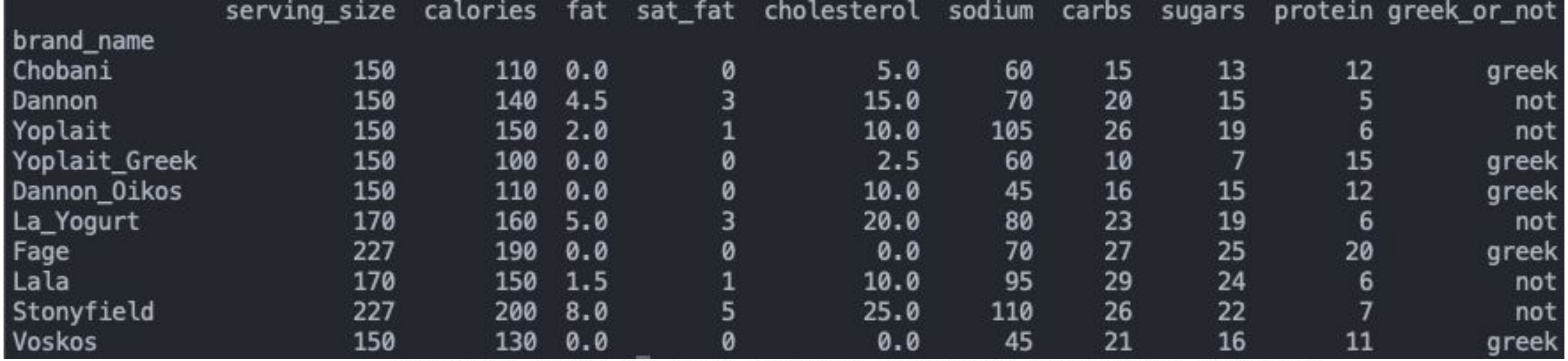
0.15.2

Multidimensional data visualization across files. Explore relationships within and among related datasets.

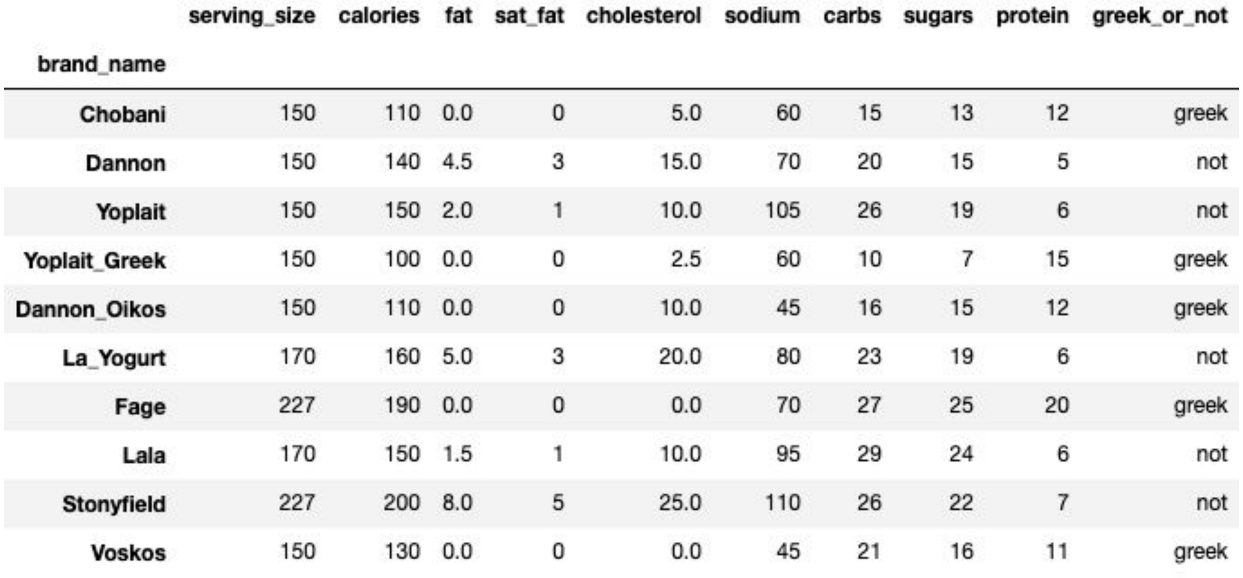
Install

Jupyter Notebooks

- An open-source web application that allows us to create and share codes and documents.
- It provides an environment, where you can document your code, run it, look at the outcome, visualize data and see the results without leaving the environment.
- Code can be written in individual cells that are run separately



