T.C.
iSTANBUL KÜLTÜR UNIVERSITY
FACULTY OF ENGINEERING
DEPARTMENT OF
ELECTRICAL & ELECTRONICS ENGINEERING

INTERNSHIP REPORT

Mert ERTÜRK
Student Number: 1700002485
Period of the Internship:
2021.08.13 - 2021.09.10

Course:
EE7021 Industrial Training I☒

iSTANBUL KÜLTÜR ÜNİVERSİTESİ

Ataköy 7-8-9-10, E5 Karayolu Üzeri Ataköy Yerleşkesi,
34158 Bakırköy

Authorized Personel Information:
Instructor: Basri Erdoğan
b.erdogan@iku.edu.tr

Staj Faaliyet Özeti

Bölüm ve İş Tanımı	Başlangıç	Bitiş		İş Günü
Python ve Makine Öğrenmesi	2021.08.13	2021.08.27		11
Makine Öğrenmesi Projeleri	2021.08.31	2021.09.02		3
Pandas ve Matplotlib ile Veri Analizi ve Veri Görselleştirmesi	2021.09.03	2021.09.06		2
Exploratory Data Analysis with Python	2021.09.07	2021.09.10		4

Yetkili personel Adı Soyadı Ünvanı	 Basri Erdoğan Elektronik Müh.
İmza	

Kurum Hakkında

İstanbul Kültür Üniversitesi, 9 Temmuz 1997'de Bakırköy, İstanbul'da kurulan vakıf üniversitesidir. Üniversite; Ataköy, Şirinevler, İncirli ve Basın Ekspres Yerleşkesi olarak dört kampüste faaliyet göstermektedir.

İstanbul Kültür Üniversitesi akademik ve bilimsel faaliyetlerini Eğitim Fakültesi, Fen-Edebiyat Fakültesi, Sanat ve Tasarım Fakültesi, Hukuk Fakültesi, İktisadi ve İdari Bilimler Fakültesi, Mimarlık Fakültesi, Mühendislik Fakültesi, Sağlık Bilimleri Fakültesi, Meslek Yüksekokulu ve Adalet Meslek Yüksekokulunda toplam 57 bölüm-programda eğitim sürdürmektedir.

İstanbul Kültür Üniversitesi Elektrik-Elektronik Mühendisliği Bölümü Vizyonu: Uygulamalı ve sanayiye yönelik ders içerikleri ve projeler ile vakıf üniversitelerinin Elektrik-Elektronik Mühendisliği Bölümleri arasında öncelikle tercih edilen bölümlerden biri olmaktır.

İstanbul Kültür Üniversitesi Elektrik-Elektronik Mühendisliği Bölümü Misyonu: Etik değerler öncelikli olmak üzere çalıştığı alanı sorgulayabilen, eleştirebilen ve özgün çözümler sunabilen, takım çalışması yapabilen Elektrik-Elektronik Mühendisleri yetiştirmek, aynı zamanda üretim ve sanayi iş birliği ağırlıklı projelerle ülke ekonomisine katkıda bulunmak, uluslararası alanda tanınır hale gelmektir.

Yaz stajımı, kariyerimde önemli olacağını düşündüğüm ve öğrenmeyi içten istediğim bir alanda araştırma yaparak gerçekleştirmeme imkân sağladıkları için bölüm başkanımız Doç. Dr. Esra SAATÇİ ve gözetiminde staj yaptığım Arş. Gör. Basri Erdoğan'a teşekkür ediyorum.

Staj Faaliyetleri

Bölüm: Elektrik – Elektronik Mühendisliği	**Tarihler:** 2021.08.13 - 2021.08.27
İş Tanımı: Python ve Makine Öğrenmesi	**Süre: 11 gün**

1) Makine Öğrenmesi Nedir?

Makine öğrenmesinin olayı esasında "**veri**" den "**bilgi**" çıkartmaktır. İstatistik, yapay zekâ ve bilgisayar biliminin kesişimi olarak görülebilir ve **istatistiksel öğrenme** ve **tahmin analitiği** olarak da bilinir.

İki tür makine öğrenim algoritma tipi vardır. **Supervised** ve **Unsupervised** öğrenim.
NOT: ([Gözetimli – Gözetimsiz] raporun devamında bu tip terimleri İngilizce kullanacağım.)

Supervised learningde sisteme öğrenilmek istenen kavram ile ilgili etiketlenmiş veriler input (boyut, renk vs.) ve bu veriler için istenen çıktı/output (muz, portakal vs.) verilir. Bu bilgiler kullanılarak girdi ve çıktı arasında ilişki kurulur ve bu ilişki kullanılarak hiç görülmemiş yeni veri üzerine tahmin yapılabilir.

Unsupervised Learning basitçe sisteme sokulan verilerin arasındaki pattern/ilişkiyi keşfedilip, (örnek olarak, en yakın komşu sayısı ve örnekler arası uzaklık) sistemin içerisinde kümeleme yapmak amaçlanır. (Üstteki örnek setimizi düşünürsek sonuç **"verimizde turuncu ve yuvarlak /sarı ve kavisli nesne şeklinde iki ayrı küme var"** a benzeyecektir.)

Supervised örnekleri,

- Elimizde Sağlıklı ve kanserli görüntü verisi varsa hiç görülmemiş görüntü üzerinde tahminde bulunmak.
- El yazısı görselindeki rakamın hangi rakam olduğunun tahminini yapmak.
- Ev özellikleri (konum, metrekare, oda sayısı) üzerinden fiyat tahmininde bulunmak.

Unsupervised Örnekleri

- Benzer tercihleri yapan müşteri/kullanıcıları gruplandırmak.
- Website erişiminde anormal davranış tespit etmek.
- Film özetlerinden içerik olarak birbirine en çok benzeyenleri keşfetmek.

Bu iki öğrenme tipi için de veriyi bilgisayara **düzgün bir şekilde sunmak** çok önemlidir. Veriyi excel tablosu olarak düşünürsek **her satır bir örnektir** ve **her sütun bir "feature"dır**. (Renk, boy) ve doğal olarak verimizde **soyadların olduğu bir sütun** varsa bunu sisteme dahil edip **cinsiyet tahmini yapma** konusunda sadece karmaşa yaratacaktır.

İşe koyulmadan önce düşünmemiz gerekenler.
• Cevaplamak istediğim soru(lar) nedir? Cevap verecek veriyi topladım mı?

• Sisteme bu problemi en iyi nasıl ifade ederim?

• Problemi ortaya koyacak ve çözecek kadar çözecek kadar veriyi topladım mı?

• Ortaya çıkardığım "feature"lar neler ve doğru tahmini sağlayabilecekler mi?

• Uygulamadaki başarımı nasıl ölçebilirim?

• Bu makine öğrenmesi çözümü araştırmama ya da ürünüme nasıl bir etkisi olacak?

Neden Python?

Kullanımı basit bir yüksek seviye programlama dili. Veri yükleme, görselleştirme, istatistik ve veri işleme için topluluğun oluşturduğu çok güçlü kütüphaneler var. Ayrıca Python bir programlama dili olduğu için ürünümüzü başka sistemlere kolayca entegre etmek mümkün.

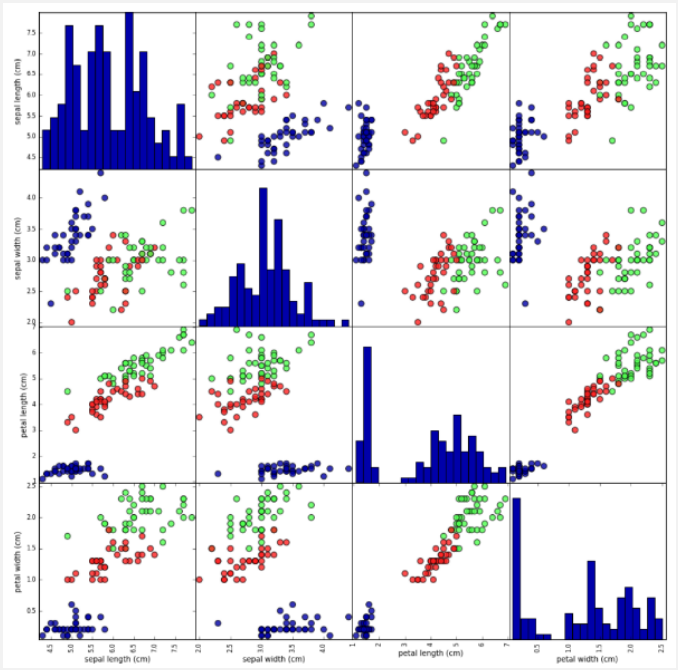
2) Genel konseptler ve basit bir Örnek

İki tip Makine öğrenmesi uygulaması vardır. **Classification ve Regression**.

- Elimizde müzik verileri varsa ve bunları Rock mı yoksa Pop mu gibi sınırlı sayıda sınıfa ayırmak istersek bu "**classification"**dır. Regresyona göre tanımı daha basittir.

- **Regression** ise esasında girdi ve çıktı arasında bağımlı ve bağımsız değişken ilişkisi kurup. Girdi üzerinde bağımsız değişkeni değiştirdiğimizde sonucun ne olduğunu tahmin etmektir. Basit bir örnek vermek gerekirse, boy kilo endeksini düşünelim. **Kısa birinin uzun birine göre daha hafif olmasını bekleriz misal ortalama 160 boyunda insanlar ortalama 65 kiloysa ve 190 boyunda insanlar ortalama 85 kiloysa. Bu bilgiyi kullanıp farklı değerleri tahmin edebiliriz**. (tabii ki daha büyük bir veri setiyle yani başka "feature"ların da olduğu için ve basit bir çizgiden ziyade daha karışık ve çok boyutlu düzlem üzerinde olan bir eğriyle karşılaşırız.) Başka bir örnek yaş, eğitim durumu, şehir gibi "feature"lar ile yapılan gelir tahmini karmaşık bir eğri olacaktır ve tahmin %100 doğru olmasa bile çok yakın ve geçerli olacaktır.

