Lesson_2a_ML_Landscape

August 29, 2020

Chapter 1 – The Machine Learning landscape

"Machine Learning is field of study that gives computers the ability to learn without being explicitly programmed". Arthur Samuel, 1959

"A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E."

For spam detection exersise users first flag spam email on *training set*. Each email in the training set is an instance (or sample). The task (T) is to flag spam, P is the share of false identifiers.

** Machine Learning Approach vs Econometrics ** 1. In Econometrics model comes from a human, ML apprach is to generate/update model from the data. (draw pictures) 2. ML considers overfitting and out-of-sample prediction seriourly 3. ML help to understand the problem, not just to find a solution. For example, which feature are relevant for the problem. 4. ML is more efficient, but it requires a clear crietria to evaluate a model. It may be missing in some marginal effect estimations. Best prediction of Y does not equal to the best estimation of effect of X on Y.

To summarize, Machine Learning is great for: * Problems for which existing solutions require a lot of hand-tuning or long lists of rules: one Machine * Learning algorithm can often simplify code and perform better. * Complex problems for which there is no good solution at all using a traditional approach: the best * Machine Learning techniques can find a solution. * Fluctuating environments: a Machine Learning system can adapt to new data. * Getting insights about complex problems and large amounts of data.

** Type of ML Systems **

1. Supervised learning. All desired solutions are known. For example, a classification images of cats and dogs. Another example, is the predicting of future target value, like home price. Typical methods of supervised learning: k-nearest neighbors, regression, support vector machines, Decision trees and forests, some neural networks.
2. Unsupervised learning. The classifiers are not known in advance. For example, we need to identify clusters of consumers similar along multiple dimensions for advertisements (pic 1-8). The customers in a cluster should have similar preferences and the number of clusters should not too large.
3. The structire of these clusters should be determined by the ML algorithm. Typical methods are k-means, Hierarchical Cluster Analysis, Principal Component Analysis, visualisation. Often unsupervised learning helps to identify and to construct relevant parameters for the supervised learning exersise. Unsupervised learing can reduce the dimensions of the problem think about the number of products you buy every year. We may aggregate products into categories determined by the alhorithm reducing dimension and creating new features. Finally, using unsupervised learning we can detect anomalies and outliers (pic 1-10).

4. There are combincations of supervised and unsupervised learning
5. Reinforcement learning. This is an optimization problem where an agent performs actions for rewards and punishment. For example: Chess, go, computer games. The algorithm analyzes millions historic games or/and plays against itself many games rediscovering strategies.
6. Batch Learning – one estimation on full dataset. Online learning: system learns incrementally by adding one/few observations each time updating previously estimated results. Benefits of incremental learning: can be automated (24hr a day), fast, does not need many resources (drones, Mars exploration). Online learning is great for continious stream of data.
7. Instance learning is the learning only specific examples: recognition of users' identified cats in images. Model learning allows recognition of images not identified by users. (pic 1-15, 1-16).

Typical ML Project * Study the data * Select a model * Train on a training data. The model searches for the parameters to minimize costs function – the difference between prediction and observation * Apply the model on the new testing data. The good model should generalize well on the data not used in the training.

1 Testing and Validation

The only way to know how well a model will generalize to new cases is to actually try it out on new cases. Split your data into two sets: the training set and the test set. As these names imply, you train your model using the training set, and you test it using the test set. The error rate on new cases is called the generalization error (or out-of-sample error), and by evaluating your model on the test set, you get an estimation of this error. This value tells you how well your model will perform on instances it has never seen before.If the training error is low (i.e., your model makes few mistakes on the training set) but the generalization error is high, it means that your model is overfitting the training data.

So evaluating a model is simple enough: just use a test set. Now suppose you are hesitating between two models (say a linear model and a polynomial model): how can you decide? One option is to train both and compare how well they generalize using the test set.

Training many models: Now suppose that the linear model generalizes better, but you want to apply some regularization to avoid overfitting. The question is: how do you choose the value of the regularization hyperparameter? One option is to train 100 different models using 100 different values for this hyperparameter. Suppose you find the best hyperparameter value that produces a model with the lowest generalization error, say just 5% error.

So you launch this model into production, but unfortunately it does not perform as well as expected and produces 15% errors. What just happened?

