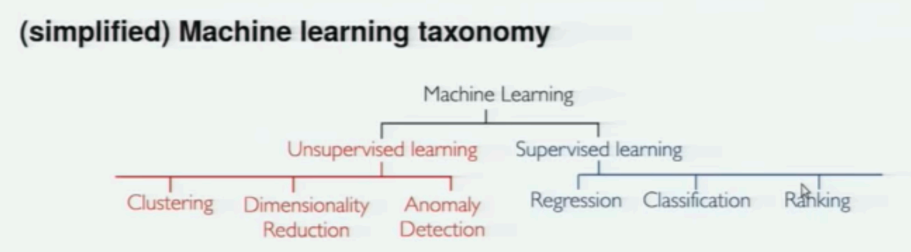
Report of Machine Learning

Piruz Alemi
March 28th, 2020

This paper and its accompanying model relies on Boruki (2010) paper, and builds on its findings:
https://apps.dtic.mil/dtic/tr/fulltext/u2/a559102.pdf

The following approaches were applied:



Background

Over a period of nine years in deep space, the NASA Kepler space telescope has been out on a planet-hunting mission to discover hidden planets outside of our solar system.

To help process this data, I created machine learning models capable of classifying candidate exoplanets from the raw dataset.

In this work I needed to:

1. Preprocess the raw data
2. Tune the models
3. Compare two or more models

Actions I took:

Preprocessed the Data

- Preprocessed the dataset prior to fitting the model.
- Performed feature selections and removed unnecessary features.
- Used `MinMaxScaler` to scale the numerical data.
- Separated the data into training and testing data.

Tuned the Model Parameters

- Used `GridSearch` to tune model parameters.
- Tuned and compared at least two different classifiers.

Reporting

- Created a README that reports a comparison of each model's performance
- Provided a summary about my findings and any assumptions I made based on my model (is my model good enough to predict new exoplanets? Why or why not? What would make this model be better at predicting new exoplanets?).

Resources I used:

- Exoplanet Data Source
- Scikit-Learn Tutorial Part 1
- Scikit-Learn Tutorial Part 2
- Grid Search

For incorrect indices, see: position 2:24 of the following tape – part1:
https://www.youtube.com/watch?v=4PXAztQtoTg

Supervised Learning:

a. In the Supervised Leaning process we start with gathering data – A collection of pairs (input,output).
b. Each input feature vector will have a lable that contributes to the output, like "Spam/Not Spam"
c. We need to transform these labels, so some of our algorithms can process a 0/1 or a -1/+1 instead of strings.
d. In case of SVM (Support Vector Machine), we construct a data set, consisting only of numeric values
e. SVM sees every feature vector as a point in a high-dimensional space.
f. The algorithm puts all feature vectors on an imaginary say n-dimensions plots and draws an (n-1) line (a hyperplane) that separates the 0/1 or the -1/+1 targets (Spam and not spam)
g. The boundary separating the examples of different classes is called decision boundary
h. The predicted label for some input feature vector x is given like this:
    a. $Y = \text{sign}(wx - b)$
i. Where sign is a mathematical operator that takes any value as input and returns +1 if the input is positive or -1 if the input is a negative number.
j. The goal of SVM algorithm – SVM in this case – is to leverage the data set and find the optimal values w* and b* for parameters w & b. Once the learning algorithm identifies these optimal values, the model f(x) is then defined as:

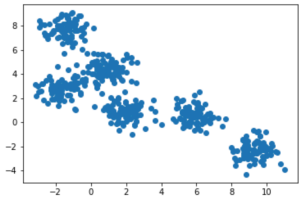
$$F(x) \text{ sign}(W^*x - b)$$

Therefor to predict whether an e-mail message is say spam or not spam using SVM model, we have to take a text message, convert it into a feature vector, then multiply this vector by w*, substract b* and take the sign of the result. This will give us the prediction (+1 means "spam", -1 means "not spam")

k. Now, how does the machine find w* and b*? It solves an optimization problem.
l. Recall that each example i=1,….,10,000 is given by a pair (x_i, y_i), where ….

