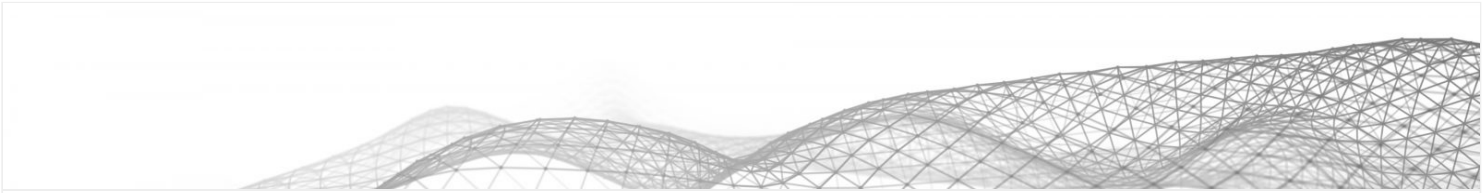


A BRIEF PROOF-OF-CONCEPT FOR KERNELIZED DECISION TREES

March 22, 2018 (https://numbersandcode.com/a-brief-proof-of-concept-for-kernelized-decision-trees)

Sarem (https://numbersandcode.com/author/sarem)

Applications (https://numbersandcode.com/category/applications), Classification (https://numbersandcode.com/category/classification), Decision Trees (https://numbersandcode.com/category/decision-trees), Machine Learning (https://numbersandcode.com/category/machine-learning), Python (https://numbersandcode.com/category/python)

+1 share share share

Search

RECENT POSTS

Generalized Additive Neural Networks (https://numbersandcode.com/generali additive-neural-networks)

Some experiments with Local Linear Regression. Part I – A primer on Local Linear Regression (https://numbersandcode.com/some-experiments-with-local-linear-regression-part-i-a-primer-on-local-linear-regression)

Introduction

In my last post (https://numbersandcode.com/how-decision-trees-can-learn-non-rectangular-decision-splits) I replaced the original features of a simulated 2-dimensional classification problem with the matrix of euclidean distances between all data-points. This meant that the distances to all points in the training set became the new features for the learner - formally speaking we performed the learning task on the Gram Matrix of the original features. This idea of using similarity measures among the training observations instead of the base variables is the core idea of successful Kernel Regression techniques like Support Vector Machines. However, the common approach of using Ridge Regression on the kernelized features was replaced by a simple Decision Tree.
Today, I want to see if this approach will also work on a real-world dataset but before that, let's have a short recap on Kernels in a Machine Learning context.

More simple time-series models – this time with Decision Trees (https://numbersandcode.com/more-simple-time-series-models-this-time-with-decision-trees)

Interpretable, scalable, non-linear – ca have it all? An approach with K-Means (https://numbersandcode.com/interpre scalable-non-linear-can-we-have-it-all-approach-with-k-means)

Non-Greedy MARS regression (https://numbersandcode.com/non-greedy-mars-regression)

RECENT COMMENTS

In on RuleFit on real-world data (https://numbersandcode.com/rulefit-real-world-data#comment-192)
Sarem on RuleFit on real-world data (https://numbersandcode.com/rulefit-real-world-data#comment-186)
Sarem on RuleFit for interpretable Machine Learning (https://numbersandcode.com/rulefit-interpretable-machine-learning#comment-185)
In on RuleFit for interpretable Machine Learning (https://numbersandcode.com/rulefit-interpretable-machine-learning#comment-184)
Min on RuleFit on real-world data (https://numbersandcode.com/rulefit-real-world-data#comment-183)

ARCHIVES

- December 2019 (https://numbersandcode.com/2019/12
- November 2019 (https://numbersandcode.com/2019/11
- February 2019 (https://numbersandcode.com/2019/02

How do Kernels work - a quick summary

As mentioned above, last time I used the euclidean distances to each point in the training data as a new feature instead of the original ones. Kernel can be seen as a generalization to distances and will use what is called a Kernel function

$$k(X, Y)$$

where X and Y are the feature vectors of single observations. To be a valid Kernel function, there are a few requirements that have to be fulfilled:

- $k(X, Y) = k(Y, X)$ (symmetry)
- The resulting $n \times n$ Gram Matrix (see G below) is positive semi-definite

The mathematical theory behind Kernels is much richer and you might want to dig in deeper on it, for example here (https://ocw.mit.edu/courses/sloan-school-of-management/15-097-prediction-machine-learning-and-statistics-spring-2012/lecture-notes/MIT15_097S12_lec13.pdf), here (https://www.microsoft.com/en-us/research/publication/a-tutorial-on-support-vector-machines-for-pattern-recognition/) or here (https://www.amazon.com/gp/product/0262194759/ref=as_li_tl?ie=UTF8&camp=1789&creative=9325&creativeASIN=0262194759&linkCode=as2&tag=numberscode-20&linkId=6d99b3d5dbad57ce566b2f869a80d044). The simplest kernel apart from degenerate ones is probably the Linear Kernel

$$k(X, Y) = X^T Y$$

which is just the dot-product of both vectors. Another valid kernel would be the Euclidean Distance that I used in the last post. For a good overview on different popular kernels and how to combine them, you can have a look at the Kernel Cookbook by David Duvenaud (https://www.cs.toronto.edu/~duvenaud/cookbook/). While he describes Kernels in the context of Bayesian Statistics (Gaussian Processes) most ideas carry over to the Frequentist domain as well.
Now suppose we have a a small training set of $n = 3$ observations and $k = 2$ features. A plain OLS model would look like this (omitting intercept and noise):

$$Y = X\beta \rightarrow \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \end{pmatrix} \bullet \begin{pmatrix} \beta_1 \\ \beta_2 \end{pmatrix}$$

where ● denotes plain matrix multiplication and the application on test/unseen data is done as usual.
Switching to the kernelized version, we won't use the original features in X anymore but will switch to the so-called Gram matrix G:

$$G = \begin{pmatrix} k(x_{1.}, x_{1.}) & k(x_{1.}, x_{2.}) & k(x_{1.}, x_{3.}) \\ k(x_{2.}, x_{1.}) & k(x_{2.}, x_{2.}) & k(x_{2.}, x_{3.}) \\ k(x_{3.}, x_{1.}) & k(x_{3.}, x_{2.}) & k(x_{3.}, x_{3.}) \end{pmatrix}$$

which results in a transformed regression problem:

$$Y = G\alpha \rightarrow \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} k(x_{1.}, x_{1.}) & k(x_{1.}, x_{2.}) & k(x_{1.}, x_{3.}) \\ k(x_{2.}, x_{1.}) & k(x_{2.}, x_{2.}) & k(x_{2.}, x_{3.}) \\ k(x_{3.}, x_{1.}) & k(x_{3.}, x_{2.}) & k(x_{3.}, x_{3.}) \end{pmatrix} \bullet \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{pmatrix}$$

For two new test instances $x_{4.}, x_{5.}$, we can now predict the y-values through

- October 2018 (https://numbersandcode.com/2018/10
- September 2018 (https://numbersandcode.com/2018/09
- August 2018 (https://numbersandcode.com/2018/08
- July 2018 (https://numbersandcode.com/2018/07
- April 2018 (https://numbersandcode.com/2018/04
- March 2018 (https://numbersandcode.com/2018/03

CATEGORIES

- Algorithms (https://numbersandcode.com/categor
- Applications (https://numbersandcode.com/categor
- Classification (https://numbersandcode.com/categor
- Decision Trees (https://numbersandcode.com/categor trees)
- Forecasting (https://numbersandcode.com/categor
- Gradient Boosting (https://numbersandcode.com/categor boosting)
- Hilbert Spaces (https://numbersandcode.com/categor spaces)
- Interpretable Machine Learning (https://numbersandcode.com/categor machine-learning)
- Julia (https://numbersandcode.com/categor
- Linear Model (https://numbersandcode.com/categor model)

$$\begin{pmatrix} y_4 \\ y_5 \end{pmatrix} = \begin{pmatrix} k(x_{4.}, x_{1.}) & k(x_{4.}, x_{2.}) & k(x_{4.}, x_{3.}) \\ k(x_{5.}, x_{1.}) & k(x_{5.}, x_{2.}) & k(x_{5.}, x_{3.}) \end{pmatrix} \bullet \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{pmatrix}$$

This allows us to non-linearly transform the original features in X and hopefully gain a predictive advantage in the new kernel space. Of course this introduction only scratches the surface of Kernel methods but the rest of this post doesn't need more theory to be understood so I won't go further here.
The only thing you should also be aware of is that in case of a classification problem we would need a link function to map the output from $\mathbb{R}$ to a binary response or probability respectively. Common link functions are the sign-function $sign : \mathbb{R} \to \{-1; 1\}$ in Support Vector Classification or the logit-function $logit : \mathbb{R} \to (0, 1)$ which would be used in Kernel Logistic Regression.

Generalizing Kernels beyond linearity

In most academic literature the relationship between Y and G will be parametrized in a linear fashion like the one above and estimated via some penalized regression technique. However, as stated before, Decision Trees might also take advantage of the transformed features, so a generalization in the form of

$$Y = f(G)$$

looks reasonable. Naturally, this brings almost every supervised Machine Learning algorithm into play but here I want to focus solely on plain Decision Trees.

Assessing the performance of Kernelized Decision Trees

To get a picture of the accuracy of the proposed approach I carried out a small train-test-split Monte Carlo simulation (2500 repetitions) on the Iris dataset, classifying 'Species'. The original data and the kernelized features of 3 different kernels

- $Euclidean\ Distance : k(X, Y) = ||X - Y||_2$
- $Laplacian\ Kernel : k(X, Y) = exp(-\gamma||X - Y||_1)$
  $||\cdot||_1 := Manhattan\ Distance$
- $RBF\ Kernel : k(X, Y) = exp(-\gamma||X - Y||_2)$

were used as the base features for the Decision Tree to learn on. No hyperparameter-tuning was done to compare out-of-box performance - adjusting options like maximum tree depth or adjusting γ in the Kernels might improve performance further. The performance metric was simply Accuracy.

› Machine Learning (https://numbersandcode.com/categor learning)
› MARS (https://numbersandcode.com/categor
› Naive Bayes (https://numbersandcode.com/categor bayes)
› Neural Networks (https://numbersandcode.com/categor networks)
› Python (https://numbersandcode.com/categor
› Random Forest (https://numbersandcode.com/categor forest)
› Time Series (https://numbersandcode.com/categor series)
› Uncategorized (https://numbersandcode.com/categor

OTHER

› About (https://numbersandcode.com/about)
› Books (https://numbersandcode.com/books)
› Disclaimer (https://numbersandcode.com/disclaim
› Learn Data Science (https://numbersandcode.com/learn-data-science)
  › Introduction (https://numbersandcode.com/learn-data-science/introduction)

In [1]:
```
import pandas as pd

df = pd.read_csv("Iris.csv").drop("Id",1)
df.head()
```

Out[1]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	S
0	5.1	3.5	1.4	0.2	Ir s
1	4.9	3.0	1.4	0.2	Ir s
2	4.7	3.2	1.3	0.2	Ir s
3	4.6	3.1	1.5	0.2	Ir s
4	5.0	3.6	1.4	0.2	Ir s

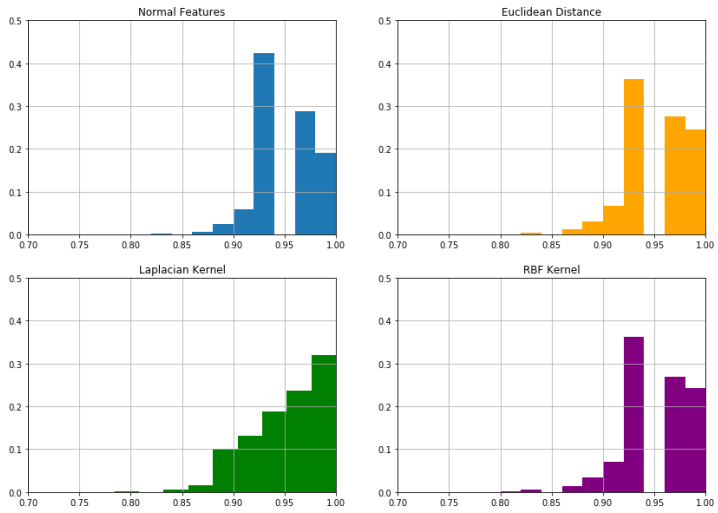
In [2]:
```
import numpy as np
from scipy.spatial.distance import pdist, squareform
from sklearn.metrics.pairwise import rbf_kernel, laplac
ian_kernel

X = df.drop("Species",1)
y = pd.get_dummies(df["Species"])

#the gram matrices can be calculated in advance and not
need to be recalculated during the simulation
gram_dist = squareform(pdist(X))
gram_laplace = laplacian_kernel(X)
gram_rbf = rbf_kernel(X)


acc_simulation = pd.DataFrame(np.nan, index = range(250
0), columns =[
    "Normal", "Euclidean", "Laplace","RBF"])
```

In [3]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import numpy as np


np.random.seed(123)
for rep in range(2500):

    arange = np.arange(len(df))
    samples_train = np.random.choice(arange,100, replac
e=False)
    samples_test = np.delete(arange,samples_train)


    #this slices the gram matrices according to train a
nd test samples
    dist_train = gram_dist[samples_train,:][:,samples_t
rain]
    dist_test = gram_dist[samples_test,:][:,samples_tra
in]

    laplace_train = gram_laplace[samples_train,:][:,sam
ples_train]
    laplace_test = gram_laplace[samples_test,:][:,sampl
es_train]

    rbf_train = gram_rbf[samples_train,:][:,samples_tra
in]
    rbf_test = gram_rbf[samples_test,:][:,samples_train
]

    X_train = X.iloc[samples_train,:]
    X_test = X.iloc[samples_test,:]


    y_train = y.iloc[samples_train,:]
    y_test = y.iloc[samples_test,:]



    tree_norm = DecisionTreeClassifier().fit(X_train,y_
train)
    tree_dist = DecisionTreeClassifier().fit(dist_train
,y_train)
    tree_laplace = DecisionTreeClassifier().fit(laplace
_train,y_train)
    tree_rbf = DecisionTreeClassifier().fit(rbf_train,y
_train)


    pred_norm = tree_norm.predict(X_test)
    pred_dist = tree_dist.predict(dist_test)
    pred_laplace = tree_laplace.predict(laplace_test)
    pred_rbf = tree_rbf.predict(rbf_test)


    acc_simulation.loc[rep,"Normal"] = accuracy_score(y
_test,pred_norm)
    acc_simulation.loc[rep,"Euclidean"] = accuracy_scor
e(y_test,pred_dist)
    acc_simulation.loc[rep,"Laplace"] = accuracy_score(
y_test,pred_laplace)
    acc_simulation.loc[rep,"RBF"] = accuracy_score(y_te
st,pred_rbf)
```

Results

In [69]:
```
#Comparing the accuracies through histograms
import matplotlib.pyplot as plt

fig, ax = plt.subplots(nrows = 2, ncols = 2)
fig.set_size_inches(14,10)


ax[0][0].hist(acc_simulation.iloc[:,0],bins=10,
              weights=np.zeros_like(acc_simulation.iloc
[:,0])+1./acc_simulation.iloc[:,0].size)
ax[0][0].grid()
ax[0][0].set_ylim([0,0.5])
ax[0][0].set_xlim([.7,1.])
ax[0][0].set_title("Normal Features")


ax[0][1].hist(acc_simulation.iloc[:,1],bins=10,color="o
range",
              weights=np.zeros_like(acc_simulation.iloc
[:,1])+1./acc_simulation.iloc[:,1].size)
ax[0][1].grid()
ax[0][1].set_ylim([0,0.5])
ax[0][1].set_xlim([.7,1.])
ax[0][1].set_title("Euclidean Distance")


ax[1][0].hist(acc_simulation.iloc[:,2],bins=10,color="g
reen",
             weights=np.zeros_like(acc_simulation.iloc
[:,2])+1./acc_simulation.iloc[:,2].size)
ax[1][0].grid()
ax[1][0].set_ylim([0,0.5])
ax[1][0].set_xlim([.7,1.])
ax[1][0].set_title("Laplacian Kernel")


ax[1][1].hist(acc_simulation.iloc[:,3],bins=10,color="p
urple",
              weights=np.zeros_like(acc_simulation.iloc
[:,3])+1./acc_simulation.iloc[:,3].size)
ax[1][1].grid()
ax[1][1].set_ylim([0,0.5])
ax[1][1].set_xlim([.7,1.])
ax[1][1].set_title("RBF Kernel")
```

Out[69]: <matplotlib.text.Text at 0x7f824efa23d0>



In [5]:
```
#summary statistics
print "Mean Accuracy:"
print acc_simulation.mean()
print "\nStandard Deviation Accuracy:"
print acc_simulation.std()


print "\nMedian Accuracy:"
print acc_simulation.quantile(0.5)
print "\n5% Quantile Accuracy:"
print acc_simulation.quantile(0.05)
print "\n95% Quantile Accuracy:"
print acc_simulation.quantile(0.95)
```

```
Mean Accuracy:
Normal       0.946064
Euclidean    0.947712
Laplace      0.949568
RBF          0.946808
dtype: float64

Standard Deviation Accuracy:
Normal       0.028370
Euclidean    0.031304
Laplace      0.033777
RBF          0.031980
dtype: float64

Median Accuracy:
Normal       0.94
Euclidean    0.96
Laplace      0.96
RBF          0.96
Name: 0.5, dtype: float64

5% Quantile Accuracy:
Normal       0.90
Euclidean    0.90
Laplace      0.88
RBF          0.88
Name: 0.05, dtype: float64

95% Quantile Accuracy:
Normal       0.98
Euclidean    1.00
Laplace      1.00
RBF          1.00
Name: 0.95, dtype: float64
```

In [70]:
```
#depth of trees
print "Tree depth - Normal Features:"
print tree_norm.tree_.max_depth

print "\nTree depth - Kernelized Features"
print tree_dist.tree_.max_depth
print tree_laplace.tree_.max_depth
print tree_rbf.tree_.max_depth
```

```
Tree depth - Normal Features:
5

Tree depth - Kernelized Features
3
3
3
```

The Kernelized Trees look competitive - additionally, they appear to be able to achieve a slightly higher accuracy at the cost of an increased standard deviation/variance. Especially the Laplacian Kernel seems to be highly performant on the dataset. The main advantage that might arise from Kernelized Decision Trees is being able to keep up with the standard version but with a more shallow structure. To reduce the variance of the predictions it could also be interesting to consider Tree Ensembles in a Kernel context - this paper (https://link.springer.com/article/10.1007/s10489-014-0575-4) investigates the performance of that approach if you want to dig in deeper.
One should also be aware of the computational disadvantage of transforming the features through Kernels as both the computation and storage of the Gram matrix can become infeasible for large datasets. As always, a single dataset is not representative to derive a final conclusion but only shows how the proposed technique might work. Nevertheless, the idea looks reasonable and sound to me and using different kernels and other non-tree algorithms might be worthwhile to try out as well.

< PREVIOUS (HTTPS://NUMBERSANDCODE.COM/HOW-DECISION-TREES-CAN-LEARN-TOO-MUCH)

NEXT > (HTTPS://NUMBERSANDCODE.COM/INTERPRETABLE-SEMI-PARAMETRIC-NEURAL-NETWORK)

Leave Comment

YOUR MESSAGE…

YOUR NAME (REQUIRED

YOUR EMAIL (REQUIRED

YOUR WEBSITE…

SUBMIT NOW

Developed by Think Up Themes Ltd (//www.thinkupthemes.com/). Powered by WordPress (//www.wordpress.org/).