Overtesting: The problem is that you measured the generalization error multiple times on the test set, and you adapted the model and hyperparameters to produce the best model for that set. This means that the model is unlikely to perform as well on new data.

Validation: A common solution to this problem is to have a second holdout set called the validation set. You train multiple models with various hyperparameters using the training set, you select the model and hyperparameters that perform best on the validation set, and when you're happy with your model you run a single final test against the test set to get an estimate of the generalization error.

Crossvalidation: To avoid "wasting" too much training data in validation sets, a common technique is to use crossvalidation: the training set is split into complementary subsets, and each model is trained against a different combination of these subsets and validated against the remaining parts. Once the model type and hyperparameters have been selected, a final model is trained using these hyperparameters on the full training set, and the generalized error is measured on the test set.

ML Challenges: 1. Insufficient Quantity of Training Data. You often need millions of example to train a complex model speech-recognition. 2. The choice of the algorithm matters less with large data 3. Non-representative data would bias the results. Countries examples with happy Latin America relatively unhappy high income European countries. 4. Poor quality data: missing observations, erroneous values, etc. Use imputation to fix the issue. 5. Irrelevant features: garbage in -> garbage-out, which would improve in-sample prediction and damage out-of-sample prediction. Need to select/engineer only relevant features.

2 Overfitting

Treating overfitting seriously is one of the main differences between ML and Econometrics. Say you are visiting a foreign country and the taxi driver rips you off. You might be tempted to say that all taxi drivers in that country are thieves. In ML the model performs well on the training data, but it does not generalize well.

High degree polynomial is great in predicting the training data and is very bad in predicting testing data. This problem is esspecially serious if the training data is small and noisy. Neural network can detect obscure data patterns that make little practical sense: like the fact that all countries in the training data with a w in their name have a life satisfaction greater than 7: New Zealand (7.3), Norway (7.4), Sweden (7.2), and Switzerland (7.5). How confident are you that the W-satisfaction rule generalizes to Rwanda or Zimbabwe? Obviously this pattern occurred in the training data by pure chance, but the model has no way to tell whether a pattern is real or simply the result of noise in the data.

3 What to do with overfitting?

1. To simplify the model by selecting one with fewer parameters (e.g., a linear model rather than a high-degree polynomial model), by reducing the number of attributes in the training data or by constraining the model. One way is regularization that eighter sets some parameters to zero, or reduces the size of all parameters. For example, in regression: $Y = \theta_0 + \theta_1 * X_1 + \theta_2 * X_2$ We can set $\theta_2 = 0$, to a have simpler linear model. In regulartization regressions (LASSO, RIDGE, Elastic Net) we can test various regularization *hyperparameters* to find optimal fitting on testing data.
2. To gather more training data
3. To reduce the noise in the training data (e.g., fix data errors and remove outliers)

4 Undefitting

Underfitting occurs when your model is too simple tolearn the underlying structure of the data. For example, a linear model of life satisfaction is prone to underfit; reality is just more complex than the model, so its predictions are bound to be inaccurate, even on the training examples.

The main options to fix this problem are: * Selecting a more powerful model, with more parameters * Feeding better features to the learning algorithm (feature engineering) * Reducing the constraints on the model (e.g., reducing the regularization hyperparameter)

5 No Free Lunch

In a famous paper David Wolpert demonstrated that if you make absolutely no assumption about the data, then there is no reason to prefer one model over any other. This is called the No Free Lunch (NFL) theorem. For some datasets the best model is a linear model, while for other datasets it is a neural network. There is no model that is a priori guaranteed to work better (hence the name of the theorem). The only way to know for sure which model is best is to evaluate them all. Since this is not possible, in practice you make some reasonable assumptions about the data and you evaluate only a few reasonable models. For example, for simple tasks you may evaluate linear models with various levels of regularization, and for a complex problem you may evaluate various neural networks.