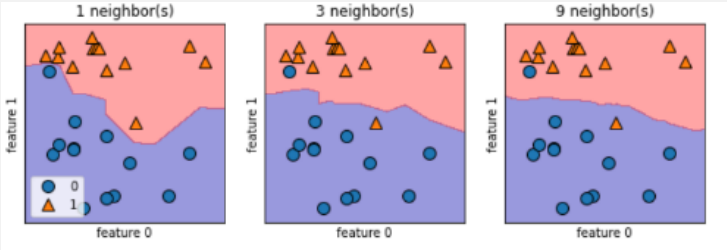
Basit bir uygulama örneği, elimizde üç farklı tip çiçek ve özellikleri olan bir veri var ve tahminde bulunacağız. Bu bir **classification** örneğidir.



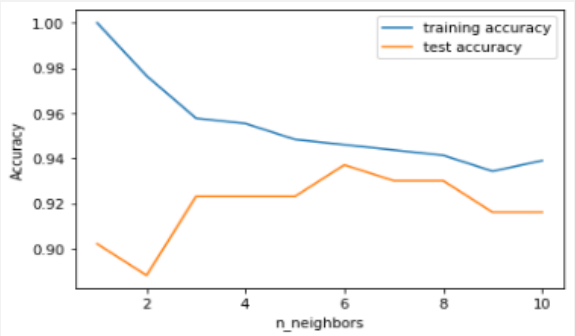
(Çaprazda kalan grafikler her özelliğin dağılım aralığının histogramıdır.)

Soldaki görselde üç farklı çiçek türünün özellikleri "sepal" ve "petal" uzunlukları ve genişlikleri olarak belirlenmiş. Görselde sadece iki boyutlu düzlem üzerinde bu dört özelliğin her tabloda ayrı iki tanesini kullanarak çizilmiş halde. Bu görselle beraber anlatılmak istenen şey ise,
Örnek olarak sondan ikinci **dağılım** tablosunu ele alalım. Üç ayrı tip çiçeği sınıflandırmak için sadece "petal" uzunluğu ve kalınlığını kullanmak bile gayet yeterli gözüküyor. Bu veriyi artık makine öğrenmesi algoritmalarından birine yerleştirebiliriz.

K-nn algoritması (en yakın komşular)
Her sınıf için sınırlar oraya en yakın bizim tarafımızdan belirlenmiş komşu sayısı (k) ile belirlenir. Farklı komşu sayılarıyla sistemi yerleştirip en iyi sonucu bulmaya çalışabiliriz.



Veri setimizi train %80 ve test %20 olarak ayırıp. Sistemin hiç görmediği test örneklerimizle en iyi başarıyı kaç komşuyla aldığımızı keşfederiz. Bu işe **parameter tuning** denir.

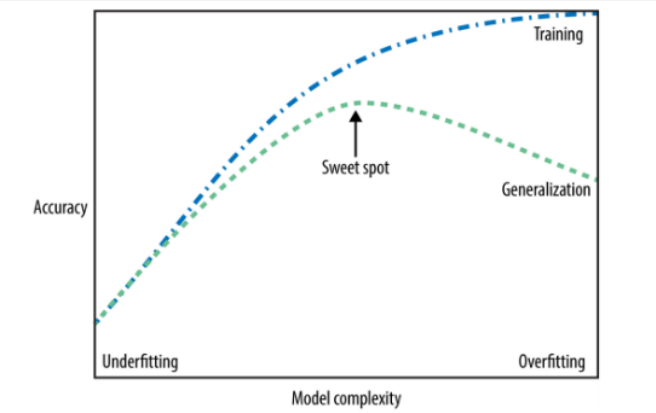


2.1) Generalization, Underfitting, Overfitting

Eğer modelimiz hiç görülmemiş veri için başarılı tahminlerde bulunuyorsa, train ettiğimiz veriden test ettiğimiz veriye "**generalize**" edebildiğini söyleyebiliriz.
Doğal olarak iyi generalizasyon yapan bi model isteriz, **fakat** eğer train ve test verimiz çok fazla ortak özellik taşıyorsa eğitim setini aşırı öğrenip, çok karmaşık modeller oluşturup başka tür bir soruna sebep olabiliriz.

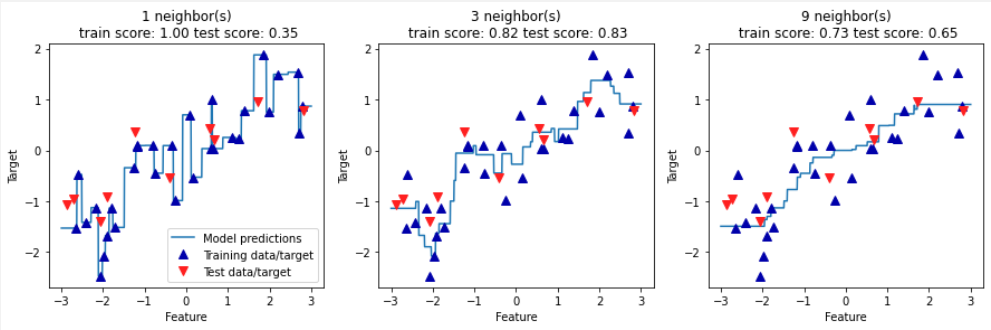
Bu duruma örnek olarak az sayıda örneğimiz olan bir veri düşünelim. Bu durumda kendimizi "**Eğer müşterimiz 45 yaşından büyük, üçten az çocuklu ve evli ise tekne satın almak istiyordur**." Gibi bir genelleme yapıp bu cümlenin %100 isabetli olduğu ama gene de gerçekliği yansıtmadığı bir duruma düşebiliriz.
Bu soruna "**overfitting**" denir ve elimizdeki bilgiyi fazla öğrenip gereksiz karmaşık bir model kurduğumuz anlamına gelir. Böyle küçük bir veri seti ile varılan çıkarım "**50 yaşından yaşlılar tekne satın almak ister**." olsaydı daha doğru olurdu.

Karşılaşılan başka bir sorun ise "**underfitting**"dir. Bu durum ise elimizdeki veriye çok basit bir model yerleştirmek olur. Örnek "**Evi olan herkes tekne almak ister.**" Bu şekilde de model oluşturunca eğitim setimizdeki tüm özelliklerden yeterince faydalanmamış oluruz.



Mesele burada hem çok fazla bireysel noktaya odaklanmayıp hem de eğitim verimizden olabildiğince fazla faydalanmaktır ve bu da "**sweet spot**" bulma işidir. Ne kadar fazla verimiz varsa "overfiting" yapmadan daha karmaşık ve daha iyi genelleyebilen modeller yaratabiliriz.

Bir önceki sayfada K-nn algoritmasının classification örneğini vermiştik. **Çoğu algoritmanın hem classification hem de regression varyantları vardır** ve k-nn algoritmasının regresyonu aşağıdaki görsele benzer. Tahmin için varsayılan en yakın komşuların ortalama değeri kullanılır.



2.2) Model başarımı Ölçümü

Classification için başarım ölçümü gayet basittir test tahmini doğruysa 1 yanlışsa 0 değeri alır ve hepsinin ortalaması başarımdır.

Regression'da ise **R^2 skoru** hesaplanır bu sayı test örneklerin her birinin tahminin ne kadar güvenilir olduğunu belirten 0 ile 1 arası sayıdır, bu sayıların ortalaması başarımdır.

3) Kitapta Örnekleri bulunan Makine Öğrenmesi Modelleri

Bahsi geçen tüm modellerin kısaca özellikleri ve kullanımları. (İsim aşinalığı için modellerden kısaca bahsedip açıklamalarına ve uygulamalarına sonrasında başlayacağım bu sayfa ilk okuyuşta pek bir şey ifade etmeyecektir.)

Nearest neighbors : Küçük veri setleri ile başlangıç için iyi anlatımı kolay.

Linear models : İlk başta denenmesi gereken algoritma, çok büyük veri setleri için iyi, multi-dimensional veri setleri için iyi.

Naive Bayes : Sadece classification da kullanılabilir, lineer modellerden bile hızlı, büyük ve multi-dimensional veri setleri için genelde lineer modellerden daha az isabetli.

Decision trees: Çok hızlı, verinin ölçeklenmesine gerek yok, görselleştirilebilir ve kolayca anlatılabilir.

Random forests : Tek bir tane karar ağacından her zaman daha iyi çalışır, güçlüdür. Veriyi ölçeklemeye gerek yoktur, multi-dimensional veriye uygun değildir.

Gradient boosted decision trees : Genelde random forest dan daha isabetlidir, eğitimi daha uzun sürse de tahmin sonucu daha hızlı çıkar ve hafızada daha az yer kaplar, random forest'a göre daha fazla parameter tuning e ihtiyacı vardır.

Support vector machines : Orta ölçekli ve feature'ları benzer olan veri setleri için uygundur, ölçeklemeye ihtiyacı vardır, parametre değişimine karşı hassastır.

Neural networks : Özellikle büyük veri setleri içi karmaşık modeller oluşturabilir. Verinin ölçeklendirilmesine ve parametrelere karşı hassastır, büyük modelleri eğitmek uzun zaman alabilir.

