Python programming and data analysis

Lecture 9

Introduction to Machine Learning 2

Robert Szmurło

e-mail: robert.szmurlo@ee.pw.edu.pl 2018Z

Lecture outline

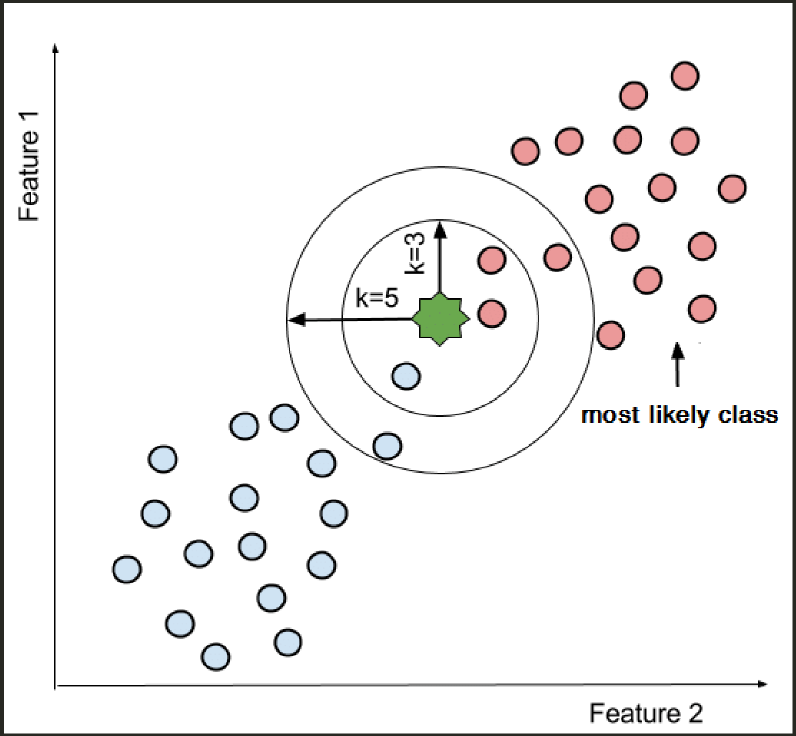
1. KNN - K Nearest Neighbours
2. SVM - support vector machine
3. PCA - principal component analysis

Next lecture

- K-Means Clustering - unlabelled classification
- Natural Language Processing, Big Data and Spark with Python, AWS Account setup?

KNN - k-nearest neighbours

Find n-nearest neighbours of a new point (future set) and approximate the output value

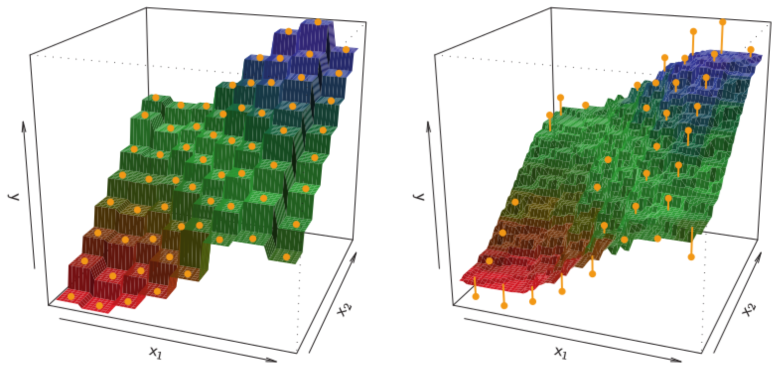


KNN for regression - Motivation

Parametric methods (Linear Regression) do have a disadvantage: by construction, they make strong assumptions about the form of $f(X)$.

In contrast, non-parametric methods do not explicitly assume a parametric form for $f(X)$, and thereby provide an alternative and more flexible approach for performing regression.