1. Kmeans – This is a Clustering Classification model on the basis of means for each cluster (Blobs)

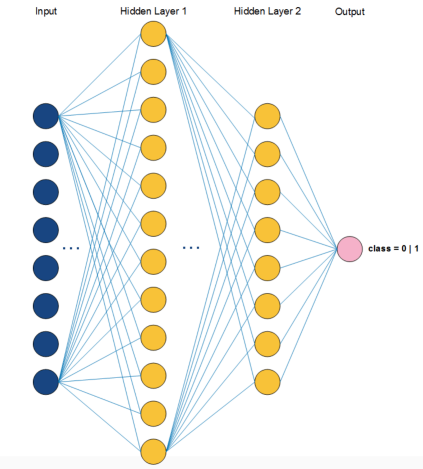
```
# Plot and show scatter
plt.scatter(X[:, 0], X[:, 1])
plt.show()
```



2. Deep Learning:. In this case we built a Deep Neural Network:
    a. In the First Layer we included the features, for the Training data
    b. Next I added the hidden layers

```
model.add(Dense(12, input_dim=8, init='uniform', activation='relu'))
```

It means 8 input parameters, with 12 neurons in the FIRST hidden layer.



    c. Deep Learning refers to training neural networks with more than two non-output layers.
    d. These models use a backpropagation for computing gradients on neural networks using the chain rule.
    e. We call them Deep, as we can go up to 100 layers deep. In practice with 2- layers usually we get the results.
    f. Layers that are neither input nor output are called hidden layers.
        i. For hidden layers I used activation = "relu" and for output layer, activation = "softmax"
    g. https://medium.com/@himanshuxd/activation-functions-sigmoid-relu-leaky-relu-and-softmax-basics-for-neural-networks-and-deep-8d9c70eed91e

    h. https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

3. Random Forrest Classifier.

    a. This model works in the following way:
    b. Given a training set, we create B random samples S_b (for each b = 1,2,…..,B)
    c. To sample S_b with do sampling with replacement
    d. We keep picking samples at random until the $|S_b| = N$
    e. After Training we have B decision trees. The prediction for a new sample x is obtained as the average pf B predictions
    f. Note:
        i. If one or a few features are very strong predictors for the target, these features will be selected to split examples in many trees. This would result in many correlated trees in our "forest"
        ii. Correlated predictors cannot help in improving the accuracy of prediction.
    g. In this model, the most important hyperparameters to tune are the number of trees, B, and the size of the random subset of the features to consider at each split
    h. By using multiple samples of the original data set, we reduce the **Variance** of the final model.
    i. Recall low variance means low overfitting. Overfitting happens when our model attempts to explain the small variations
    j. If we are unlucky with how our training data set was sampled, then it could contain some undesirable (but unavoidable) artifacts: noise, outliers and under-represented examples.

```
In [6]: from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(data, target, random_state=42)

In [7]: clf = tree.DecisionTreeClassifier()
        clf = clf.fit(X_train, y_train)
        clf.score(X_test, y_test)

Out[7]: 0.8478260869565217

In [10]: from sklearn.ensemble import RandomForestClassifier
         rf = RandomForestClassifier(n_estimators=20)
         rf = rf.fit(X_train, y_train)
         rf.score(X_test, y_test)

Out[10]: 0.8895881006864989

In [11]: sorted(zip(rf.feature_importances_, feature_names), reverse=True)

Out[11]: [(0.09448765886482223, 'koi_fpflag_co'),
          (0.08898260126884405, 'koi_fpflag_nt'),
          (0.07328931536249259, 'koi_fpflag_ss'),
          (0.05975025093752773, 'koi_model_snr'),
          (0.059464722458763275, 'koi_prad'),
          (0.041910465369439355, 'koi_duration_err1'),
          (0.04101272977036457, 'koi_prad_err2'),
          (0.04032884879837784, 'koi_fpflag_ec'),
          (0.03655235078795964, 'koi_duration_err2'),
          (0.03202198413615089, 'koi_steff_err2'),
```

Random Forest Classification report + Confusion Matrix

The precision captures how well our model for classification of the disposition was correct, running it on our test data.

```
In [31]: from sklearn.metrics import confusion_matrix, classification_report

In [34]: print(classification_report(y_test,rf_pred))

                   precision    recall  f1-score   support

        CANDIDATE       0.79      0.77      0.78       411
        CONFIRMED       0.82      0.81      0.82       484
    FALSE POSITIVE       0.97      0.99      0.98       853

         accuracy                           0.89      1748
        macro avg       0.86      0.86      0.86      1748
     weighted avg       0.89      0.89      0.89      1748


In [35]: print(confusion_matrix(y_test,rf_pred))

[[316  81  14]
 [ 82 394   8]
 [  4   3 846]]
```