Yeni bir veri setiyle çalışmaya başlarken lineer veya Naive bayes gibi **basit modellerle veriyi iyice anladığımızdan emin olduktan sonra daha karmaşık modeller kurabilecek** random forest, SVM ya da neural networkler gibi **modellere geçiş yapılmalı.**

4) Supervised Learning Uygulaması

Bi veri setini pandas ve matplotlib gibi kütüphanelerle inceledikten sonra artık scikit-learn kullanarak tahmin mekanizması oluşturabiliriz. Bu örnekde önceden temizlenmiş bir veri setiyle basit bir makine öğrenme algoritması eğitiyoruz.

Not_1: Gerçek uygulamada veri nadiren elimize bu kadar temiz ulaşacağı için "pandas" ve "numpy" gibi kütüphanelere iyi hâkim olmamız gerekiyor.
Not_2: Aşağıdaki kod olabildiğince basit bir koddur ve tüm uygulamalar genel hat olarak bu aşamalardan geçer. Bu kodun aşamalarını iyi sindirmek gerçekten faydalı olacaktır.
Not_3: Kodu kopyalayıp bir text editörüne yapıştırmak daha anlaşılır kılar.

4.1)K-nn Classifier

```
# import the libraries

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd


# import the data

from sklearn.datasets import load_iris
iris_dataset = load_iris()
```

```
# split the data in input/output for training and test

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
                                iris_dataset['data'], iris_dataset['target'], random_state=0)

# import the predictor algorithm and fit data to the classifier

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)


# Predict the X_test and compare it with the actual labels and get the score

y_pred = knn.predict(X_test)
print("Test set predictions:\n {}".format(y_pred))
print("Test set score: {:.2f}".format(np.mean(y_pred == y_test)))


# make prediction from any unseen data and assign it the label

X_new = np.array([[5, 2.9, 1, 0.2]])

prediction = knn.predict(X_new)
print("Prediction: {}".format(prediction))
print("Predicted target name: {}".format(iris_dataset['target_names'][prediction]))
```

Not_4: Bu data setinde üç farklı iris çiçeği ve bu çiçeklerin özellikleri var.
Not_5: Eğer aynı veriye başka bir classification modeli yerleştirmek isteseydik değiştireceğimiz tek yer kırmızıyla yazılı olan kısımdaki classifier tipi ve onun parametreleri olacaktı. Sonraki örnekte aynı veri iki ayrı modele öğretilip farkları karşılaştırılıyor.

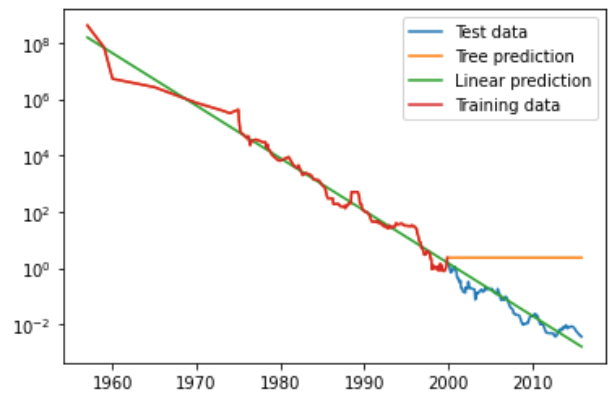
4.2) Linear Regression ve Decision Tree Regressor Karşılaştırması (Fiyat tahmini)

Not_1: Bu örnek regresyonların nasıl çalıştığını basitçe özetleyebilir.

Not_2: Aşağıdaki kodda iki ayrı regresyon modelinin karşılaştırması yapılıyor Lineer regresyon basit de olsa gelecek hakkında tahmin yapmak da zorluk çekmese de Karar ağacı if-else şeklinde çalıştığı için tahmin olarak en yakın değeri veriyor. Oysa ara değerlerden birini tahmin etmemiz gerekseydi lineer regresyondan çok daha başarılı bir tahminde bulunurdu.

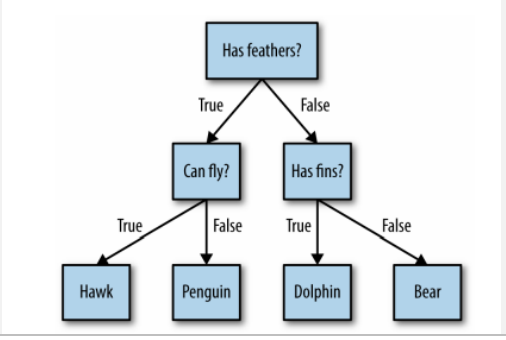
```
# import the libraries

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import mglearn
from sklearn.model_selection import train_test_split


#import the data

import os
ram_prices = pd.read_csv(os.path.join(mglearn.datasets.DATA_PATH,
 "ram_price.csv"))
```

```
#import the models for regression

from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor


# filter the data by date (data before year 2000 will be used to train and data after year 2000 will be used to
# test the success in prediction)


data_train = ram_prices[ram_prices.date < 2000]
data_test = ram_prices[ram_prices.date >= 2000]


# predict prices based on date
X_train = np.array(data_train.date)[:, np.newaxis]

# we use a log-transform to get a simpler relationship of data to target

y_train = np.log(data_train.price)
tree = DecisionTreeRegressor().fit(X_train, y_train)
linear_reg = LinearRegression().fit(X_train, y_train)

# predict on all data
X_all = np.array(ram_prices.date)[:, np.newaxis]
pred_tree = tree.predict(X_all)
pred_lr = linear_reg.predict(X_all)
# undo log-transform

price_tree = np.exp(pred_tree)
price_lr = np.exp(pred_lr)

#plot the comparison of the results

plt.semilogy(data_test.date, data_test.price, label="Test data")
plt.semilogy(ram_prices.date, price_tree, label="Tree prediction")
plt.semilogy(ram_prices.date, price_lr, label="Linear prediction")
plt.semilogy(data_train.date, data_train.price, label="Training data")
plt.legend()
```

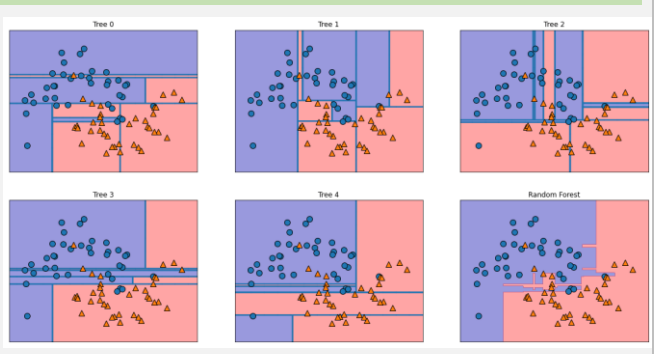


4.3) Random Forest Classifier

```
#import the libraries

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import mglearn
from sklearn.model_selection import train_test_split

#import the  classifier and the data

from sklearn.ensemble import RandomForestClassifier
```



```
from sklearn.datasets import make_moons

# split the data and fit the model onto it with the optional parameters

X, y = make_moons(n_samples=100, noise=0.25, random_state=3)
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,
                                                    random_state=42)
forest = RandomForestClassifier(n_estimators=5, random_state=2)
forest.fit(X_train, y_train)

# visualization of our randomforest and its estimator trees
fig, axes = plt.subplots(2, 3, figsize=(20, 10))
for i, (ax, tree) in enumerate(zip(axes.ravel(), forest.estimators_)):
    ax.set_title("Tree {}".format(i))
    mglearn.plots.plot_tree_partition(X_train, y_train, tree, ax=ax)
mglearn.plots.plot_2d_separator(forest, X_train, fill=True, ax=axes[-1, -1],
                                alpha=.4)
axes[-1, -1].set_title("Random Forest")
mglearn.discrete_scatter(X_train[:, 0], X_train[:, 1], y_train)
```

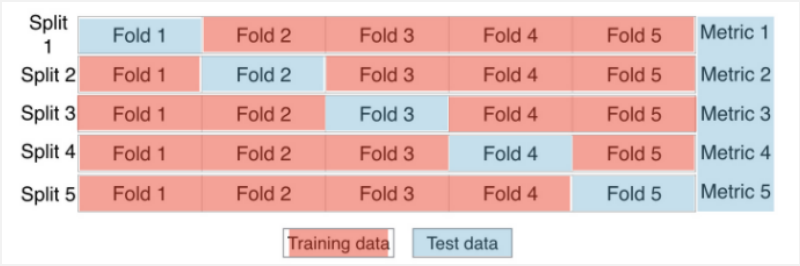


Not_1: Sınıflandırma konusunda random forest her zaman tek bir ağaçtan daha başarılı sonuç verir. (Sağ alt tablo)

Not_2: Son grafiği gerçek bir uygulamadan ziyade konsepte aşinalık yaratması için rapora ekledim.

5) Cross Validation, Feature Selection

5.1)Cross-validation (k-fold CV)



Önceki örneklerde elimizdeki veriyi %80 train ve %20 test seti olarak ikiye ayırmıştık ve sonuçta elimizde eğitimde kullanamadığımız %20'lik bir veri kalıyordu.