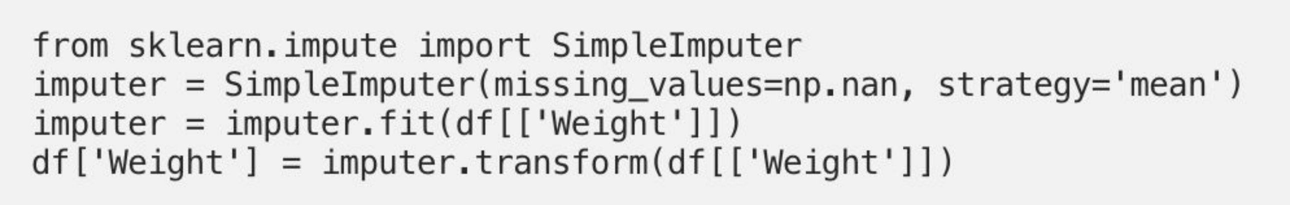
```
              serving_size  calories  fat  sat_fat  cholesterol  sodium  carbs  sugars  protein greek_or_not
brand_name
Chobani                150       110  0.0        0          5.0      60     15      13       12        greek
Dannon                 150       140  4.5        3         15.0      70     20      15        5          not
Yoplait                150       150  2.0        1         10.0     105     26      19        6          not
Yoplait_Greek          150       100  0.0        0          2.5      60     10       7       15        greek
Dannon_Oikos           150       110  0.0        0         10.0      45     16      15       12        greek
La_Yogurt              170       160  5.0        3         20.0      80     23      19        6          not
Fage                   227       190  0.0        0          0.0      70     27      25       20        greek
Lala                   170       150  1.5        1         10.0      95     29      24        6          not
Stonyfield             227       200  8.0        5         25.0     110     26      22        7          not
Voskos                 150       130  0.0        0          0.0      45     21      16       11        greek
```



	serving_size	calories	fat	sat_fat	cholesterol	sodium	carbs	sugars	protein	greek_or_not
brand_name										
Chobani	150	110	0.0	0	5.0	60	15	13	12	greek
Dannon	150	140	4.5	3	15.0	70	20	15	5	not
Yoplait	150	150	2.0	1	10.0	105	26	19	6	not
Yoplait_Greek	150	100	0.0	0	2.5	60	10	7	15	greek
Dannon_Oikos	150	110	0.0	0	10.0	45	16	15	12	greek
La_Yogurt	170	160	5.0	3	20.0	80	23	19	6	not
Fage	227	190	0.0	0	0.0	70	27	25	20	greek
Lala	170	150	1.5	1	10.0	95	29	24	6	not
Stonyfield	227	200	8.0	5	25.0	110	26	22	7	not
Voskos	150	130	0.0	0	0.0	45	21	16	11	greek

Data Preprocessing

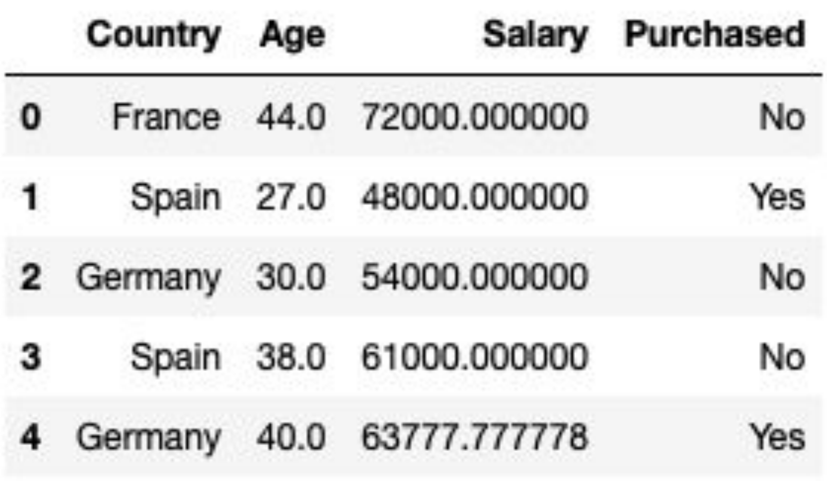
- Handling null values
  - df.isnull()
  - Returns a boolean matrix, if the value is NaN then True otherwise False
  - df.dropna()
- Imputation
  - Process of substituting the missing values of our dataset