6 Coding Part of the Lesson

First, let's make sure this notebook works well in both python 2 and 3, import a few common modules, ensure MatplotLib plots figures inline and prepare a function to save the figures:

[23]:
```
# To support both python 2 and python 3
from __future__ import division, print_function, unicode_literals

# Common imports. numpy is a package of math commands
import numpy as np
# os is the package for path file specification
import os

# to make this notebook's output stable across runs
np.random.seed(42)

# To plot pretty figures use matplotlib package
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
# Plot arameters
plt.rcParams['axes.labelsize'] = 14
plt.rcParams['xtick.labelsize'] = 12
plt.rcParams['ytick.labelsize'] = 12

def save_fig(fig_id, tight_layout=True):
    path = os.path.join(fig_id + ".png")
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format='png', dpi=300)
```

```
# Ignore useless warnings (see SciPy issue #5998)
import warnings
warnings.filterwarnings(action="ignore", module="scipy", message="^internal␣
↪gelsd")
```

7 Code example 1-1

This function just merges the OECD's life satisfaction data and the IMF's GDP per capita data.

[11]:
```
def prepare_country_stats(oecd_bli, gdp_per_capita):
    # Create a column "INEQUALITY with the string "TOT". Data for both genders.
    oecd_bli = oecd_bli[oecd_bli["INEQUALITY"]=="TOT"]
    # Sort data by country name
    oecd_bli = oecd_bli.pivot(index="Country", columns="Indicator",␣
↪values="Value")
    # Rename column name "2015" with the column name GDP per capita
    gdp_per_capita.rename(columns={"2015": "GDP per capita"}, inplace=True)
    # sort by Country name and set it as an index variable
    gdp_per_capita.set_index("Country", inplace=True)
    # Merge life satisfaction with GDP per capita. This is 1:1 merge. Left␣
↪columns are life satisfaction, right columns
    # GDP per capita. Merge by the index variables (Country names in both␣
↪datasets)
    full_country_stats = pd.merge(left=oecd_bli, right=gdp_per_capita,
                                  left_index=True, right_index=True)
    # sort by GDP per capita
    full_country_stats.sort_values(by="GDP per capita", inplace=True)
    return full_country_stats["GDP per capita", 'Life satisfaction']
```

The code in the book expects the data files to be located in the current directory. I just tweaked it here to fetch the files in datasets/lifesat.

[12]:
```
# To plot pretty figures directly within Jupyter
%matplotlib inline
import matplotlib as mpl
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)
import os
# Code example
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import sklearn.linear_model
```

[13]:
```
# Download the data
import urllib
DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml2/master/"
os.makedirs(datapath, exist_ok=True)
for filename in ("oecd_bli_2015.csv", "gdp_per_capita.csv"):
    print("Downloading", filename)
    url = DOWNLOAD_ROOT + "datasets/lifesat/" + filename
    urllib.request.urlretrieve(url, datapath + filename)
```

```
Downloading oecd_bli_2015.csv
Downloading gdp_per_capita.csv
```

8 Load and prepare Life satisfaction data

If you want, you can get fresh data from the OECD's website. Download the CSV from http://stats.oecd.org/index.aspx?DataSetCode=BLI and save it to `datasets/lifesat/`.

[14]:
```
# Code example
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import sklearn.linear_model

# Load the data
oecd_bli = pd.read_csv(datapath + "oecd_bli_2015.csv", thousands=',')
gdp_per_capita = pd.read_csv(datapath + "gdp_per_capita.
 ↪csv",thousands=',',delimiter='\t',
                             encoding='latin1', na_values="n/a")
```

[15]:
```
oecd_bli = oecd_bli[oecd_bli["INEQUALITY"]=="TOT"]
# Sort by country name
# Transpose data from long to wide
oecd_bli = oecd_bli.pivot(index="Country", columns="Indicator", values="Value")
# Akk columns
oecd_bli.head(2)
```

[15]:

Indicator	Air pollution	Assault rate	…	Water quality	Years in education
Country			…		
Australia	13.0	2.1	…	91.0	19.4
Austria	27.0	3.4	…	94.0	17.0

[2 rows x 24 columns]

[16]:
```
# Look at just one column
oecd_bli["Life satisfaction"].head()
```

[16]:
```
Country
Australia    7.3
Austria      6.9
Belgium      6.9
Brazil       7.0
Canada       7.3
Name: Life satisfaction, dtype: float64
```

9 Load and prepare GDP per capita data

Just like above, you can update the GDP per capita data if you want. Just download data from http://goo.gl/j1MSKe (=> imf.org) and save it to `datasets/lifesat/`.