Cross validation'da ise elimizdeki veriyi istediğimiz kadar parçaya bölüp, parça sayısı kadar veriyi aynı algoritmaya sokuyoruz. Hesaplanan her metric için r^2 skorunu alttaki kodla erişebiliriz. Ortalamaları başarımdır ve train_test_splitten genelde daha başarılı olur.

```
from sklearn.model _ selection import cross _ val _ score
from sklearn.linear _ model import LinearRegression
reg = LinearRegression()
cv _ results = cross _ val _ score(reg, X, y, cv=5) print(cv _ results)
```

5.2) Lasso Regression for Feature Selection

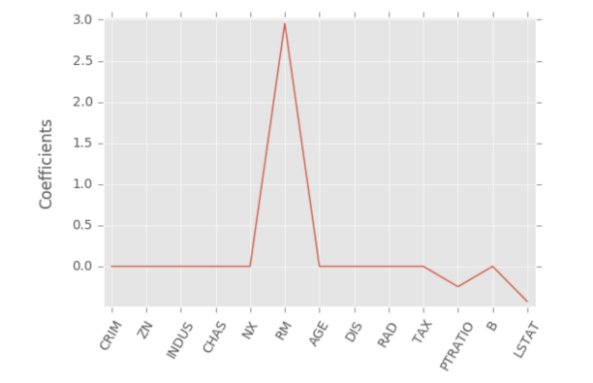
```
from sklearn.linear _ model import Lasso
X _ train, X _ test, y_ train, y_ test = train _ test _ split(X, y, test _ size = 0.3, random _ state=42)
lasso = Lasso(alpha=0.1, normalize=True)
lasso.fit(X _ train, y_ train)
lasso _pred = lasso.predict(X _ test)
lasso.score(X _ test, y_ test)
```

Lasso verisetimizdeki önemli özelliklerin neler olduğunu keşfetmemizi sağlayabilir. Çünkü tahmin sistemini kurarken her özellik için bir katsayı belirler ve eğer 0 dan farklı bir katsayı belirlediyse o özellik tahmin mekanizmasında etkilidir. Aşağıdaki örnekte Lasso Regressiona öğretilmiş "Boston House Pricing" datasetindeki özelliklerin katsayılarını grafik olarak yazdırıyoruz

```
names = boston.drop('MEDV' , axis=1).columns
lasso = Lasso(alpha=0.1)
lasso_ coef = lasso.fit(X, y).coef _
 _ = plt.plot(range(len(names)), lasso_ coef)
 _ = plt.xticks(range(len(names)), names, rotation=60)
_ = plt.ylabel('Coefficients') plt.show()
```

Bu üç feature'ın bizim için istatistiksel önem taşıdığını görürüz.

```
RM > average number of rooms per dwelling
PTRATIO >  pupil-teacher ratio by town
LSTAT > % lower status of the population
```



6) Confusion Matrix

Elimizde 100 tane e-mail örneği olduğunu düşünelim ve bu maillerden sadece bir tanesi "spam" mail olsun. Eğer elimizdeki sistem bu maillerin hepsini "spam değildir" olarak tahmin ederse "%99" isabetli sonuca ulaşır. Ama bir de şu açıdan bakarsak "spam" mail tahmini konusunda rezalet bir performans sergilemiştir.
Bu tarz dengesiz veri setleriyle çalışırken sıradan bir "accuracy test" yapmak doğru olmaz bu durumda Confusion matrix kullanmamız gerekir.

- Accuracy:

$$\frac{tp + tn}{tp + tn + fp + fn}$$

	Predicted: Spam Email	Predicted: Real Email
Actual: Spam Email	True Positive	False Negative
Actual: Real Email	False Positive	True Negative

- Precision $\frac{tp}{tp+fp}$
- Recall $\frac{tp}{tp+fn}$
- F1score: $2 \cdot \frac{precision*recall}{precision+recall}$
- High precision: Not many real emails predicted as spam
- High recall: Predicted most spam emails correctly

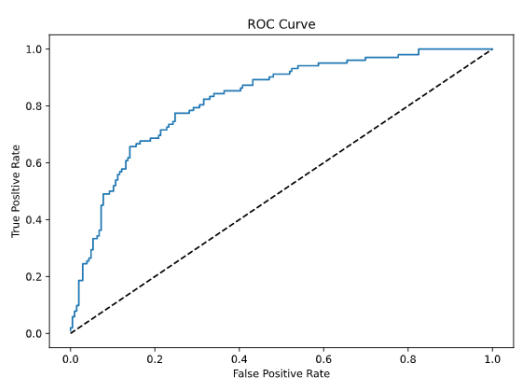
```
from sklearn.metrics import classification _ report
from sklearn.metrics import confusion _ matrix

knn = KNeighborsClassifier(n _ neighbors=8)
X _ train, X _ test, y_ train, y_ test = train _ test _ split(X, y, test _ size=0.4, random _ state=42)
knn.fit(X _ train, y_ train)
y_pred = knn.predict(X _ test)

print(confusion _ matrix(y_ test, y_pred))
```

```
[[52  7]
 [ 3 112]]
```

```
print(classification _ report(y_ test, y_pred))
```

```
             precision    recall  f1-score   support

          0       0.95      0.88      0.91        59
          1       0.94      0.97      0.96       115
avg / total       0.94      0.94      0.94       174
```

7) ROC (Region of Convergence)



Model başarımını ölçmek için ROC kullanılabilir. Elde hazırda model varsa bu grafik nasıl çıkarıldığının kodu aşağıda. Şimdilik ROC un detayına girmemize gerek yok. **Sadece bilmemiz gereken şey bu grafikteki eğrinin altında kalan alan ne kadar büyükse modelimiz o kadar başarılır**.

```
# Import necessary modules
from sklearn.metrics import roc_curve

# Compute predicted probabilities: y_pred_prob
y_pred_prob = logreg.predict_proba(X_test)[:,1]

# Generate ROC curve values: fpr, tpr, thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)



# Plot ROC curve

plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr,tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.show()
```

7.1) ROC and AUC (Area Under the Curve)

Diyelim ki elimizde yazı tura tahmini yapan ve her atışa tura tahmininde bulunan bir classifier modeli olsaydı.Yaklaşık olarak %50 doğru cevabı verecekti ve eğri tam olarak sağ taraftaki grafikteki kesikli çizgi gibi olup ve altında kalan alan da yaklaşık 0.5 olacaktı.

```
# Import necessary modules
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_auc_score

# Compute predicted probabilities: y_pred_prob
y_pred_prob = logreg.predict_proba(X_test)[:,1]

# Compute and print AUC score
print("AUC: {}".format(roc_auc_score(y_test, y_pred_prob)))

# Compute cross-validated AUC scores: cv_auc
cv_auc = cross_val_score(logreg, X, y, cv=5, scoring='roc_auc')

# Print list of AUC scores
print("AUC scores computed using 5-fold cross-validation: {}".format(cv_auc))
```

```
AUC: 0.8254806777079764
AUC scores computed using 5-fold cross-validation: [0.80148148 0.8062963  0.81481481 0.86245283 0.8554717 ]
```

8) Hyperparameter Tuning (Grid SearchCV)

Her model türü birbirinden farklı çalışır ve farklı parametreleri vardır. Bu parametreleri modeli elimizdeki datasetine yerleştirmeden önce belirleriz.
Örnek;
Ridge/lasso regression: alpha (a)
k-Nearest Neighbors: n _ neighbors (k)
Alfa ve k gibi parametrelere “Hyperparameter” denir ve en iyi değerin hangi değer olduğunu modeli datasetine yerleştirerek keşfedemeyiz.

Bu hyperparametrelerin en iyisini keşfetmenin yolu birkaç farklı değeri farklı modellere verip en iyi sonucun hangi değerlerle alındığını gözlemlemektir.
Bu işi yapmamızı sağlayan metoda “**Grid search**” denir.

NOT_1: Eğer Cross validation ve grid search beraber kullanılır. En iyi cross validation ortalamasına sahip parametreleri tercih ederiz.

NOT_2:Hatta ve hatta overfitting den kaçınmak için cross-validationun yanı sıra bir de “hold-out” seti ayırırız ve en iyi hyperparametreleri GridSearchCV ile keşfettikten sonra model başarımını bu “hold-out” set ile sınarız. (Aşağıdaki kodda test setimiz hold-out set.)

```
# Import necessary modules
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

# Create the hyperparameter grid
c_space = np.logspace(-5, 8, 15)
param_grid = {'C': c_space, 'penalty': ['l1', 'l2']}

# Instantiate the logistic regression classifier: logreg
logreg = LogisticRegression()
```

```
# Create train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.4, random_state=42)

# Instantiate the GridSearchCV object: logreg_cv
logreg_cv = GridSearchCV(logreg, param_grid, cv=5)


# Fit it to the training data
logreg_cv.fit(X_train, y_train)

# Print the optimal parameters and best score
print("Tuned Logistic Regression Parameter: {}".format(logreg_cv.best_params_))
print("Tuned Logistic Regression Accuracy: {}".format(logreg_cv.best_score_))
```

```
Tuned Logistic Regression Parameter: {'C': 0.4393970560760795, 'penalty': 'l1'}
Tuned Logistic Regression Accuracy: 0.7652173913043478
```

9) Preprocessing

9.1) Encoding Dummy Variables

Elimizde "Asya","Avrupa","Amerika" gibi sayısal olmayan kategoriler varsa bunları scikit-learn tarafından anlaşılması için 1 ve 0 olarak encode etmemiz gerekir. Bu işleme "Dummy Variable" üretmek denir. Pandas'ın bu değişkenleri üretmek için "pd.get_dummies()" fonksiyonu vardır.