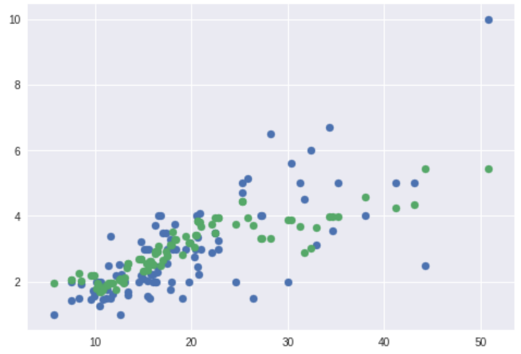
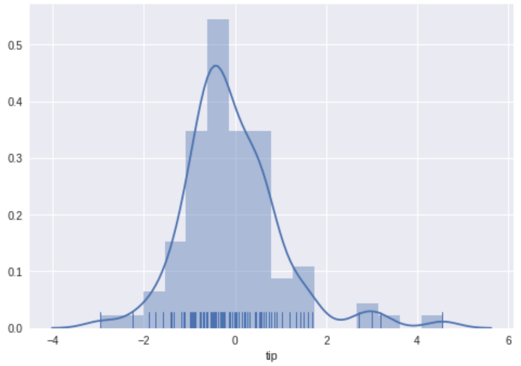
One of the simplest and best-known non-parametric methods is the *K-nearest neighbours* regression (KNN regression)

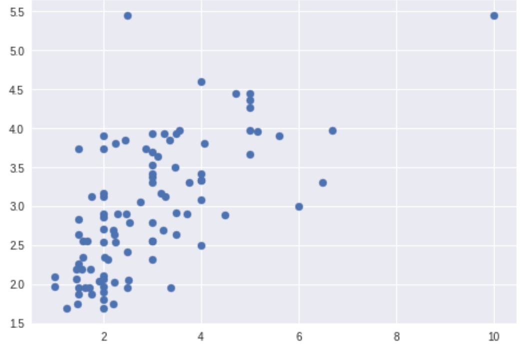


Left: K = 1 results in a rough step function fit. Right: K = 9 produces a much smoother fit

KNN regression - simple test program

```
# imports: import seaborn as sns, import pandas as pd, from numpy.random import randn, from IPython.display import display, import matplotl

tips = sns.load_dataset('tips')
tips.columns

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(tips[['total_bill']], tips['tip'], test_size=0.4, random_state=200)

from sklearn.neighbors import KNeighborsRegressor
knnr = KNeighborsRegressor()
knnr.fit(X_train, y_train)
pred = knnr.predict(X_test)

plt.scatter(X_test['total_bill'], y_test)
plt.scatter(X_test['total_bill'], pred)

sns.distplot(y_test - pred,rug=True)

plt.scatter(y_test,pred)
```



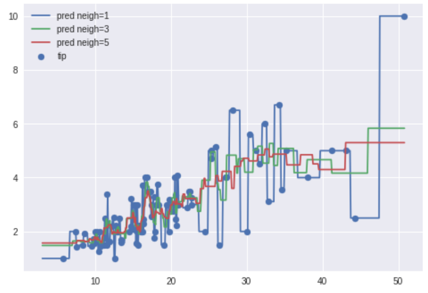
KNN - parameters evaluation

- `n_neighbors` - number of neighbours
- `weights` - should the distance be a weight or the weights should be uniform
- `p` - the distance calculation measure (p=1 absolute distance, p=2 euclidean)

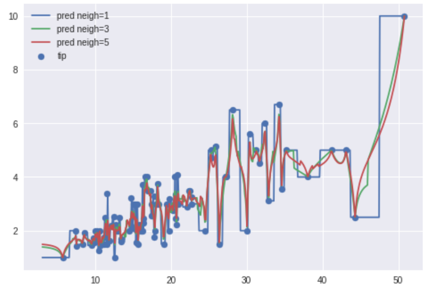
```
knnr = KNeighborsRegressor(n_neighbors=5,weights='uniform', p=2)
```

Simple testing program:

```
tb_min = tips['total_bill'].min()
tb_max = tips['total_bill'].max()
X_n = pd.DataFrame(np.linspace(tb_min, tb_max, 500), columns=['tb'])
X_n.head()
X_t = X_test[['total_bill']]

plt.scatter(X_test['total_bill'], y_test)
for i in range(1,6,2):
  knnr = KNeighborsRegressor(n_neighbors=i,weights='distance', p=2)
  #knnr = KNeighborsRegressor(n_neighbors=i,weights='uniform')
  knnr.fit(X_t, y_test)
  pred_new = knnr.predict(X_n)
  plt.plot(X_n, pred_new,label=f'pred neigh={i}')
plt.legend()
```