[17]:
```
# Read GDP data
gdp_per_capita = pd.read_csv(datapath+"gdp_per_capita.csv", thousands=',',␣
 ↪delimiter='\t',
                             encoding='latin1', na_values="n/a")
# Rename column for 2015
gdp_per_capita.rename(columns={"2015": "GDP per capita"}, inplace=True)
# sort by Country
gdp_per_capita.set_index("Country", inplace=True)
gdp_per_capita.head(2)
```

[17]:
```
                                          Subject Descriptor  … Estimates
Start After
Country                                                       …
Afghanistan  Gross domestic product per capita, current prices  …
2013.0
Albania      Gross domestic product per capita, current prices  …
2010.0

[2 rows x 6 columns]
```

[18]:
```
# Create a file with all countries without dropping any
#Merge GDP and Life satisfaction data
full_country_stats = pd.merge(left=oecd_bli, right=gdp_per_capita,␣
 ↪left_index=True, right_index=True)
# Sort by GDPS
full_country_stats.sort_values(by="GDP per capita", inplace=True)
full_country_stats
```

[18]:
```
                Air pollution  …  Estimates Start After
Country                        …
Brazil                   18.0  …                 2014.0
Mexico                   30.0  …                 2015.0
Russia                   15.0  …                 2015.0
```

```
Turkey                    35.0  …                2013.0
Hungary                   15.0  …                2015.0
Poland                    33.0  …                2014.0
Chile                     46.0  …                2014.0
Slovak Republic           13.0  …                2015.0
Czech Republic            16.0  …                2015.0
Estonia                    9.0  …                2014.0
Greece                    27.0  …                2014.0
Portugal                  18.0  …                2014.0
Slovenia                  26.0  …                2015.0
Spain                     24.0  …                2014.0
Korea                     30.0  …                2014.0
Italy                     21.0  …                2015.0
Japan                     24.0  …                2015.0
Israel                    21.0  …                2015.0
New Zealand               11.0  …                2015.0
France                    12.0  …                2015.0
Belgium                   21.0  …                2014.0
Germany                   16.0  …                2014.0
Finland                   15.0  …                2014.0
Canada                    15.0  …                2015.0
Netherlands               30.0  …                2014.0
Austria                   27.0  …                2015.0
United Kingdom            13.0  …                2015.0
Sweden                    10.0  …                2014.0
Iceland                   18.0  …                2014.0
Australia                 13.0  …                2014.0
Ireland                   13.0  …                2014.0
Denmark                   15.0  …                2015.0
United States             18.0  …                2015.0
Norway                    16.0  …                2015.0
Switzerland               20.0  …                2015.0
Luxembourg                12.0  …                2014.0

[36 rows x 30 columns]
```

[19]:
```
# What about the US?
full_country_stats[["GDP per capita", 'Life satisfaction']].loc["United States"]
```

[19]:
```
GDP per capita       55805.204
Life satisfaction        7.200
Name: United States, dtype: float64
```

[20]:
```
remove_indices = [0, 1, 6, 8, 33, 34, 35]
keep_indices = list(set(range(36)) - set(remove_indices))
# Compare missing data with the data use in regression
```

```
sample_data = full_country_stats[["GDP per capita", 'Life satisfaction']].
 ↪iloc[keep_indices]
missing_data = full_country_stats[["GDP per capita", 'Life satisfaction']].
 ↪iloc[remove_indices]
```