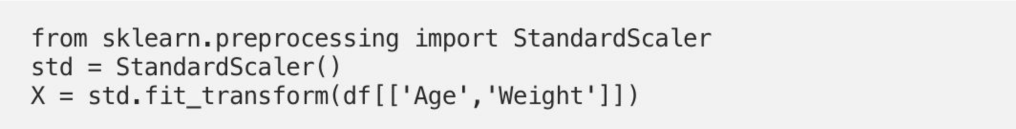
```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
imputer = imputer.fit(df[['Weight']])
df['Weight'] = imputer.transform(df[['Weight']])
```

Data Preprocessing

- Standardization:
  - Transform values such that the mean of the values is 0 and the standard deviation is 1.



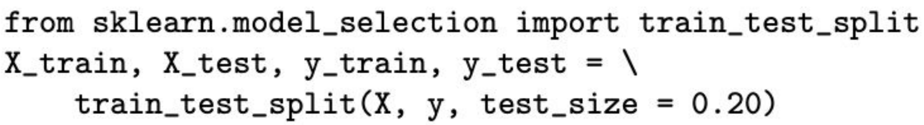
	Country	Age	Salary	Purchased
0	France	44.0	72000.000000	No
1	Spain	27.0	48000.000000	Yes
2	Germany	30.0	54000.000000	No
3	Spain	38.0	61000.000000	No
4	Germany	40.0	63777.777778	Yes



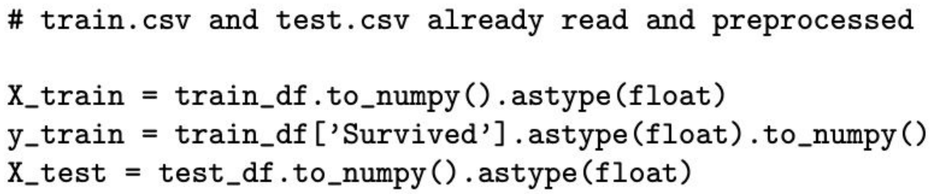
```
from sklearn.preprocessing import StandardScaler
std = StandardScaler()
X = std.fit_transform(df[['Age','Weight']])
```

Scikit-Learn

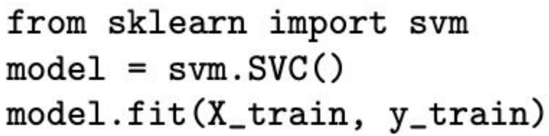
- Python library that provides many unsupervised and supervised learning algorithms
    - Regression, including Linear and Logistic Regression
    - Classification, including K-Nearest Neighbors
    - Clustering, including K-Means and K-Means++
    - Model selection
    - Preprocessing, including Min-Max Normalization



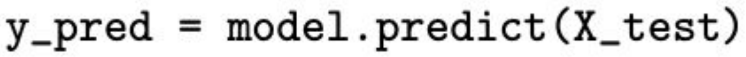
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size = 0.20)
```



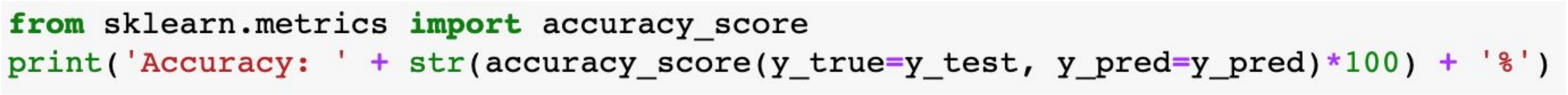
```
# train.csv and test.csv already read and preprocessed

X_train = train_df.to_numpy().astype(float)
y_train = train_df['Survived'].astype(float).to_numpy()
X_test = test_df.to_numpy().astype(float)
```