Origin
US
Europe
Asia

→

origin_Asia	origin_Europe	origin_US
0	0	1
0	1	0
1	0	0

Not: Burada bir akıllılık yapıp son sütunu kaldırabiliriz. Çünkü 3 tane sınıfımız var ve eğer Asya ve Amerika değilse biliriz ki Avrupadır. (0 0 = = US). Bunu yapmamak bazı modellerde soruna yol açabilir. (Duplicating Information)

Origin
US
Europe
Asia

→

origin_Asia	origin_US
0	1
0	0
1	0

```
import pandas as pd
df = pd.read _ csv('auto.csv')
df _ origin = pd.get _ dummies(df)

# We can drop Asia column to avoid Duplicating Information
df _ origin = df _ origin.drop('origin _ Asia' , axis=1)
print(df _ origin.head())
```

```
    mpg  displ   hp  weight  accel  size  origin_Europe  origin_US
0  18.0  250.0   88    3139   14.5  15.0              0          1
1   9.0  304.0  193    4732   18.5  20.0              0          1
2  36.1   91.0   60    1800   16.4  10.0              0          0
3  18.5  250.0   98    3525   19.0  15.0              0          1
4  34.3   97.0   78    2188   15.8  10.0              1          0
```

9.2) Handling Missing Data

Datasetimizde bazen eksik veriler NaN değer yerine “0”, “?”, “-1” gibi değerler olarak kaydedilmiş olabiliyor. Örnek olarak elimizde bir doktorun tuttuğu hastaların verisi var ve bazı hastaların insülin değerleri 0 olarak girilmiş. Biliyoruz ki bir insanın insülin seviyesi 0 olamaz bu yüzden bu değerleri “np.nan” ile değiştirmemiz gerekiyor.

```
df.insulin.replace(0, np.nan, inplace=True)
```

Modeli yanıltmasını istemediğmiz bu değerleri NaN olarak değiştirdikten sonra istersek bu bilinmeyen değere sahip hastaların tüm bilgilerini modelden dışarda tutabiliriz.
Yada bu insülin değeri bilinmeyen hastaların geri kalan verilerini kullanabilmek bu değerlere mantıklı bir değer atfedebiliriz. Bu işleme “**Imputation**” denir. Bilinmeyen değerlere geri kalan değerlerin ortalamasını atamak yada bir önceki değere eşitlemek gibi çözümler getirilebilir.

```
from sklearn.preprocessing import Imputer
imp = Imputer(missing_ values='NaN' , strategy='mean' , axis=0)
imp.fit(X)
X = imp.transform(X)
```

9.3) Scaling

Bazı veri setlerinde, veriler çok farklı büyüklüklere sahip olabilir. Bu durum lineer modellerde tahminde bulunmayı zorlaştırabilir ve Support Vector Machine gibi algoritmalarda ise sonuçta çok kötü sonuç çıkmasına sebep olabilir. Bu durumdan kaçınmak için veriyi ölçeklendirmek (scaling) gerekir.

Scaling verimizdeki sayısal sütunların her birindeki değerleri “max = 1 ,min = 0” olarak yada “-1 ve +1” arası ölçeklendirmektir. Daha farklı ölçeklendirme metodları için scikit-learn dökümanlarına bakılabilir.

```
from sklearn.preprocessing import scale
X _ scaled = scale(X)

np.mean(X), np.std(X)
```

```
(8.13421922452, 16.7265339794)
```

```
np.mean(X _ scaled),  np.std(X _ scaled)
```

```
(2.54662653149e-15, 1.0)
```

10) Creating a Pipeline

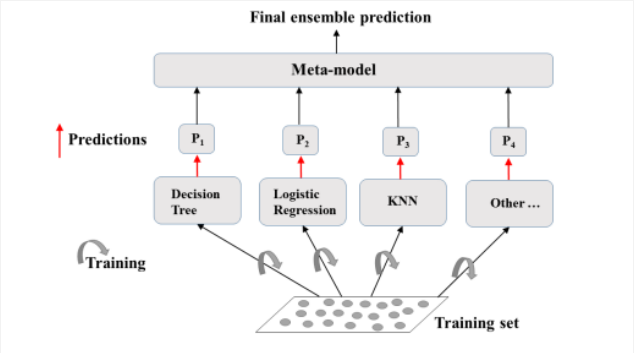
Pipeline çalışmamızın daha okunabilir olmasını sağlar. İmplementasyonun adımlarının doğru atıldığından emin olmamızı sağlar ve dolayısıyla çok daha “reproducible” bir kod yazmış oluruz.

```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
```

```
# Setup the pipeline
steps = [('scaler', StandardScaler()),
         ('SVM', SVC())]

pipeline = Pipeline(steps)

# Specify the hyperparameter space
parameters = {'SVM__C':[1, 10, 100],
              'SVM__gamma':[0.1, 0.01]}



# Create train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=21)

# Instantiate the GridSearchCV object: cv
cv = GridSearchCV(pipeline, parameters)

# Fit to the training set
cv.fit(X_train, y_train)

# Predict the labels of the test set: y_pred
y_pred = cv.predict(X_test)

# Compute and print metrics
print("Accuracy: {}".format(cv.score(X_test, y_test)))
print(classification_report(y_test, y_pred))
print("Tuned Model Parameters: {}".format(cv.best_params_))
```

```
<script.py> output:
    Accuracy: 0.7795918367346939
                 precision    recall  f1-score   support

          False       0.83      0.85      0.84       662
           True       0.67      0.63      0.65       318

    avg / total       0.78      0.78      0.78       980

    Tuned Model Parameters: {'SVM__C': 10, 'SVM__gamma': 0.1}
```

11) Ensemble Learning



Aynı veri üzerinde farklı tür model eğitip hepsinin kendi tahminlerini yapması ve
Sonra bu tahminlerin hepsini birleştirip bir “meta-model” oluşturmak.
Her modelin kendine özgü becerileri olduğu için çok daha sağlam tahminlerde
bulunabiliriz.Örnek olarak 5 ayrı modelimiz olduğunu düşünelim ve bu Modellerden 4 tanesi “1” tahmininde bulunurken bir tanesi “0” tahmininde bulunsun.
Bu tahminler arasında oylama yapıldığında cevap “1” olacaktır
Ve yüksek ihtimal doğru tahmindir.

```
# Import functions to compute accuracy and split data

from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

# Import models, including VotingClassifier meta-model

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier as KNN
from sklearn.ensemble import VotingClassifier

# Set seed for reproducibility

SEED = 1

# Split data into 70% train and 30% test

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.3, random_state= SEED)

# Instantiate individual classifiers

lr = LogisticRegression(random_state=SEED)
knn = KNN()
dt = DecisionTreeClassifier(random_state = SEED)


# Define a list called classifier that contains the tuples (classifier_name, classifier)

classifiers = [('Logistic Regression', lr),
               ('K Nearest Neighbours', knn),
               ('Classification Tree', dt)]


# Iterate over the defined list of tuples containing the classifiers

for clf_name, clf in classifiers:
    #fit clf to the training set
    clf.fit(X_train, y_train)
    # Predict the labels of the test set
    y_pred = clf.predict(X_test)
    # Evaluate the accuracy of clf on the test set
    print('{:s} : {:.3f}'.format(clf_name, accuracy_score(y_test, y_pred)))




# Instantiate a VotingClassifier 'vc'

vc = VotingClassifier(estimators=classifiers)
```

```
# Fit 'vc' to the traing set and predict test set labels

vc.fit(X_train, y_train)
y_pred = vc.predict(X_test)

# Evaluate the test-set accuracy of 'vc'

print('Voting Classifier: {:.3f}'.format(accuracy_score(y_test, y_pred)))
```

```
Logistic Regression: 0.947
K Nearest Neighbours: 0.930
Classification Tree: 0.930
```

```
Voting Classifier: 0.953
```

Makine öğrenmesindeki genel kavramların üzerine hazırladığım raporun burada sonuna geldim. Veri bilimi ve yapay zekâ konularının ne kadar uçsuz bucaksız olduğunun farkındayım ve çalışmalarım bu rapordan ibaret kalmayacak. Bu raporu hazırlarken önceden yaptığım araştırmaları derlemek ve konuya yeni başlayacak birine bir rehber görevi görebilecek şekilde tasarlamak yaklaşık 10 iş günümü aldı.

Bir sonraki adım olarak veriyi nasıl makine öğrenmesine sunulabilir hale getirebilirim üzerine çalışacağım. Bu veri üniversitede öğrendiğim dijital sinyal işleme yöntemlerini kullanarak da olabilir ya da internetten bulduğum bir dataseti inceleyip üzerine yorum yaparak bilgi çıkarmak da olabilir. Kendime çizdiğim yol haritasında,

1) Python Programlama, Algoritmalar ve Veri yapıları,
2) Numpy, matplotlib,scipy,scikit-learn, nltk ,keras ve benzeri veri bilimi kütüphanelerinde tecrübe edinmek,
3) Görüntü ve Ses dosyalarını işleyebilmek,
4) Web scraping örnekleriyle çalışmak,
5) Kendi projelerimi gerçekleştirebiliyor olmak,
6) Cloud services, Git ve “veri ile hikâye anlatımı” gibi konularda tecrübe edinmek,
7) Kendi Projelerimi iş hayatına entegre edebiliyor konumda olmak.

Stajımın ilk bölümünde yaptığım bu araştırmadan sonra mezuniyetime kadar olan hedefim, bu adımları teker teker atmış olmak. Buraya kadar raporumu okuduysanız ilginiz için teşekkür ederim.

Öğrencinin imzası:

Yetkili personel

İsim ve İmza:

Bölüm: Elektrik - Elektronik Mühendisliği	**Tarihler:** 2021.08.31 -2021.09.02
İş Tanımı: Makine Öğrenmesi Projeleri	**Süre: 3 gün**

Yaptığım çalışmalarda makine öğrenmesi ile ilgili sayısız proje inceledim, başlangıç seviyesindeki bir öğrenciye uygulamalı ve basit örnek olacak bu projeleri staj defterime dahil etmeyi uygun gördüm.

Bir önceki kısımda kalan örneklerin hepsi biz konuyu anlayalım diye hazırlanmışlardı. Artık bundan sonraki projeler ise anladığımız konuyu ve "tool"ları kullanarak **bir soruyu cevaplamış olmak** için varlar.