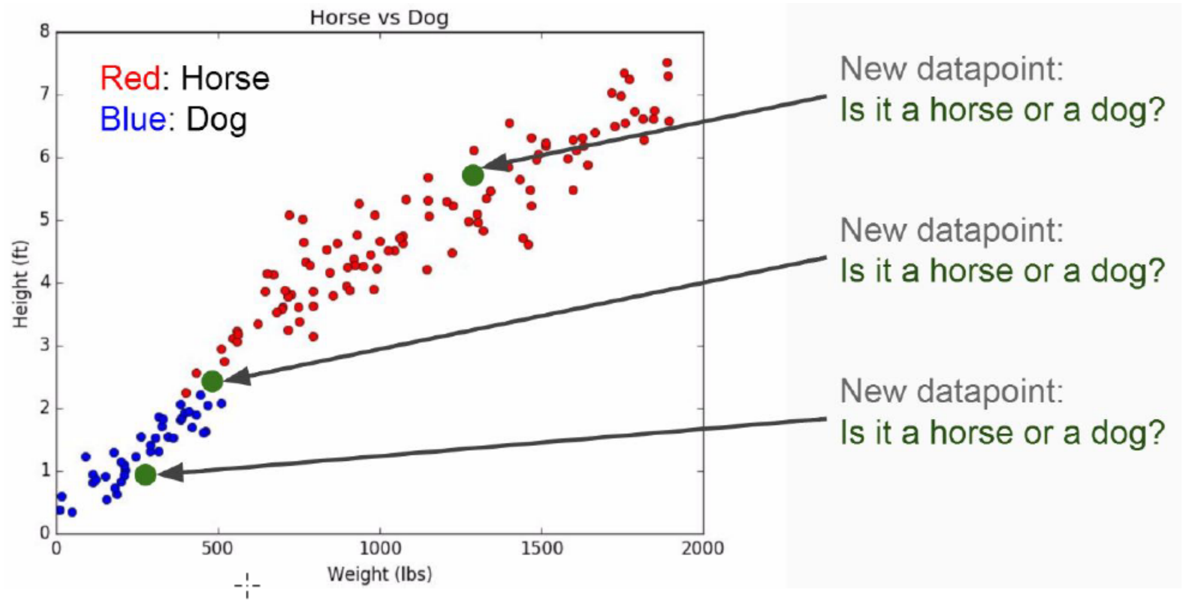
weights='uniform'



weights='distance'

KNN for classification problems 1

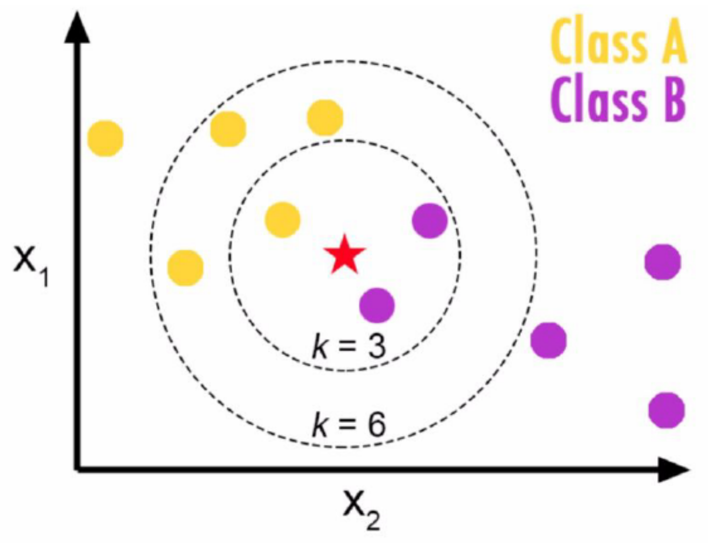
- Imagine we had some imaginary data on Dogs and Horses, with heights and weights.
- In our classification problem we want to predict if a new 'animal' is a dog or a horse on the basis of its weight and height.
- We're just trying to **classify** this new data point.



- For example this top point in green would probably be a horse since all the points around it are red horse points.