[21]:
```
sample_data
```

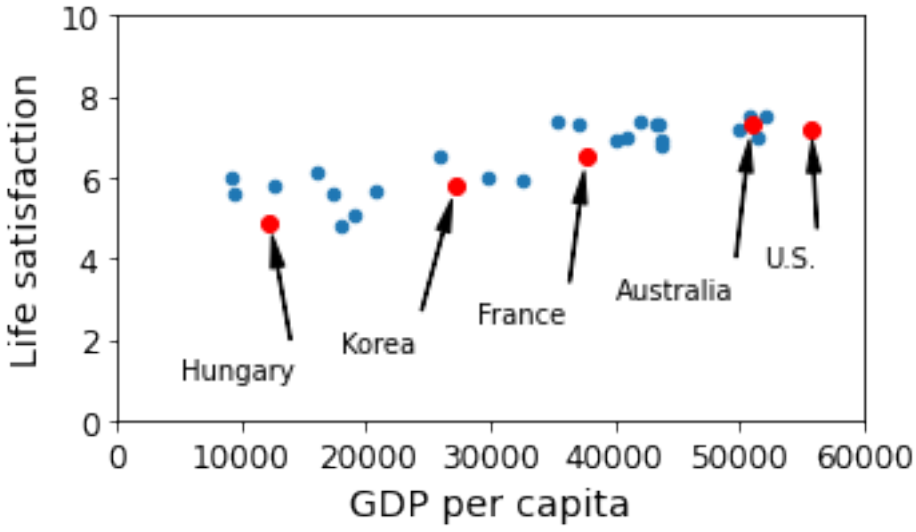
[21]:

Country	GDP per capita	Life satisfaction
Russia	9054.914	6.0
Turkey	9437.372	5.6
Hungary	12239.894	4.9
Poland	12495.334	5.8
Slovak Republic	15991.736	6.1
Estonia	17288.083	5.6
Greece	18064.288	4.8
Portugal	19121.592	5.1
Slovenia	20732.482	5.7
Spain	25864.721	6.5
Korea	27195.197	5.8
Italy	29866.581	6.0
Japan	32485.545	5.9
Israel	35343.336	7.4
New Zealand	37044.891	7.3
France	37675.006	6.5
Belgium	40106.632	6.9
Germany	40996.511	7.0
Finland	41973.988	7.4
Canada	43331.961	7.3
Netherlands	43603.115	7.3
Austria	43724.031	6.9
United Kingdom	43770.688	6.8
Sweden	49866.266	7.2
Iceland	50854.583	7.5
Australia	50961.865	7.3
Ireland	51350.744	7.0
Denmark	52114.165	7.5
United States	55805.204	7.2

[24]:
```
#plot data. Set axes
sample_data.plot(kind='scatter', x="GDP per capita", y='Life satisfaction',␣
 ↪figsize=(5,3))
# Set plot scale
plt.axis([0, 60000, 0, 10])
position_text = {
# Position markers for countries
    "Hungary": (5000, 1),
```

```
    "Korea": (18000, 1.7),
    "France": (29000, 2.4),
    "Australia": (40000, 3.0),
    "United States": (52000, 3.8),
}
# Loop over countries and positions.
for country, pos_text in position_text.items():
    #assigne country location
    pos_data_x, pos_data_y = sample_data.loc[country]
    # Shorten US
    country = "U.S." if country == "United States" else country
    # Plot text and arroaws
    plt.annotate(country, xy=(pos_data_x, pos_data_y), xytext=pos_text,
            arrowprops=dict(facecolor='black', width=0.5, shrink=0.1,␣
 ↪headwidth=5))
    plt.plot(pos_data_x, pos_data_y, "ro")
save_fig('money_happy_scatterplot')
plt.show()
```

```
Saving figure money_happy_scatterplot
```



[25]:
```
# save data to CSV
sample_data.to_csv(os.path.join("datasets", "lifesat", "lifesat.csv"))
```

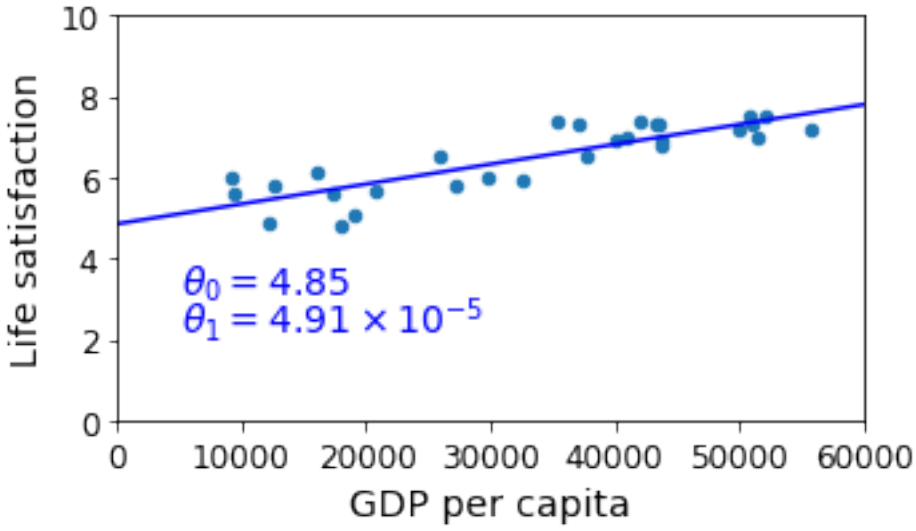
[33]:
```
# Estimate OLS
from sklearn import linear_model
lin1 = linear_model.LinearRegression()
# Set X and Y variables
Xsample = np.c_[sample_data["GDP per capita"]]
```

```
ysample = np.c_[sample_data["Life satisfaction"]]
# Fit the data to linear model
lin1.fit(Xsample, ysample)
t0, t1 = lin1.intercept_[0], lin1.coef_[0][0]
# Estimated coefficients
t0, t1
```