The confusion matrix gives 316, 394, 846 good (same between trained and tests) predictions across Candidate, Confirmed and False Positives. See: https://en.wikipedia.org/wiki/F1_score
And: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

We take the model's Target to be classification on whether the planet, given a set of features has the probability of detecting water – that is a temperature range, where water can exist. The above results can be substantially improved, if we reduce the number of featured columns to simply

```
In [11]: sorted(zip(rf.feature_importances_, feature_n

Out[11]: [(0.09448765886482223, 'koi_fpflag_co'),
          (0.08898260126884405, 'koi_fpflag_nt'),
          (0.07328931536249259, 'koi_fpflag_ss'),
          (0.05975025093752773, 'koi_model_snr'),
          (0.059464722458763275, 'koi_prad'),
          (0.041910465369439355, 'koi_duration_err1'),
          (0.04101272977036457, 'koi_prad_err2'),
          (0.04032884879837784, 'koi_fpflag_ec'),
```

Recall, our preliminary analysis started with the relation of some of the Keppler's features, which includes the following for our model:

Or its disposition may be summed in koi_disposition as:

1. Confirmed
2. Candidate
3. False Positive

The target values are then validated against the original paper of Boruki (2010), where they derive the latter 3 classes. Boruki's model's Features compared each star relative/similar revolved around the following key variables:

1. Radius
2. Mass,
3. Temperature/Heat
4. Duration (Light)
5. Time (in light reaching earth)

Our first approach is to use a different set of features predicting same koi_dispositions. We used GridSearch with a set of hyper parameters, in search of a set of models we may use for our purpose. A description and application of various models to the above model thru Machine Learning follows:

Logistic Regression:

Kernels:

https://www.ijraset.com/fileserve.php?FID=3040

For setting parameters in my GridSearch:
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

Machine Learning Exoplanet Exploration

1. Created a new repository for this project called `machine-learning-challenge`.
2. Cloned the new repository to my computer.
3. Given each model, I choose their own Jupyter notebook, **did not use more than one model per notebook.**
4. Saved my best model to a file. This was the model used to test our accuracy.
5. Committed my Jupyter notebooks and model file and pushed them to GitHub.

Input data:

For a description of input columns, see: https://exoplanetarchive.ipac.caltech.edu/docs/API_kepcandidate_columns.html

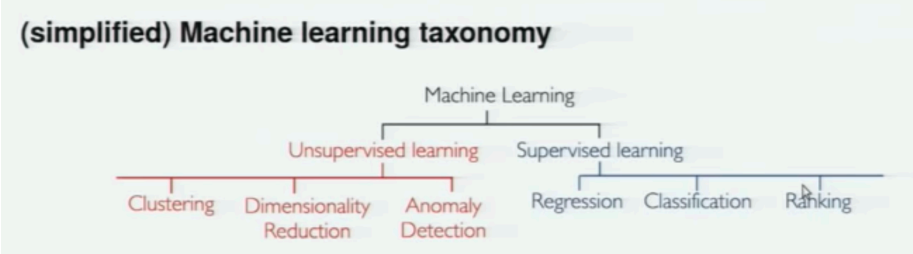
koi_disposition	koi_fpflag_nt	koi_fpflag_ss	koi_fpflag_co	koi_fpflag_ec	koi_period	koi_period_err1	koi_period_err2	koi_time0bk	koi_time0bk_err1	koi_time0bk_err2	koi_impact	koi_impact_err
CONFIRMED	0	0	0	0	54.4183827	0.0002479	-0.0002479	162.51384	0.00352	-0.00352	0.586	0.05
FALSE POSITIVE	0	1	0	0	19.89913995	1.49E-05	-1.49E-05	175.850252	0.000581	-0.000581	0.969	5.12
FALSE POSITIVE	0	1	0	0	1.736952453	2.63E-07	-2.63E-07	170.307565	0.000115	-0.000115	1.276	0.11
CONFIRMED	0	0	0	0	2.525591777	3.76E-06	-3.76E-06	171.59555	0.00113	-0.00113	0.701	0.23
CONFIRMED	0	0	0	0	4.13443512	1.05E-05	-1.05E-05	172.97937	0.0019	-0.0019	0.762	0.13
CONFIRMED	0	0	0	0	2.56658897	1.78E-05	-1.78E-05	179.55437	0.00461	-0.00461	0.755	0.21
CONFIRMED	0	0	0	0	16.06864674	1.09E-05	-1.09E-05	173.621937	0.000517	-0.000517	0.052	0.26
CONFIRMED	0	0	0	0	2.470613377	2.7E-08	-2.7E-08	122.763305	8.7E-06	-8.7E-06	0.818	0.00
CONFIRMED	0	1	0	0	2.204735417	4.3E-08	-4.3E-08	121.3585417	1.6E-05	-1.6E-05	0.224	0.15
CONFIRMED	0	0	0	0	3.522498429	1.98E-07	-1.98E-07	121.1194228	4.71E-05	-4.71E-05	0.631	0.00
CONFIRMED	0	0	0	0	3.709214104	6.54E-06	-6.54E-06	133.98318	0.00143	-0.00143	0.051	0.39
FALSE POSITIVE	0	1	0	0	11.52144606	1.98E-06	-1.98E-06	170.839688	0.000131	-0.000131	2.483	2.85