KNN for classification problems 2

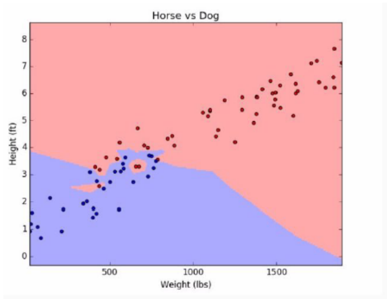
1. You calculate the distance from X to all the points in your data X indicating that particular new data point.
2. Then you sort the points near data by increasing distance from X.
3. Then you predict the majority label of the K. K being a number.
4. Closest points choosing a K will affect what class a new point is assigned to.
5. Training data is just yellow points in class A and purple points in class B.
6. This red star indicates a few points I want to predict whether this point belongs to class A or class B.
7. If I choose K equals 3 then I look at the three nearest neighbors to this new point.
8. In this case I have two purple ones and one yellow one are a class a so with equal to 3.
9. I will predict that this new red star point belongs to class B - purple.
10. However if I set K equal to 6 I will actually have a majority of yellow or class 8 points meaning if
11. I had KCl 6 I would predict that the new point belongs to a class A choosing a K will affect what class a new point is assigned to.
12. So if we look back on our horse vs dog we can here see the plaudit effects of various k values by choose cake was to 1 and then to actually pick up a lot of noise.
13. But as I go larger and larger k values you'll notice I smooth out and create more of a bias in my model or I get a cleaner cut off at the cost of mislabeling some points.



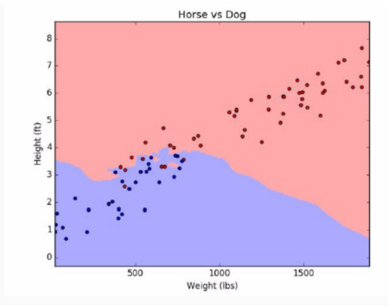
Choosing K

Choosing a K will affect what class a new point is assigned to:

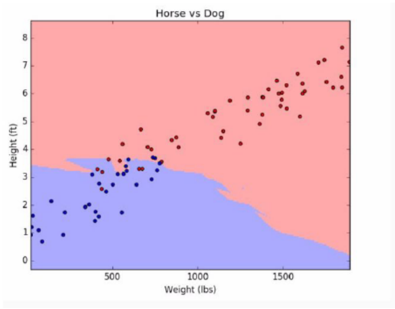
k=1



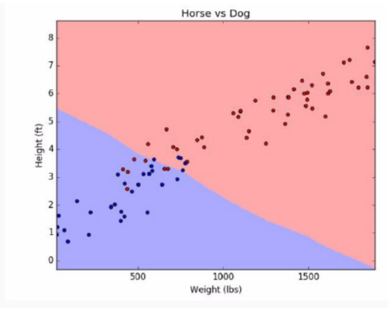
k=5



k=10



k=50



KNN

Advantages

- Very simple
- Training is trivial
- Works with any number of classes
- Easy to add more data
- Few parameters
    - K
    - Distance Metric (uniform, distance to point based)

Disadvantages

- High Prediction Cost (worse for large data sets)
- Not good with high dimensional data
- Categorical Features don’t work well

Hands on project

- A common interview task for a data scientist position is to be given anonymized data and attempt to classify it without knowing the full context behind the data.
- A lot of times these are known as take home tasks where you're given a data set but you don't actually get the knowledge of what each column represents.
- We're going to simulate a similar scenario by giving you some quote unquote classified data where what the columns represent isn't known.
- But you're still going to have to try to use K and N to classify the data in two different classifications

Hands on - data

```
df = pd.read_csv('KNN_Project_Data.csv')
df.head()
```

- We have a bunch of data but we just have a target class - column 1 or 0.
- You just know that you need to use these features that are unknown to you as far as what they actually represent in order to predict a target class 1 or 0 because the K and classifier predicts the class of a given test observation by identifying the observations that are nearest to it.
- The scale of the variable actually matters a lot and any variables that are on a large scale will have a much larger effect on the distance between observations and because of this when you're using K-nearest neighbors to say fire which going to want to do is try to standardize everything to the same scale.

Standardize the Variables

Because the KNN classifier predicts the class of a given test observation by identifying the observations that are nearest to it, the scale of the variables matters. Any variables that are on a large scale will have a much larger effect on the distance between the observations, and hence on the KNN classifier, than variables that are on a small scale.

```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(df.drop('TARGET CLASS',axis=1))
scaled_features = scaler.transform(df.drop('TARGET CLASS',axis=1))
df_feat = pd.DataFrame(scaled_features,columns=df.columns[:-1])
df_feat.head()
```

Train Test Split

```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(scaled_features,df['TARGET CLASS'], test_size=0.30)
```

Using KNN

We are trying to come up with a model to predict whether someone will TARGET CLASS or not. We'll start with k=1.