Supervised Learning

Song Genre Classification

https://github.com/MerttErturkk/Internship_repository/tree/main/SONG%20GENRE%20CLASSIFICATION

Predicting Credit Card Approvals

https://github.com/MerttErturkk/Internship_repository/tree/main/Predicting%20Credit%20Card%20Approvals

Who is Tweeting Trump or Trudeau?

https://github.com/MerttErturkk/Internship_repository/tree/main/Who's%20Tweeting_%20Trump%20or%20Trudeau_

Unsupervised Learning

Find Movie Similarity From Plot Summaries

https://github.com/MerttErturkk/Internship_repository/tree/main/Find%20Movie%20Similarity%20from%20Plot%20Summaries

Öğrencinin imzası:

Yetkili personel

İsim ve İmza:

Bölüm: Elektrik - Elektronik Mühendisliği	**Tarihler:** 21.09.03 -2021.09.06
İş Tanımı: Pandas ve Matplotlib ile Veri Analizi ve Veri Görselleştirmesi	**Süre: 2 gün**

Makine öğrenmesini temellerini anladıktan sonra internet üzerinden veriyi hazırlama ve makine öğrenmesi ile cevap vermek istediğim soruyu algoritmaya doğru tarif edebilmek için veri işleme ve istatistik gibi genel konseptlere daha fazla aşina olmam gerektiğini düşündüm. Bu yüzden önceden az da olsa kullanmış olduğum Matplotlib, Numpy, Pandas kütüphaneleriyle pratik yapabileceğim projeleri araştırmaya başladım.

Bu iki günümü bu araştırdığım projelerden veri işlemeyle ilgili temel fikirleri verebilecek 3 farklı rehberli Jupyter Notebook projesini seçip bunların analizini yapmaya ayırdım.

Bu 3 projeyi de “datacamp.com” un Projects sekmesinden aldım. Araştırmalarım sırasında datacamp çok faydalı bir kaynak oldu.

Proje 1
Analysing Netflix Data

https://github.com/MerttErturkk/Internship_repository/tree/main/Analysing%20Netflix

Proje 2
Discovery of Handwashing : Dr. Ignaz Semmelweiss

https://github.com/MerttErturkk/Internship_repository/tree/main/Discovery%20of%20Handwashing%20Dr.Semmelweis

Proje 3
Writing Functions For Product Analysis

https://github.com/MerttErturkk/Internship_repository/tree/main/Writing%20Functions%20for%20Product%20Analysis

Öğrencinin imzası:

Yetkili personel

İsim ve İmza:

Bölüm: Elektrik – Elektronik Mühendisliği	**Tarihler:** 2021.09.07 -2021.09.10
İş Tanımı: Exploratory Data Analysis with Python	**Süre: 4 gün**

Bu kısımda bu staja başlamadan önce konuya ilgimi uyandıran kitaplardan biri olan “ThinkStats2”nin içeriğinden derlenmiş Allan Downey Hocanın “Datacamp” kursundan dikkatimi çeken noktaları raporlayacağım.

“Exploratory Data Analysis” veri setlerini keşfetmek, temel soruları cevaplamak ve sonuçları görselleştirmeyi içeren işlemdir.

Bu çalışmada yapılacaklar;

- Veriyi temizlemek,
- Dağılımları görselleştirmek,
- Değişkenler arasındaki ilişkiyi keşfetmek,
- Regresyon modelleri kullanarak tahmin yürütüp yapılan keşfin önemini anlatmak.

Bu işlemlerden sonra makine öğrenmesi modelini kurmaya geçebiliriz ve yahut bu “keşif” işlemini yaparken ulaştığımız bilgilerle karmaşık bir makine öğrenmesi modeli kurmadan da kafamızdaki soruya cevap bulabiliriz. İstatistik bilmek veri bilimi sayılabilecek tüm alanlarda çok işe yarayan bir beceridir.
Bu kısımda akla takılabilecek tüm kod parçalarına ve kütüphanelere bu linkten ulaşılabilir.

- https://github.com/AllenDowney/ThinkStats2

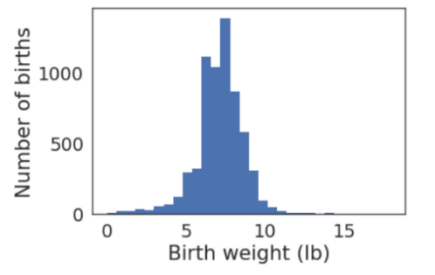
1) Importing / Cleaning the Data

```
import pandas as pd

#read the data
nsfg = pd.read_hdf('nsfg.hdf5','nsfg')

print(nsfg.head())
print(nsfg.shape)
print(nsfg.columns)

# Assign birthweight of the newborns to variable
pounds = nsfg['birthwgt_lb1']

print(pounds.describe())
```

```
count    6485.000000
mean        8.055204
std        11.178893
min         0.000000
25%         6.000000
50%         7.000000
75%         8.000000
max        99.000000
Name: birthwgt_lb1, dtype: float64
```

```
#We see that there is a new born weighing 99 pounds and it is imposible
#We check if there is more faulty data / then we replace them with NaN
print(pounds.value_counts().sort_index())

pounds.replace([99, 98,0], np.nan, inplace=True)
print(pounds.describe())
```

```
import matplotlib.pyplot as plt

plt.hist(pounds.dropna(), bins=30)
plt.xlabel('Birth weight (lb)')
plt.ylabel('Fraction of births')
plt.show()
```




```
# See which of the babies are
# "erken dogum bebekler" or "full-term"
preterm = nsfg['prglngth'] < 37
print(preterm.head(3))

# Count the True values
print(preterm.sum())

# Calculate the percentage
print(preterm.mean())


# Filter for babies that are "not-preterm"
full_term_weight = pounds[~preterm]
print(full_term_weight.mean())

# Note: we can also use other logical operators like
# '&, |' (and, or) for logical operations
```

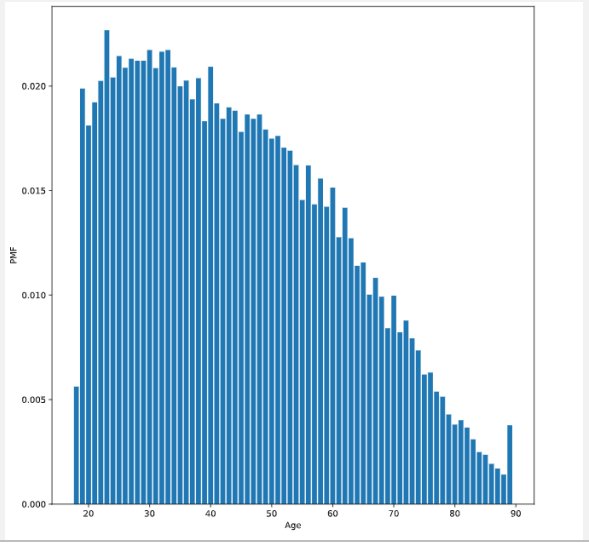
```
0    False
1     True
2     True
Name: prglngth, dtype: bool
3742
0.39987176747168196
```

```
7.372323879231473
```

2) Distributions

2.1) Probability Mass Function (PMF)

```
# Select the age column
age = gss['age']

# Make a PMF of age
pmf_age = Pmf(age)    # normalize = False for
                      # actual amounts

# Plot the PMF
pmf_age.bar()

# Label the axes
plt.xlabel('Age')
plt.ylabel('PMF')
plt.show()
```



2.2) Cumulative Distribution Function (CDF)

PMF de aradığımız değer(ler)in ihtimali hesaplanır.
CDF de ise aradığımız değerden küçük bir değer olma ihtimali hesaplanır.

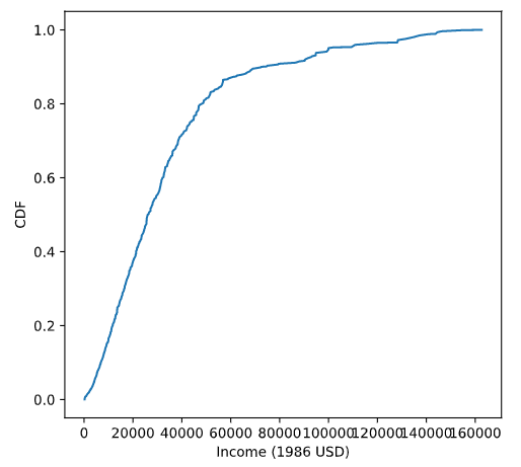
```
# Select the age column
age = gss['age']

# Compute the CDF of age
cdf_age = Cdf(age)

# Calculate the CDF of 30
print(cdf_age[30])

# what is the 75th percentile of age
print(cdf_age.inverse(0.75))
```

```
0.2539137136526388
57.0
```

```
# Calculate the 75th percentile
percentile_75th = cdf_age.inverse(0.75)

# Calculate the 25th percentile
percentile_25th = cdf_age.inverse(0.25)

# Calculate the interquartile range
iqr = percentile_75th - percentile_25th
```