```
from sklearn import svm
model = svm.SVC()
model.fit(X_train, y_train)
```



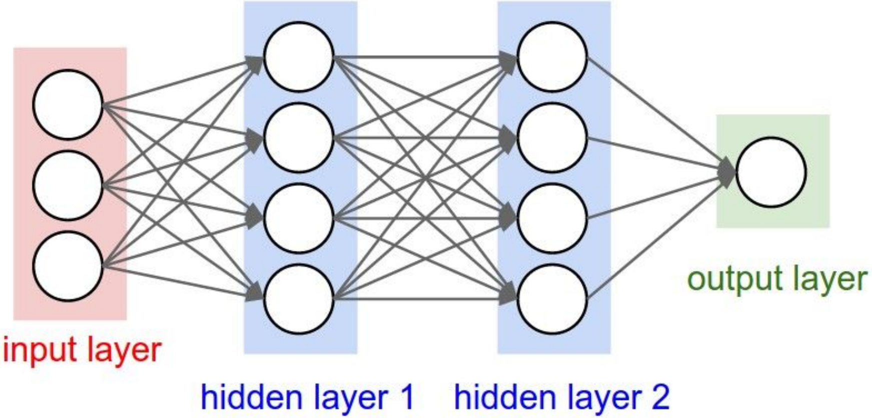
```
y_pred = model.predict(X_test)
```



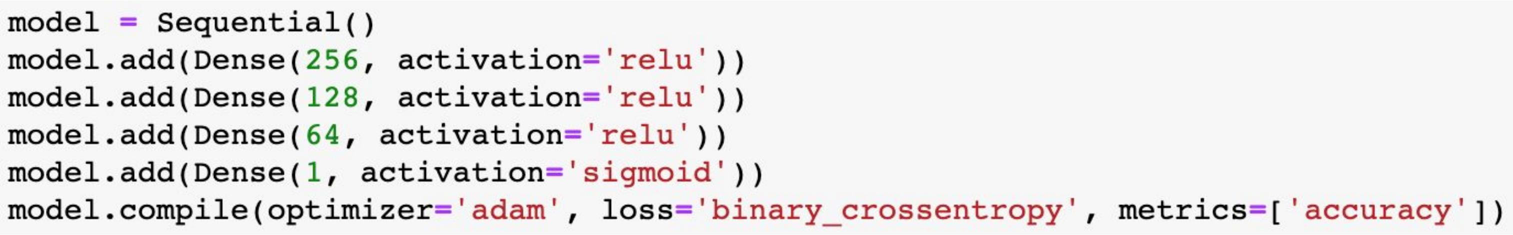
```
from sklearn.metrics import accuracy_score
print('Accuracy: ' + str(accuracy_score(y_true=y_test, y_pred=y_pred)*100) + '%')
```

Keras

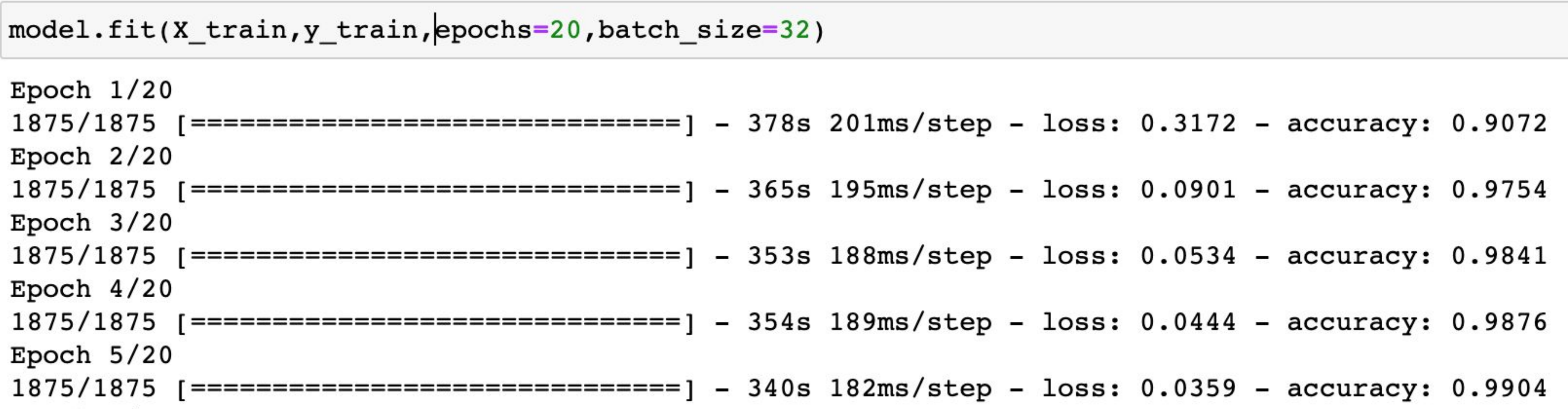
- One of the most common deep learning frameworks
- Uses Tensorflow backend
  - Used for Deep Learning
  - Neural Networks
  - CNNs



Neural Networks with Keras