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

Predictions and Evaluations

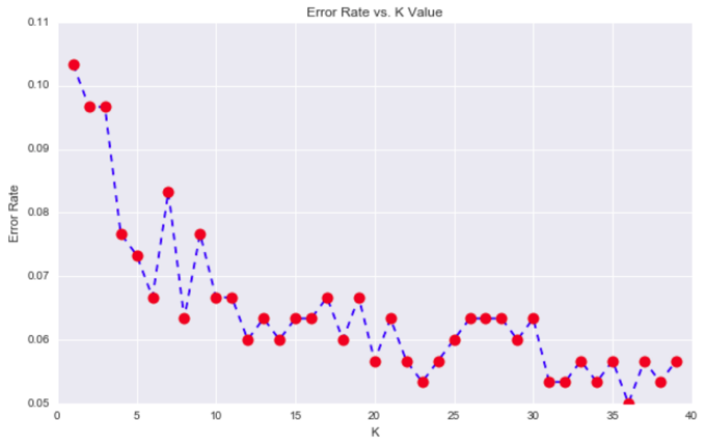
Let's evaluate our KNN model!

```
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

Choosing a K Value

We shall use the elbow method to pick a good K Value:

```
error_rate = []

# Will take some time
for i in range(1,40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))

plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```



Here we can see that that after around K>23 the error rate just tends to hover around 0.06-0.05 Let's retrain the model with that and check the classification report!

Comparison to original K=1

Let we check if we were able to squeeze some more performance out of our model by tuning to a better K value.

```
knn = KNeighborsClassifier(n_neighbors=1)

knn.fit(X_train,y_train)
pred = knn.predict(X_test)

print('WITH K=1')
print('\n')
print(confusion_matrix(y_test,pred))
print('\n')
print(classification_report(y_test,pred))
```

```
WITH K=1


[[125  18]
 [ 13 144]]


             precision    recall  f1-score   support

          0       0.91      0.87      0.89       143
          1       0.89      0.92      0.90       157

avg / total       0.90      0.90      0.90       300
```

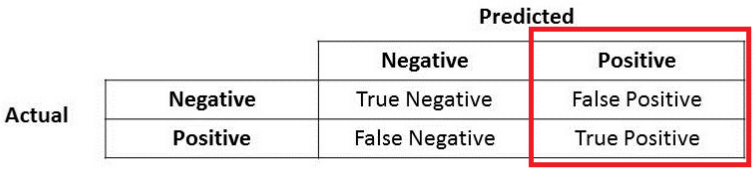
```
knn = KNeighborsClassifier(n_neighbors=23)

knn.fit(X_train,y_train)
pred = knn.predict(X_test)

print('WITH K=23')
print('\n')
print(confusion_matrix(y_test,pred))
print('\n')
print(classification_report(y_test,pred))
```

```
WITH K=23


[[132  11]
 [  5 152]]


             precision    recall  f1-score   support

          0       0.96      0.92      0.94       143
          1       0.93      0.97      0.95       157

avg / total       0.95      0.95      0.95       300
```

Understanding classification report

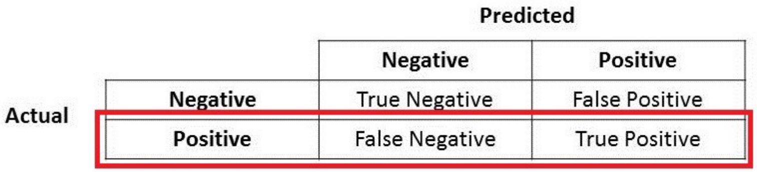
```
[[132  11]
 [  5 152]]

             precision    recall  f1-score   support

          0       0.96      0.92      0.94       143
          1       0.93      0.97      0.95       157

avg / total       0.95      0.95      0.95       300
```

- Precision: $\frac{\text{true positive}}{\text{true positive} + \text{false positive}}$
  Precision talks about how precise/accurate your model is out of those predicted positive, how many of them are actual positive.



		Predicted	
		Negative	Positive
Actual	Negative	True Negative	False Positive
	Positive	False Negative	True Positive

- Recall: $\frac{\text{true positive}}{\text{true positive} + \text{false negative}}$
  So Recall actually calculates how many of the Actual Positives our model capture through labeling it as Positive (True Positive). Applying the same understanding, we know that Recall shall be the model metric we use to select our best model when there is a high cost associated with False Negative.