[33]: (4.853052800266436, 4.911544589158484e-05)

[34]:
```
# Plot estimated coefficietns
sample_data.plot(kind='scatter', x="GDP per capita", y='Life satisfaction',␣
 ↪figsize=(5,3))
plt.axis([0, 60000, 0, 10])
X=np.linspace(0, 60000, 1000)
plt.plot(X, t0 + t1*X, "b")
plt.text(5000, 3.1, r"$\theta_0 = 4.85$", fontsize=14, color="b")
plt.text(5000, 2.2, r"$\theta_1 = 4.91 \times 10^{-5}$", fontsize=14, color="b")
save_fig('best_fit_model_plot')
plt.show()
```

```
Saving figure best_fit_model_plot
```



[38]:
```
Brazil = full_country_stats[["GDP per capita", 'Life satisfaction']].
 ↪loc["Brazil"]
print("Brazil real life satisfaction", Brazil[1])
Brazil_gdp = Brazil[0]
print("Brazil real GDP per capita", Brazil_gdp)
Brazil_predicted_life_satisfaction = lin1.predict([[Brazil_gdp]])[0][0]
print("Predicted life satisfaction is ",Brazil_predicted_life_satisfaction)
```

```
Brazil real life satisfaction 7.0
Brazil real GDP per capita 8669.998
Predicted life satisfaction is  5.278883617915585
```

[41]:
```
# We miss few countries from the data
missing_data
```

[41]:

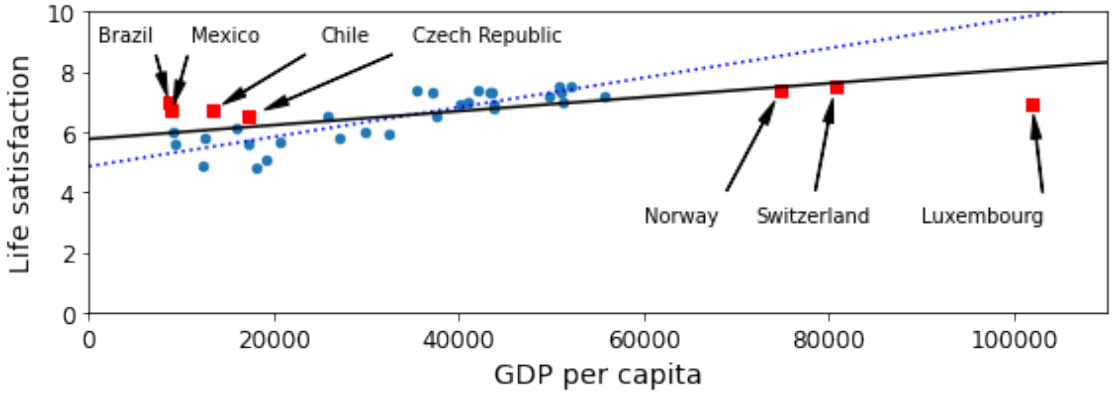
Country	GDP per capita	Life satisfaction
Brazil	8669.998	7.0
Mexico	9009.280	6.7
Chile	13340.905	6.7
Czech Republic	17256.918	6.5
Norway	74822.106	7.4
Switzerland	80675.308	7.5
Luxembourg	101994.093	6.9