```
model = Sequential()
model.add(Dense(256, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```



```
model.fit(X_train,y_train,epochs=20,batch_size=32)
```

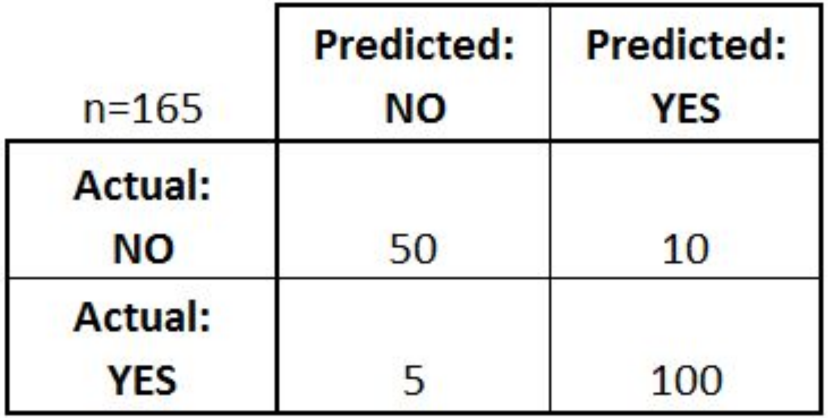
```
Epoch 1/20
1875/1875 [==============================] - 378s 201ms/step - loss: 0.3172 - accuracy: 0.9072
Epoch 2/20
1875/1875 [==============================] - 365s 195ms/step - loss: 0.0901 - accuracy: 0.9754
Epoch 3/20
1875/1875 [==============================] - 353s 188ms/step - loss: 0.0534 - accuracy: 0.9841
Epoch 4/20
1875/1875 [==============================] - 354s 189ms/step - loss: 0.0444 - accuracy: 0.9876
Epoch 5/20
1875/1875 [==============================] - 340s 182ms/step - loss: 0.0359 - accuracy: 0.9904
```

Hyperparameter Tuning

- Tuning hyperparameters can largely be guess-and-check
- Sklearn has built-in ‘guess-and-check’ processes to help you find the best parameters
    - GridSearchCV()
    - RandomSearchCV()

Real-World ML

After splitting your data into training and testing data with sklearn, a confusion matrix is a simple way to gauge a model, once you know how to read one. A confusion matrix for a binary classification problem looks like this:



n=165	Predicted: NO	Predicted: YES
Actual: NO	50	10
Actual: YES	5	100

Real-World ML

- True positives and true negatives are cases where our model is correct
- A false positive (top right) is when the predicted value is yes but the actual value is no
- A false negative (bottom-left) is the opposite

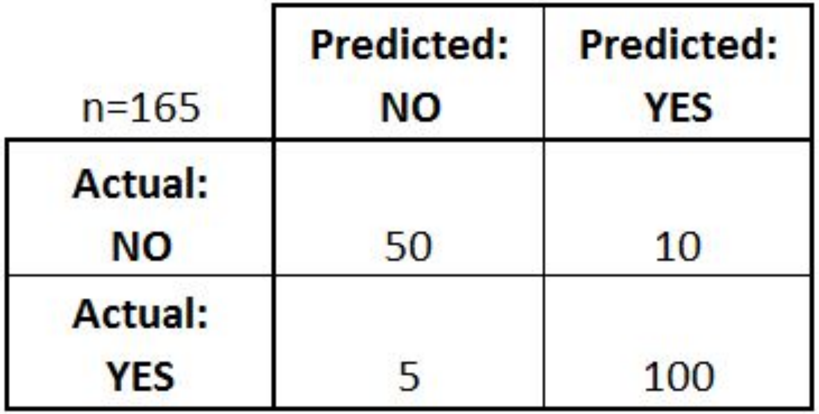
Precision and Recall

- Precision: What proportion of positive identifications was actually correct?