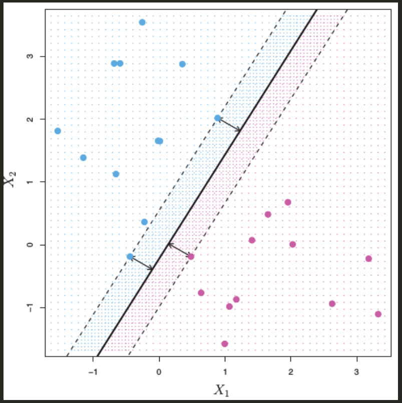
		Predicted	
		Negative	Positive
Actual	Negative	True Negative	False Positive
	Positive	False Negative	True Positive

F1 Score

$$\mathrm{F1} = 2 \times \frac{Precision \cdot Recall}{Precision + Recall}$$

F1 Score is needed when you want to seek a balance between Precision and Recall. Right…so what is the difference between F1 Score and Accuracy then? The accuracy can be largely contributed by a large number of True Negatives which in most business circumstances, we do not focus on much whereas False Negative and False Positive usually has business costs (tangible & intangible) thus F1 Score might be a better measure to use if we need to seek a balance between Precision and Recall AND there is an uneven class distribution (large number of Actual Negatives).

Support Vector Machines



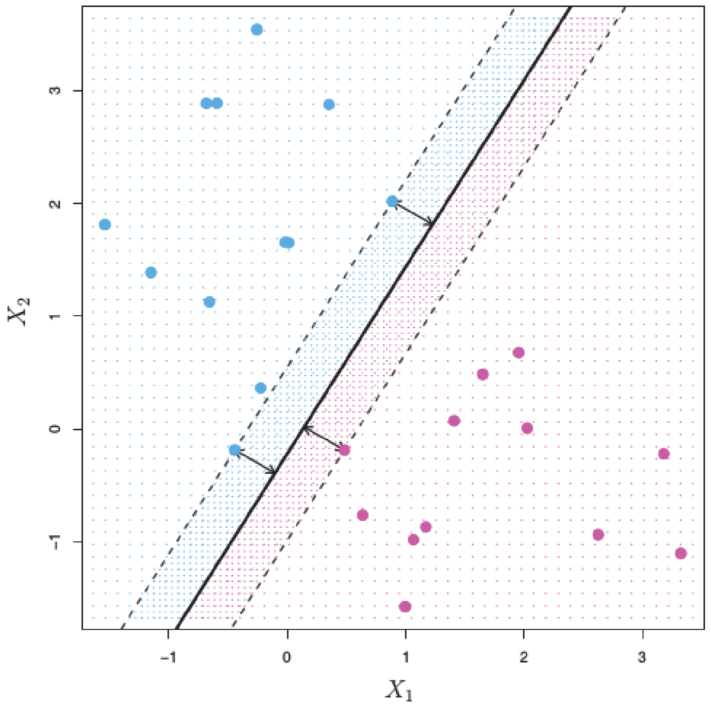
Support Vector Machines

We choose a hiperplane of order $p = n - 1$, where n is the number of features which separates two classes of observations given by $y_i \in 1, -1$. Thus we want to find coefficients β_j which will satisfy both:

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} > 0 \text{ if } y_i = 1$$

and

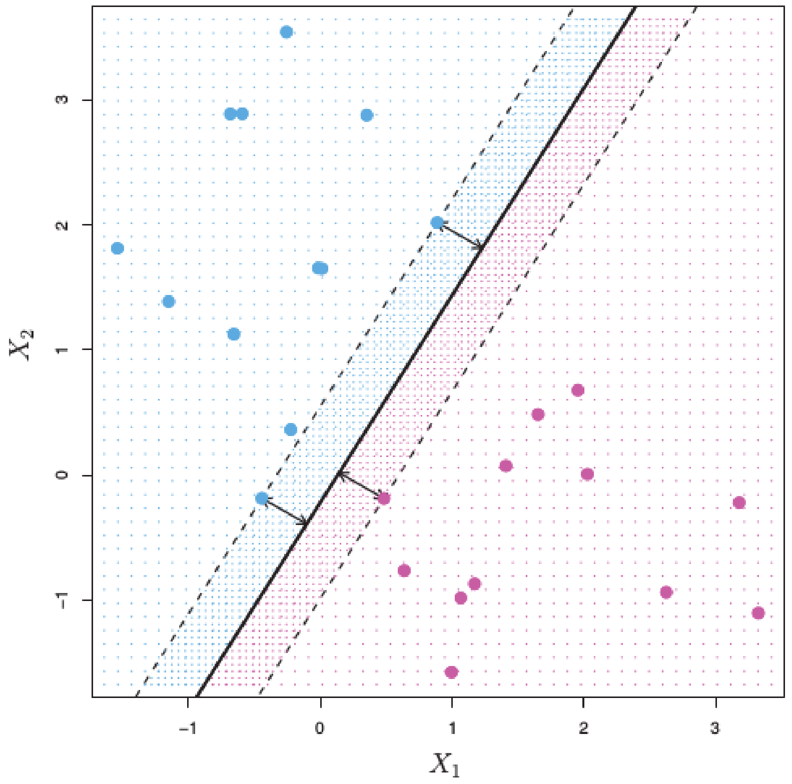
$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} < 0 \text{ if } y_i = -1$$