Some considerations on selection/usage of data columns:

1. http://www.ucolick.org/~bolte/AY4_00/week5/star_radii.html

Steps and Considerations

1. Started by cleaning the data, removing unnecessary columns, and scaling the data.
2. Not all variables are significant. I made sure to remove any insignificant variables.
3. Made sure my `sklearn` package is up to date.
4. Tried a simple model first, and then tune the model using `GridSearch`.



- First I tried a Fit & Predict Method of a linear and then RBF kernel, with a loss tradeoff C=1:
  - Sklearn .svm , with model = SVC(kernel = "rbf"
  - with model = logisticRegression(C=1,
  - with model = RandomForest
- Second On the GridSearch, I changed the Hyper Parameters to apply a Knn model
- Third I applied a Deep_Learning Model, on same train test data.

Some Technical considerations:

1. Reshaping does not make a copy, just a view
2. Slicing does not allocate new memory or making a copy, just a view
3. Fancy indexing makes a copy.

Supervised Learning Classification:

K Nearest Neighbours:

1. In K-neighbourhoods approach we have a fixed set of features and a fixed set of class outputs, that we set.

This was achieved, via Splitting & Scaling & Normalizing, Training, Testing + Scoring

Here our Target was the Category. And a predefined number of Categories and we try to predict the categories. There were no continuous variables.

1. Here the Knn Classifier is n=35 to specify the nearest neighbors
2. If we set n=1 and run a scatterplot on the Training data, we derive our decision boundaries.
3. With n =5, we get a disconnected decision boundary. With n=1 we have a continuous line, but all errors are also accounted for, which may not be generalizable to the test data, i.e. we have over fit the model.
4. When n=1, the model is doing a perfect prediction, but the Knn.score = 0.76
5. First we split the data into Train & Test, and then we break it into X-Trainsub and validation.
6. We also need to do a Cross-Validation.

```
X_trainsub, X_valid, y_trainsub, y_valid = train_test_split(
    X_train, y_train,
    test_size=0.5,
    random_state=1234,
    stratify=y_train)

for k in range(1, 20):
    knn = KNeighborsClassifier(n_neighbors=k)
    train_score = knn.fit(X_trainsub, y_trainsub).\
        score(X_trainsub, y_trainsub)
    valid_score = knn.score(X_valid, y_valid)
    print('k: %d, Train/Valid Acc: %.3f/%.3f' %
          (k, train_score, valid_score))
```

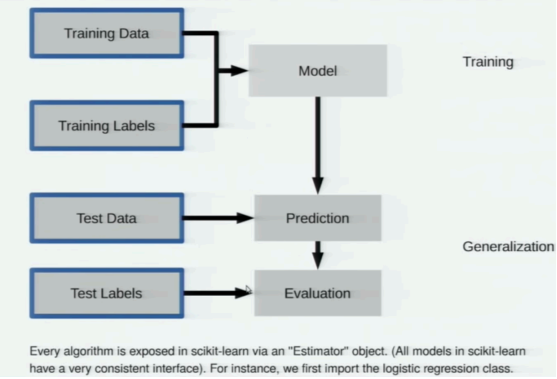
```
k: 1, Train/Valid Acc: 1.000/0.946
k: 2, Train/Valid Acc: 1.000/0.964
k: 3, Train/Valid Acc: 1.000/0.946
```

1. Here our goal is to find the best k.
    1. We Train the model
    2. We Validate the model by its score
    3. The sample size can affect and we can get multiple ks that are good, but would be unstable.
2. Once we found the optimum classifier, we Train the model on the optimum of k, say n_neighbours = 2.
3.