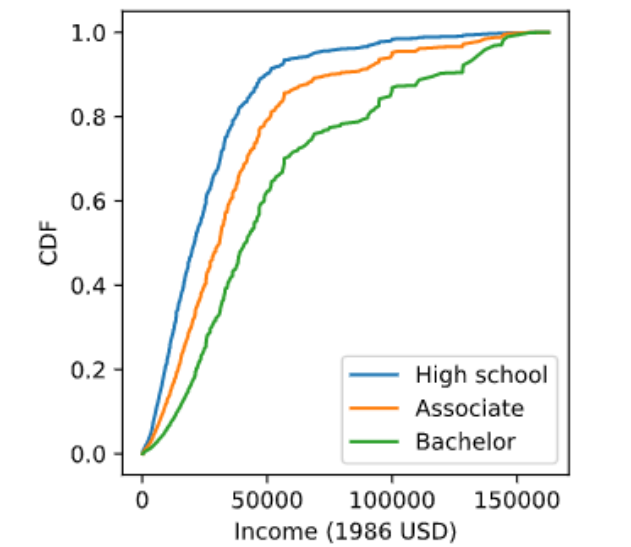
```
27.0
```

```
# Select realinc
income = gss["realinc"]

# Make the CDF
cdf_income = Cdf(income)

# Plot it
cdf_income.plot()

# Label the axes
plt.xlabel('Income (1986 USD)')
plt.ylabel('CDF')
plt.show()
```



2.3) Comparing Distributions

```
# Select educ
educ = gss['educ']

# Bachelor's degree
bach = (educ >= 16)

# Associate degree
assc = (educ >= 14) & (educ < 16)

# High school
high = (educ <= 12)

income = gss['realinc']

# Plot the CDFs of the 3 groups of education
# We could also use "~" for not operation

Cdf(income[high]).plot(label='High school')
Cdf(income[assc]).plot(label='Associate')
Cdf(income[bach]).plot(label='Bachelor')

# Label the axes
plt.xlabel('Income (1986 USD)')
plt.ylabel('CDF')
plt.legend()
plt.show()
```

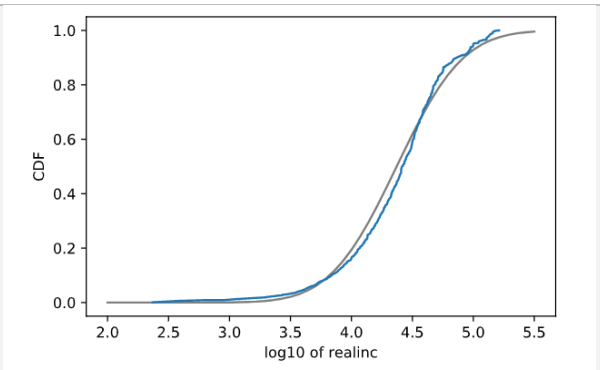


2.4) Probability Density Functions (PDF) / Kernel Density Estimation (KDE)

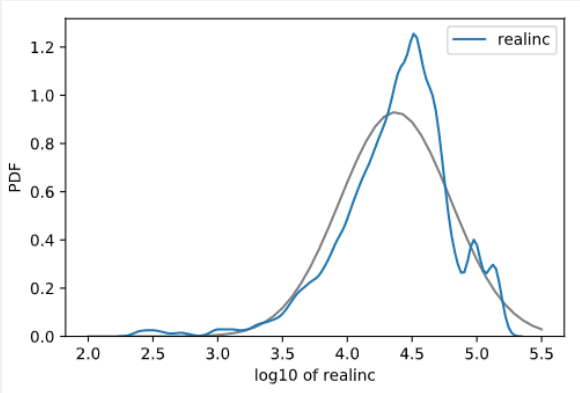
Keşif için CDF kullanırız.
Az sayıda unique değere sahipsek PMF işimizi görür.
Çok sayıda değer varsa KDE kullanmalıyız.

```
# Extract realinc and compute its log
income = gss['realinc']
log_income = np.log10(income)

# Compute mean and standard deviation
mean = log_income.mean()
std = log_income.std()
print(mean, std)

# Make a norm object
from scipy.stats import norm
dist = norm(mean,std)

# Evaluate the model CDF
xs = np.linspace(2, 5.5)
ys = dist.cdf(xs)
```

```
# Plot the model CDF
plt.clf() # Clear the current figure
plt.plot(xs, ys, color='gray')

# Create and plot the Cdf of log_income
Cdf(log_income).plot()

# Label the axes
plt.xlabel('log10 of realinc')
plt.ylabel('CDF')
plt.show()
```

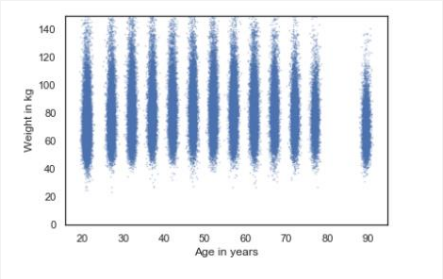



```
# Evaluate the normal PDF
xs = np.linspace(2, 5.5)
ys = dist.pdf(xs)

# Plot the model PDF
plt.clf()
plt.plot(xs, ys, color='gray')
# Plot the data KDE

import seaborn as sns
sns.kdeplot(log_income)

# Label the axes
plt.xlabel('log10 of realinc')
plt.ylabel('PDF')
plt.show()
```



3) Exploring Relationships

```
# Select the first 1000 respondents
brfss = brfss[:1000]

# Extract age and weight
age = brfss['AGE']
weight = brfss['WTKG3']

# Make a scatter plot
age = brfss['AGE'] + np.random.normal(0, 0.5, size=len(brfss))
weight = brfss['WTKG3'] + np.random.normal(0, 2, size=len(brfss))
plt.plot(age, weight,'o', markersize=1, alpha=0.2)
plt.xlabel('Age in years')
plt.ylabel('Weight in kg')

plt.show()
```



NOT: Bu tarz görselleştirmeler için Seaborn kütüphanesinde olan ViolinPlot / Box plot daha göze hitap eden görseller çıkarır. Sunumlara koymak için Seaborn'a aşinalık kazanılabilir.

```
# Drop rows with missing data / they can me misleading
data = brfss.dropna(subset=['_HTMG10', 'WTKG3'])
```

3.1) Correlation

Korelasyon ismi dolayısıyla çok önemli bir istatistik gibi gelebilir ama yanıltıcı olma ihtimali çok yüksektir.

- İki sütun arasındaki ilişki her zaman lineer olmak zorunda değil (x**2) gibi bir ilişkiyi korelasyon ile saptayamayız.
- Ayrıca korelasyon bize aynı yönde eğilim olduğunu söylese bile eğim hakkında bilgi vermez.

(Örnek boy ve kilo arasında korelasyon olduğunu bilebiliriz ama eğim miktarı hakkında yorum yapamayız. Her 10 santim için 10 kilo fark da olabilir 1 kilo fark da olabilir, bunu korelasyon ile öğrenemeyiz.)

```
# Select columns
columns = ["AGE","INCOME2","_VEGESU1"]
subset = brfss[columns]

# Compute the correlation matrix
print(subset.corr())
```

```
<script.py> output:
                AGE   INCOME2  _VEGESU1
    AGE       1.000000 -0.015158 -0.009834
    INCOME2  -0.015158  1.000000  0.119670
    _VEGESU1 -0.009834  0.119670  1.000000
```

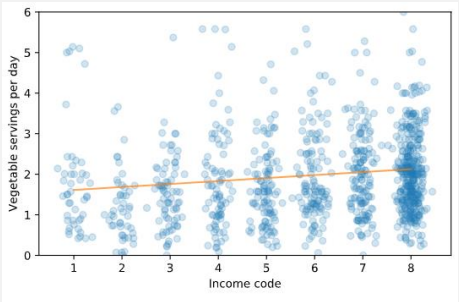
3.2) Simple Regression / Fitting a Line with Linear Regression

```
from scipy.stats import linregress

# Extract the variables
subset = brfss.dropna(subset=['INCOME2', '_VEGESU1'])
xs = subset["INCOME2"]
ys = subset["_VEGESU1"]

# Compute the linear regression
res = linregress(xs,ys)
print(res)
```

```
LinregressResult(slope=0.023981159566968724,
                 intercept=80.07977583683224,
                     rvalue=0.021641432889064068,
                     pvalue=4.374327493007566e-11,
                     stderr=0.003638139410742186)
```

Lineer regresyonla elde ettiğimiz katsayılardan korelasyonun yanı sıra istediğimiz eğim değerini de bulabiliriz ve bu sayede örneklerimize en iyi oturan çizgiyi hesaplayabiliriz.
(Burada “fx” iki ögeli “xs”’in min ve max değeri, regresyonla elde ettiğimiz intercept noktası ile bu “fx” değerlerinin ve eğimin çarpımı ile bu en iyi doğruyu çizebiliriz.)
Ayrıca burada hesaplanan “rvalue” korelasyondur.

```
# Plot the scatter plot
plt.clf()
x_jitter = xs + np.random.normal(0, 0.15, len(xs))
plt.plot(x_jitter, ys, 'o', alpha=0.2)

# Plot the line of best fit
fx = np.array([xs.min(), xs.max()])
fy = res.intercept + res.slope * fx
plt.plot(fx, fy, '-', alpha=0.7)

plt.xlabel('Income code')
plt.ylabel('Vegetable servings per day')
plt.ylim([0, 6])
plt.show()
```



NOT: Bu grafiğe bakarak güvenle sebze tüketimi ve gelir arasında mantıklı pozitif bir korelasyon olduğunu söyleyebiliriz.

4) Multiple Regression

Regresyonun başka bir adı da“OLS”dir yani “Ordinary Least Squares”.

Bu örnekte başka bir kütüphaneye geçiyoruz bu ilk örnek önceki örnekle aynı işi yapacak. Sonraki aşamalar için kütüphaneye aşinalık yaratması için dahil ettim.

Tekli regresyon

```
import statsmodels.formula.api as smf

# Run regression with StatsModels
results = smf.ols('_VEGESU1 ~ INCOME2', data = brfss).fit()
print(results.params)
```

```
Intercept    1.528779
INCOME2      0.069880
dtype: float64
```

Bu kod bize veriye en iyi oturan çizginin Income eksenini nerede intercept ettiği ve doğrunun eğimini veriyor. Burada tahmin etmek istediğimiz değer “_VEGESU1”.