[42]:
```
# Label missing countries
position_text2 = {
    "Brazil": (1000, 9.0),
    "Mexico": (11000, 9.0),
    "Chile": (25000, 9.0),
    "Czech Republic": (35000, 9.0),
    "Norway": (60000, 3),
    "Switzerland": (72000, 3.0),
    "Luxembourg": (90000, 3.0),
}
```

[43]:
```
# Plot full and restricted sample
sample_data.plot(kind='scatter', x="GDP per capita", y='Life satisfaction',␣
 ↪figsize=(8,3))
plt.axis([0, 110000, 0, 10])

for country, pos_text in position_text2.items():
    # Print missing countries labes
    pos_data_x, pos_data_y = missing_data.loc[country]
    plt.annotate(country, xy=(pos_data_x, pos_data_y), xytext=pos_text,
            arrowprops=dict(facecolor='black', width=0.5, shrink=0.1,␣
 ↪headwidth=5))
    plt.plot(pos_data_x, pos_data_y, "rs")

X=np.linspace(0, 110000, 1000)
# Blue is the line of the sample
plt.plot(X, t0 + t1*X, "b:")
lin_reg_full = linear_model.LinearRegression()
Xfull = np.c_[full_country_stats["GDP per capita"]]
yfull = np.c_[full_country_stats["Life satisfaction"]]
```

```
lin_reg_full.fit(Xfull, yfull)
t0full, t1full = lin_reg_full.intercept_[0], lin_reg_full.coef_[0][0]
X = np.linspace(0, 110000, 1000)
# Black is for full data
plt.plot(X, t0full + t1full * X, "k")
save_fig('representative_training_data_scatterplot')
plt.show()
```

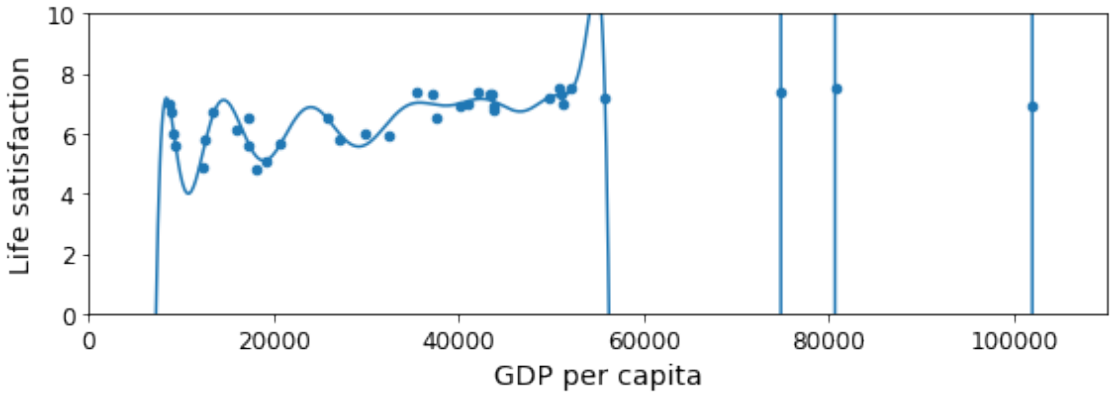
```
Saving figure representative_training_data_scatterplot
```



[44]:
```
full_country_stats.plot(kind='scatter', x="GDP per capita", y='Life␣
↪satisfaction', figsize=(8,3))
plt.axis([0, 110000, 0, 10])
# Import packages
from sklearn import preprocessing
from sklearn import pipeline
# Choose a more of 60th degree polynomic
poly = preprocessing.PolynomialFeatures(degree=60, include_bias=False)
# scale the features data by mean and staned deviation
scaler = preprocessing.StandardScaler()
lin_reg2 = linear_model.LinearRegression()
# Estimate polinomial using linear regression
pipeline_reg = pipeline.Pipeline([('poly', poly), ('scal', scaler), ('lin',␣
↪lin_reg2)])
pipeline_reg.fit(Xfull, yfull)
curve = pipeline_reg.predict(X[:, np.newaxis])
plt.plot(X, curve)
save_fig('overfitting_model_plot')
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/numpy/lib/nanfunctions.py:1546:
RuntimeWarning: overflow encountered in multiply
  sqr = np.multiply(arr, arr, out=arr)
```

```
/usr/local/lib/python3.6/dist-packages/numpy/core/fromnumeric.py:90:
RuntimeWarning: overflow encountered in reduce
  return ufunc.reduce(obj, axis, dtype, out, **passkwargs)

Saving figure overfitting_model_plot
```