Support Vector Machines cont.

Support vector machines are intended for the binary classification setting in which there are two classes.

The support vector machine is a generalization of a simple and intuitive classifier called the *maximal margin classifier*.



Maximal margin classifier

Though it is elegant and simple, this classifier unfortunately cannot be applied to most data sets, since it requires that the classes be separable by a linear boundary.

$$\underset{\beta_0,\beta_1,\ldots,\beta_p,M}{\text{maximize}} \quad M$$

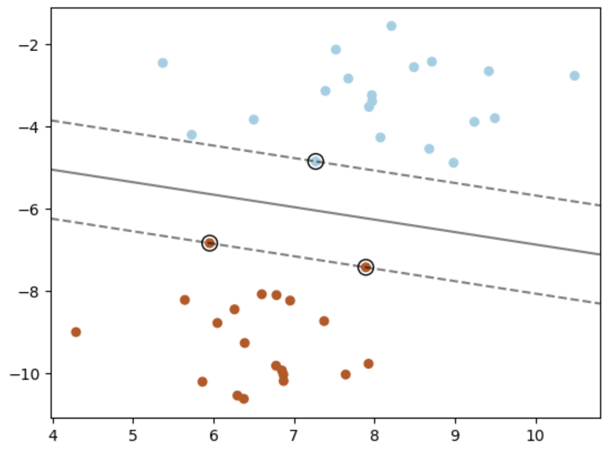
$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M \quad \forall\, i = 1, \ldots, n.$$

The constraints ensure that each observation is on the correct side of the hyperplane and at least a distance M from the hyperplane. Hence, M represents the margin of our hyperplane, and the optimization problem chooses $\beta_0, \beta_1, \ldots, \beta_p$ to maximize M.

SVC - Support vector classifier

A support vector machine constructs a hyper-plane or set of hyper-planes in a high or infinite dimensional space, which can be used for classification, regression or other tasks. Intuitively, a good separation is achieved by the hyper-plane that has the largest distance to the nearest training data points of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier.



SVC - Mathematical formulation

Given training vectors $x_i \in \mathbb{R}^p$, i=1,..., n, in two classes, and a vector $y \in \{1, -1\}^n$, SVC solves the following primal problem:

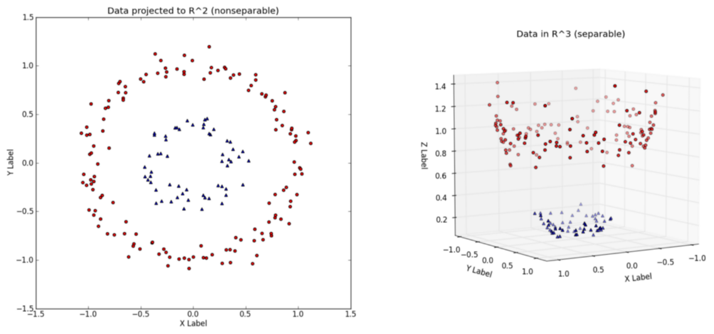
$$\begin{aligned} \min_{w,b,\zeta} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \zeta_i \\ \text{subject to } y_i(w^T \phi(x_i) + b) \geq 1 - \zeta_i, \\ \zeta_i \geq 0, i = 1, \ldots, n \end{aligned}$$