Çoklu Regresyon

```
import statsmodels.formula.api as smf

# Add a new column with educ squared
gss['educ2'] = gss["educ"]**2

# Run a regression model with educ, educ2, age, and age2
results = smf.ols("realinc ~ educ + educ2 + age + age2",data = gss).fit()

# Print the estimated parameters
print(results.params)
```

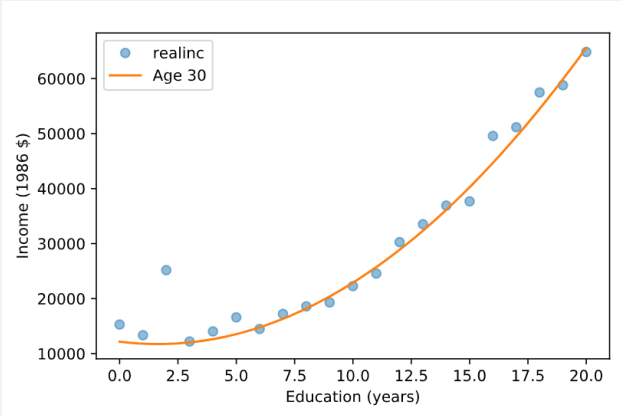
```
<script.py> output:
    Intercept   -23241.884034
    educ          -528.309369
    educ2          159.966740
    age           1696.717149
    age2           -17.196984
    dtype: float64
```

Burada “educ2” eğim değeri pozitif. Bu da demek oluyor ki modelimiz eğitim arttıkça yukarı doğru eğrileşiyor.

Sonraki örnekte yaşı 30 olarak sabit tutup eğitime göre gelirin nasıl değiştiğini görselleştireceğiz.

Visualization of Multple Regression

```
# Run a regression model with educ, educ2, age, and age2
results = smf.ols('realinc ~ educ + educ2 + age + age2', data=gss).fit()

# Make the DataFrame
df = pd.DataFrame()

df['educ'] = np.linspace(0,20)
df['age'] = 30
df['educ2'] = df['educ']**2
df['age2'] = df['age']**2

# Generate and plot the predictions
pred = results.predict(df)


# Plot mean income in each age group
plt.clf()
grouped = gss.groupby('educ')
mean_income_by_educ = grouped['realinc'].mean()
plt.plot(mean_income_by_educ, 'o', alpha=0.5)

# Plot the predictions
pred = results.predict(df)
plt.plot(df['educ'], pred, label='Age 30')

# Label axes
plt.xlabel('Education (years)')
plt.ylabel('Income (1986 $)')
plt.legend()
plt.show()
```

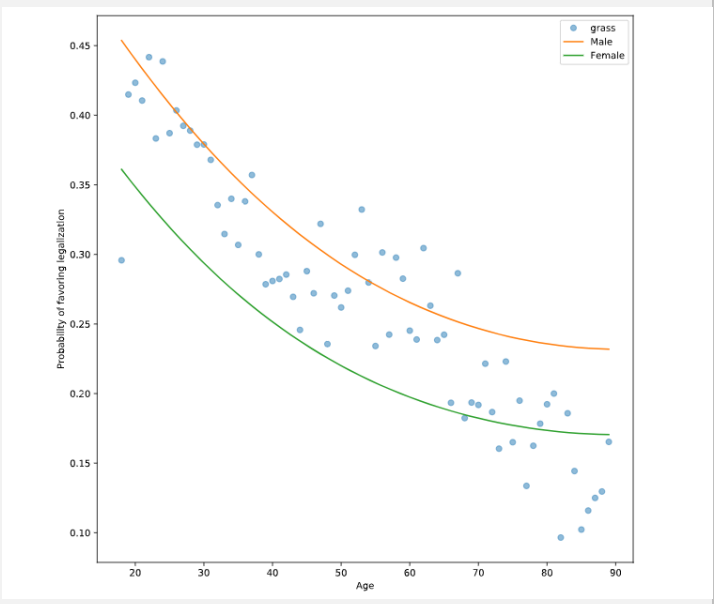


5) Logistic Regression

Aşağıdaki kodda Amerikan halkında 12 yıl eğitim görmüş kadın ve erkeklerin "marijuana" tüketiminin yasallaştırılmasından yana olup olmaması üzerine logistic regressionla yapılan tahminleri var. 12 yıl eğitim gören erkekler kadınlardan ortalama %10 daha fazla desteklediğini görüyoruz.

```
# Recode grass
gss['grass'].replace(2, 0, inplace=True)

# Run logistic regression
results = smf.logit('grass ~ age + age2 + educ + educ2 + C(sex)', data=gss).fit()
results.params

# Make a DataFrame with a range of ages
df = pd.DataFrame()
df['age'] = np.linspace(18, 89)
df['age2'] = df['age']**2
```

```
# Set the education level to 12
df['educ'] = 12
df['educ2'] = df['educ']**2
# Generate predictions for men and women
df['sex'] = 1
pred1 = results.predict(df)

df['sex'] = 2
pred2 = results.predict(df)

plt.clf()
grouped = gss.groupby('age')
favor_by_age = grouped['grass'].mean()
plt.plot(favor_by_age, 'o', alpha=0.5)

plt.plot(df['age'], pred1, label='Male')
plt.plot(df['age'], pred2, label='Female')

plt.xlabel('Age')
plt.ylabel('Probability of favoring legalization')
plt.legend()
plt.show()
```



Öğrencinin imzası:

Yetkili personel

İsim ve İmza:

Sonuç

20 günlük stajımın sonunda mühendislik kariyerim konusunda önemli tecrübeler edindiğimi düşünüyorum. Bu raporu hazırlamam 20 gün sürmüş olmasına rağmen ben bu çalışmaları yapmaya daha öncesinde başlamıştım. Sinyal işlemeyle ilgibi bir öğrenci olduğum için "Veri Mühendisliği"ne ilgi besliyordum ve bu toplanan veri ile 21. Yüzyılda ne yapılabilir diye düşünürken kendimi veri bilimi ve istatistiğin ortasında buldum.Bu stajım sayesinde üniversite başlangıcından beri hobi olarak öğrendiğim ve kullandığım Python programlama dilini mesleki olarak üretkenlik içinde kullanacağım bir alan keşfettim. Şimdiye kadar yaptığım çalışmalarımın beni getirdiği yer veri bilimi oldu ve bu süre zarfında zamanımı iyi değerlendirdiğimi düşünüyorum. Gelecekteki kariyerimde işime yarayacak tecrübeler edindim.

Bu raporum sadece makine öğrenmesi üzerine olmuş olsa da bu rapor vesilesiyle bilgisayar programlama jargonuyla daha fazla haşır neşir oldum. Yeri geldi algoritma ve veri tipleriyle ilgili bir kitap, makine öğrenmesinin temeli olan istatistiği anlamak için iki ayrı istatistik kitabı okudum. Extract, Transform, Load (ETL) hakkında bilgi edinmek için SQl öğrendim. İnternetten önceden hazırlanmış deep learning algoritmaları bulup tensor flow ve keras hakkında bilgi edindim. Scikit-learn'un yanı sıra scikit-image kütüphanesiyle çalışmayı ve görüntü işlemenin temellerini öğrendim. İnternetten bulduğum örneklerle Ses işleme ve ses verisinden feature extraction nasıl yapılır öğrendim. Kısacası bitirme projesini sadece "python ve makine öğrenmesi" üzerine yapmak istiyorum demiş olmam çok daha fazla şey öğrenmeme vesile oldu.

Ben bu raporu hazırlarken, hem ileride geri dönüp aklıma takılabilecek soruları cevaplamak ve aynı benim gibi bilgisayar programlama konusunda temeli olan ve bu bilgisiyle ne yapmak istediğini keşfetmek isteyen öğrencilere bir rehber olmasını istiyordum ve bu konuda başarılı olduğumu düşünüyorum. Bu raporda üstünden geçmediğim ve çok yüzeysel bıraktığım çok fazla konsept oldu ve bunu bilerek yaptım. Amacım bu konuyu özetlemekten ziyade okuyucuyu bir makine öğrenmesi projesinin genel hatlarına aşina olup gerçekten ilgi duyuyorsa direk Andreas Müllerin kitabından faydalanmasını sağlamaktı. Benim sıfırdan bu konuya başlamam biraz sancılı bir süreç olmuştu çünkü konuya girerken neyi bilmediğimi bile bilmeden işin içine kendimi attım ve çok üzerinde zaman harcamamış olmam gereken çok fazla kavramla boğuştum. Belki bu vesileyle öğrendiklerim daha fazla beynime kazınmıştır, olabilir. Ama hayat kısa ve her konuya bu kadar fazla zaman ayrılamaz ve illa bir konuyu öğrenmek sancılı bir süreç olmak zorunda değil daha fazla üretime yönelik bir öğrenme mantığı olması gerekli ve bu yüzden olabildiğince kısa tuttuğum temeller raporundan sonra hemen internetten bulduğum basit sayılabilecek projeleri raporuma dahil ettim.

Beni bu raporu hazırlarken kavramlara aşinalığımı sorgulayarak ve bir sonraki adımı sorduğumda beni cevapsız bırakmayan Arş. Gör. Basri Erdoğan'a, Aşırı temiz bir dilde makine öğrenmesini kitabıyla bana tanıtmış Andreas Müller hocaya ve öcü gibi gözüken istatistik konseptlerini bile eğlenceli bir şekilde anlatmayı becermiş Allan Downey hocaya teşekkürlerimi sunuyorum.