Logistic Regression (Parametric approach)

2. (2:34) position on Tape - In Regressions, we may use the Blob Data and self generate the data. A two Class Classification in 2-dimentions. (Blobs are like circles with same standard deviations, A Gaussian Distribution).

    - 2:37 – Sklearn Train_test _Split (test_size=0.25, random_state=1234, stratify = y)
    - The data is in Train an Test split and each data set has its X & y



Evaluation is to see if the predictions meets the actuals

1. In the case of Logistic Regression. We use the Shape function to ensure that the we have the #Samples + 2 features SoX_Train.shape = (75,2) and Y-Train.shape = (75,)

2. Classifier fit (X_Train, Y-Train), first the model . Fit returns the model itself. From Fit we can chain command and say Fit.Predict.

3. Prediction = classifier.predict(X-Test). Note we know nothing about Y_Test.

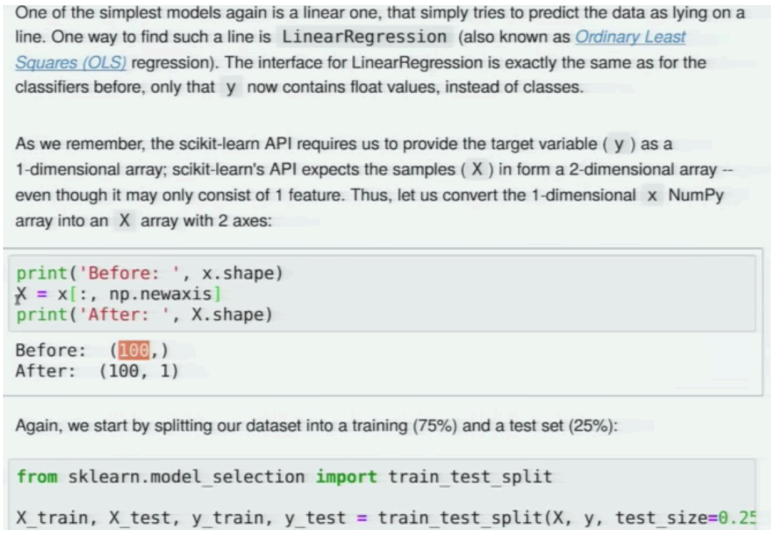
4. Print(Prediction)
   Print(Test)
   Will give us the match and miss matches as in [1,0,0,1,1,0,0,1]
   [1,1,0,1,1,0,1,1]

5. Prediction == y-test, will return a Boolean mask of True and false, whose count will give the accuracy of the model.
6. The double ?? shows us the underhood code, say for plots.
7. Classifier. Coef_ the underscore highlights that this is a variable derived by the model.
8. Clssifier.intercept_

The Regression Model:

We could pick just one feature from X to fit our model, but what we really want it to find a line that best fits the data in n-dimensional space. To achieve this, Linear Regression can be solved using the analytical approach called Ordinary Least Squares or a computational approach Gradient Descent for estimating the parameters. Note that there are tradeoffs between using either approach. The Linear Regression model in Sklearn uses the Ordinary Least Squares method.

But a review of data's features in relation to its dependent variable reflects that the data was not linear.



One of the simplest models again is a linear one, that simply tries to predict the data as lying on a line. One way to find such a line is `LinearRegression` (also known as *Ordinary Least Squares (OLS)* regression). The interface for LinearRegression is exactly the same as for the classifiers before, only that `y` now contains float values, instead of classes.

As we remember, the scikit-learn API requires us to provide the target variable (`y`) as a 1-dimensional array; scikit-learn's API expects the samples (`X`) in form a 2-dimensional array -- even though it may only consist of 1 feature. Thus, let us convert the 1-dimensional `x` NumPy array into an `X` array with 2 axes:

```
print('Before: ', x.shape)
X = x[:, np.newaxis]
print('After: ', X.shape)
```

```
Before:  (100,)
After:  (100, 1)
```

Again, we start by splitting our dataset into a training (75%) and a test set (25%):

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25
```

Note the changing of the axis in the Numpy from one-dimension to two-Dimension.

Note also that stratification in this case does not apply, as we have a continuous data.

Classification Model:

https://exoplanetarchive.ipac.caltech.edu/docs/API_kepcandidate_columns.html

Decision_Tree Model:

Submission

- Created a Jupyter Notebook for each model and host the notebooks on GitHub.
- Created a file for my best model(s) and pushed it to GitHub
- Included a README.md file that summarized my assumptions and findings.
- Submitted the link to my GitHub project to Bootcamp Spot.