$$\text{Precision} = \frac{TP}{TP + FP}$$

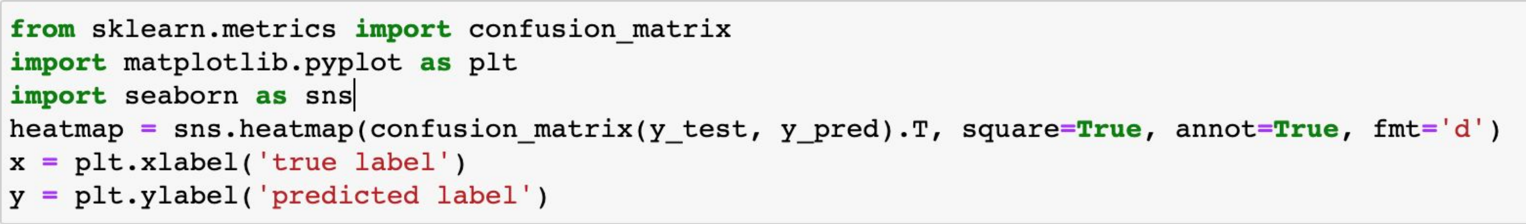
- Recall: What proportion of actual positives was identified correctly?

$$\text{Recall} = \frac{TP}{TP + FN}$$

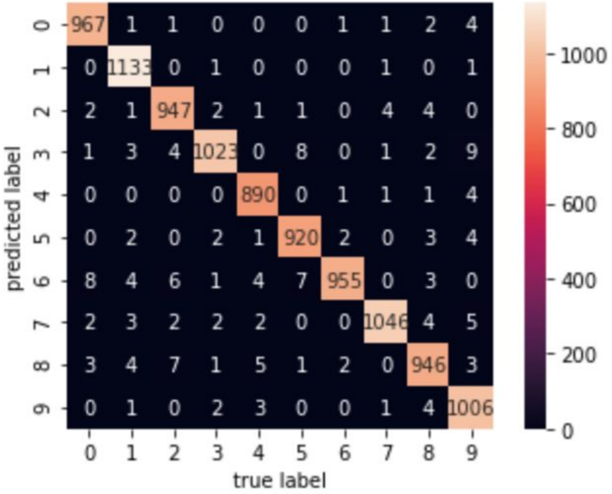


n=165	Predicted: NO	Predicted: YES
Actual: NO	50	10
Actual: YES	5	100

Confusion Matrix



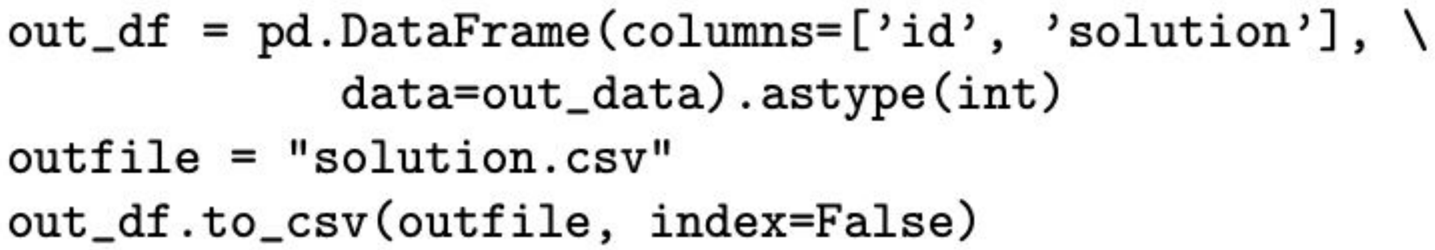
```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
heatmap = sns.heatmap(confusion_matrix(y_test, y_pred).T, square=True, annot=True, fmt='d')
x = plt.xlabel('true label')
y = plt.ylabel('predicted label')
```



	0	1	2	3	4	5	6	7	8	9
0	967	1	1	0	0	0	1	1	2	4
1	0	1133	0	1	0	0	0	1	0	1
2	2	1	947	2	1	1	0	4	4	0
3	1	3	4	1023	0	8	0	1	2	9
4	0	0	0	0	890	0	1	1	1	4
5	0	2	0	2	1	920	2	0	3	4
6	8	4	6	1	4	7	955	0	3	0
7	2	3	2	2	2	0	0	1046	4	5
8	3	4	7	1	5	1	2	0	946	3
9	0	1	0	2	3	0	0	1	4	1006

Results

After using np.reshape() and np.concatenate() to merge our y_pred array with a corresponding id row, writing to a .csv file is as simple as creating an output DataFrame and saving it:



```
out_df = pd.DataFrame(columns=[’id’, ’solution’], \
            data=out_data).astype(int)
outfile = "solution.csv"
out_df.to_csv(outfile, index=False)
```

Credits

Portions of this lecture have been adapted from Kevin Fu’s October 2019 “Real World Machine Learning” lecture