[45]:
```
# Find life satistfaction of countries that have letter "w" in their name.
full_country_stats.loc[[c for c in full_country_stats.index if "W" in c.
 ↪upper()]]["Life satisfaction"]
```

[45]:
```
Country
New Zealand    7.3
Sweden         7.2
Norway         7.4
Switzerland    7.5
Name: Life satisfaction, dtype: float64
```

[46]:
```
gdp_per_capita.loc[[c for c in gdp_per_capita.index if "W" in c.upper()]].head()
```

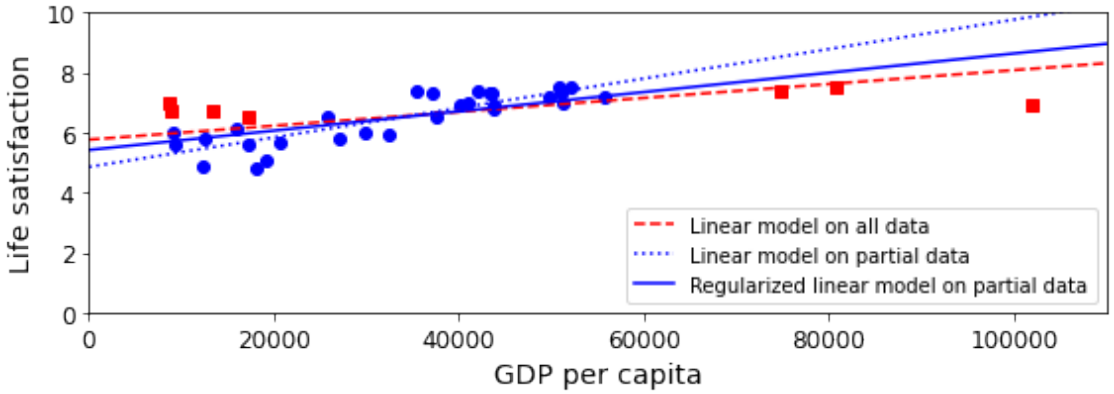
[46]:

Start	After	Subject Descriptor	…	Estimates
Country			…	
Botswana	2008.0	Gross domestic product per capita, current prices	…	
Kuwait	2014.0	Gross domestic product per capita, current prices	…	
Malawi	2011.0	Gross domestic product per capita, current prices	…	
New Zealand	2015.0	Gross domestic product per capita, current prices	…	
Norway	2015.0	Gross domestic product per capita, current prices	…	

```
[5 rows x 6 columns]
```

[47]:
```
# Example of regularized regression Ridge. Our model is simple, but Ridge␣
 ↪reduces the slope. It correct in this case, but
# it may not always be correct.
plt.figure(figsize=(8,3))

plt.xlabel("GDP per capita")
plt.ylabel('Life satisfaction')

plt.plot(list(sample_data["GDP per capita"]), list(sample_data["Life␣
 ↪satisfaction"]), "bo")
plt.plot(list(missing_data["GDP per capita"]), list(missing_data["Life␣
 ↪satisfaction"]), "rs")

X = np.linspace(0, 110000, 1000)
plt.plot(X, t0full + t1full * X, "r--", label="Linear model on all data")
plt.plot(X, t0 + t1*X, "b:", label="Linear model on partial data")

ridge = linear_model.Ridge(alpha=10**9.5)
Xsample = np.c_[sample_data["GDP per capita"]]
ysample = np.c_[sample_data["Life satisfaction"]]
ridge.fit(Xsample, ysample)
t0ridge, t1ridge = ridge.intercept_[0], ridge.coef_[0][0]
plt.plot(X, t0ridge + t1ridge * X, "b", label="Regularized linear model on␣
 ↪partial data")

plt.legend(loc="lower right")
plt.axis([0, 110000, 0, 10])
save_fig('ridge_model_plot')
plt.show()
```

```
Saving figure ridge_model_plot
```



[47]: