pawel.kasprowski@polsl.pl

Artificial Intelligence

Regression

Paweł Kasprowski, PhD, DSc.

Regression

- The model does not search for one of N classes but for a value
  - E.g. price of the house
- Examples:
  - Linear Regression
  - Support Vector Regression
  - Decision Tree Regression
  - Neural Network Regression

Evaluation of regression

- Mean Absolute Error
- Mean Squared Error
  - penalizes predictions differing greatly (outliers)
- Root Mean Squared Error
  - comparable to real values
- Coefficient of Determination (R^2)
  - value 0-1
  - negative value if worse than random

Example: wine quality dataset

- Every wine has 11 numerical features
  - 1 - fixed acidity
  - 2 - volatile acidity
  - 3 - citric acid
  - 4 - residual sugar
  - 5 - chlorides
  - 6 - free sulfur dioxide
  - 7 - total sulfur dioxide
  - 8 - density
  - 9 - pH
  - 10 - sulphates
  - 11 - alcohol

- The result: wine quality in 0-10 scale
- Dataset: 4898 wines

Wine dataset analysis

wine.ipynb

- Simple classification
- Train-test split
- Measures calculation – confusion matrix
- Problem:
  - quality is an ordinal value not just a class
  - results should be calculated as a distance from the correct one
    - it is not bad when wine with quality 3 is classified as 4
    - it is a big problem when it is classified as 9

Regression measures

- Errors:

```
from sklearn.metrics import mean_squared_error, r2_score
print('MSE=',mean_squared_error(testLabels, predictedLabels))
print('R2=',r2_score(testLabels, predictedLabels))
```

- Results:
  - MSE = 0.99
  - R2 = -0.27

Linear Regression

- Using Linear Regression:

```
from sklearn import linear_model
model = linear_model.LinearRegression()
model.fit(trainSamples, trainLabels)
predictedLabels = model.predict(testSamples)
```

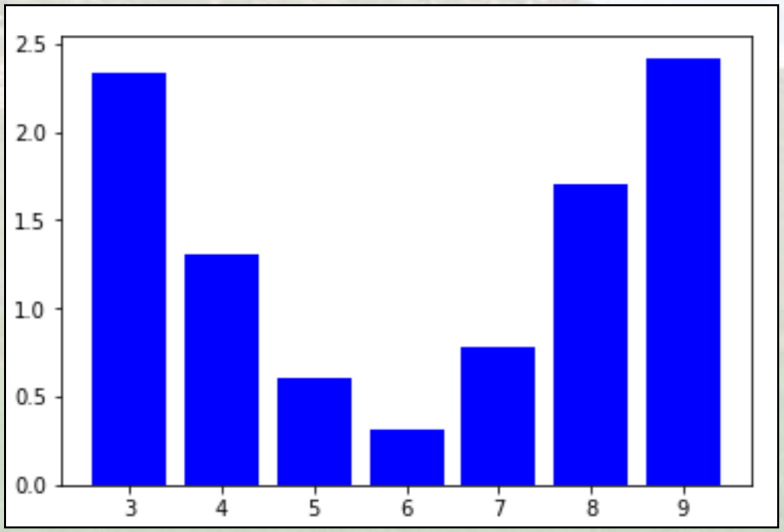
- The result for each sample is a float!
- Errors:
  - MSE = 0.53
  - R2 = 0.31

Errors per class estimation

- Mean errors per class

```
errors = np.abs(modelResults-testLabels)
for i in range(3,10):
    print('class',i,' avg error=',errors[np.where(testLabels==i)].mean())
```

- Results
  - class 3 avg error= 2.33
  - class 4 avg error= 1.31
  - class 5 avg error= 0.61
  - class 6 avg error= 0.31
  - class 7 avg error= 0.78
  - class 8 avg error= 1.71
  - class 9 avg error= 2.42



Problem: inbalanced classes

- Some qualities are rare, some are common
- Solution: Calculate weight for each class:

```
from sklearn.utils import class_weight
class_weights = class_weight.compute_class_weight('balanced',
                                    classes=np.unique(trainLabels),y=trainLabels)
weights = np.ones([len(trainLabels)]) # initialize all weights to 1
for i, label in enumerate(trainLabels):
    weights[i] *= class_weights[int(label-3)] # the first label is 3!
```

- Using weights:

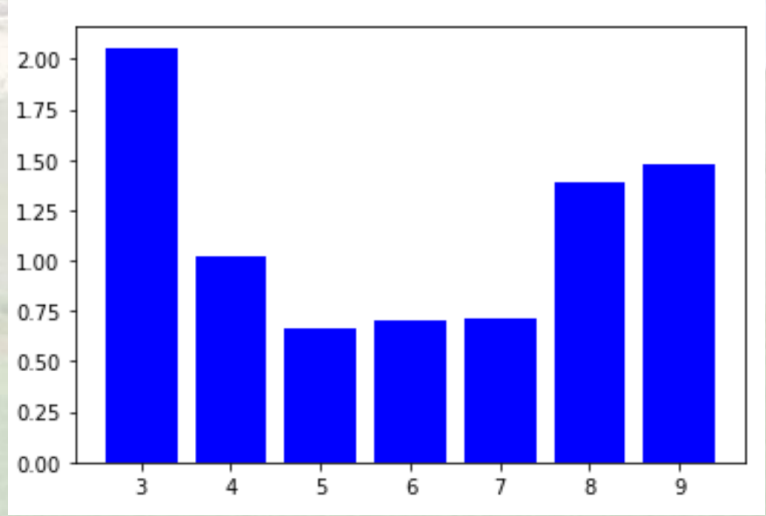
```
model.fit(trainSamples, trainLabels, sample_weight=weights)
```

Regression with weights

- Mean errors per class

```
errors = np.abs(modelResults-testLabels)
for i in range(3,10):
    print('class',i,' avg error=',errors[np.where(testLabels==i)].mean())
```

- Results
  - class 3 avg error= 2.06
  - class 4 avg error= 1.02
  - class 5 avg error= 0.66
  - class 6 avg error= 0.70
  - class 7 avg error= 0.72
  - class 8 avg error= 1.39
  - class 9 avg error= 1.48



Results

- Distribution is more equal for the weighted model
- But the error is higher (both MSE and R2)
- Question:
  - what do we really need?
  - what is our objective?
- Possibilities:
  - minimize absolute error
  - minimize MSE/RMSE
  - minimize error for high quality wines
  - ...

Feature selection

- Not every feature is valuable
  - adds some knowlegde about a class
  - e.g. bottle's color is *probably* not significant for wine quality
- Not important features may spoil classification!
- There are plethora of algoritms that aim at selecting only the relevant features from the dataset
  - topic for a new lecture (or even a new subject)
- Sklearn has a very convenient class to solve the problem

SelectKBest

- Code:

```
from sklearn.feature_selection.univariate_selection import
    SelectKBest
newSamples = SelectKBest(<algorithm>, k=<number>)
    .fit_transform(samples, labels)
```

- Available algorithms:

(see: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html

f_classif, mutual_info_classif, chi2, f_regression, mutual_info_regression, SelectPercentile, SelectFpr, SelectFdr, SelectFwe, GenericUnivariateSelect

Example

- Select 5 best features using f_regression algorithm:

```
from sklearn.feature_selection.univariate_selection import SelectKBest
print("Samples before",samples.shape)
newSamples = SelectKBest(sklearn.feature_selection.f_regression,k=5)
    .fit_transform(samples, labels)
print("Samples after", newSamples.shape)
samples = newSamples
```

- Output:
  - Samples before (4898, 10)
  - Samples after (4898, 5)
- Task for you: check if it works better!

Transforming two datasets

- If we have two datasets, e.g.:
  - trainingSamples
  - testSamples
- We must use ONE transformer for both

```
skb = SelectKBest(sklearn.feature_selection.f_regression,k=100)
skb.fit(trainSamples, trainLabels)
trainSamples = skb.transform(trainSamples)
testSamples = skb.transform(testSamples)
```

Summary

- There are many different classification algorithms
- There are many ways to tune these algorithms
- There are many measures to estimate the quality of classification/regression
- So what is so special about the deep learning methods?
  - Wait for the following sections!