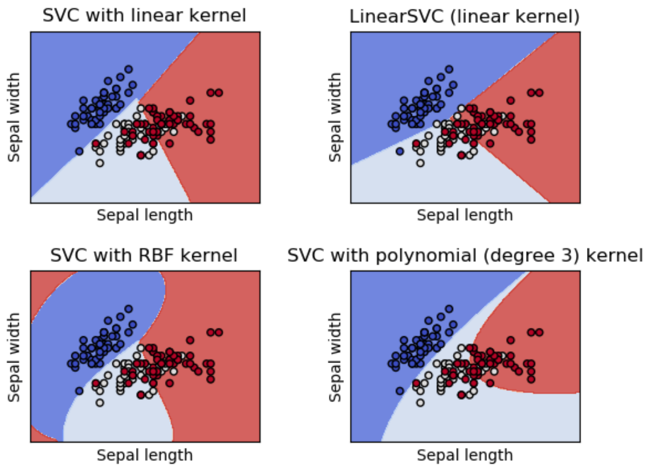
Its dual is

$$\begin{aligned} \min_{\alpha} \frac{1}{2} \alpha^T Q \alpha - e^T \alpha \\ \text{subject to } y^T \alpha = 0 \\ 0 \leq \alpha_i \leq C, i = 1, \ldots, n \end{aligned}$$

where e is the vector of all ones, $C > 0$ is the upper bound, Q is an n by n positive semidefinite matrix, $Q_{ij} \equiv y_i y_j K(x_i, x_j)$, where $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ is the kernel.

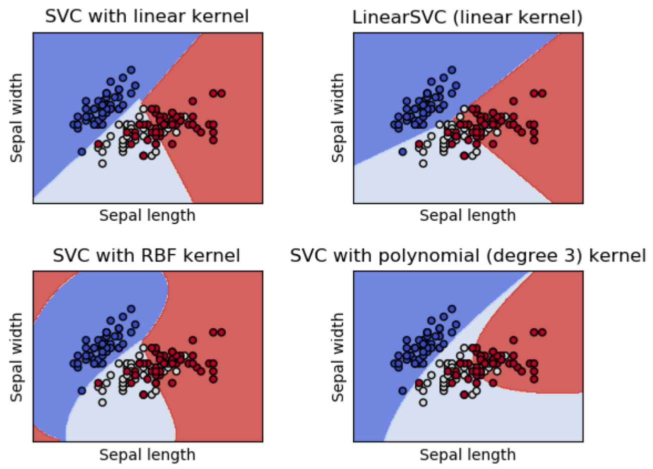
SVM Kernel Trick - Kernel functions





Comparison of four kernels for iris dataset

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets


def make_meshgrid(x, y, h=.02):
    """Create a mesh of points to plot in

    Parameters
    ----------
    x: data to base x-axis meshgrid on
    y: data to base y-axis meshgrid on
    h: stepsize for meshgrid, optional

    Returns
    -------
    xx, yy : ndarray
    """
    x_min, x_max = x.min() - 1, x.max() + 1
    y_min, y_max = y.min() - 1, y.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    return xx, yy


def plot_contours(ax, clf, xx, yy, **params):
    """Plot the decision boundaries for a classifier.

    Parameters
    ----------
    ax: matplotlib axes object
    clf: a classifier
    xx: meshgrid ndarray
    yy: meshgrid ndarray
```



SVM pros and cons

The advantages of support vector machines are

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

The disadvantages of support vector machines include

- If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial.
- SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation (see Scores and probabilities, below).

SVM hands on

We'll use the built in breast cancer dataset from Scikit Learn. We can get with the load function:

```python
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
print(cancer.keys()) # dataset is in a dictionary form
print(cancer['DESCR'])
```

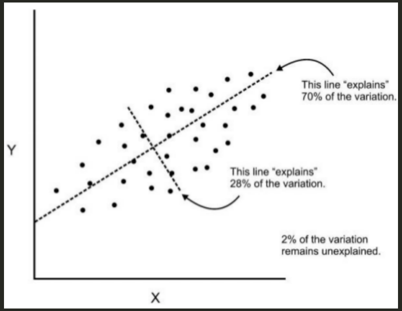
SVM + Gridsearch

Finding the right parameters (like what C or gamma values to use) is a tricky task! But luckily, we can be a little lazy and just try a bunch of combinations and see what works best! This idea of creating a 'grid' of parameters and just trying out all the possible combinations is called a Gridsearch, this method is common enough that Scikit-learn has this functionality built in with GridSearchCV! The CV stands for cross-validation which is the

GridSearchCV takes a dictionary that describes the parameters that should be tried and a model to train. The grid of parameters is defined as a dictionary, where the keys are the parameters and the values are the settings to be tested.

```
param_grid = {'C': [0.1,1, 10, 100, 1000], 'gamma': [1,0.1,0.01,0.001,0.0001], 'kernel': ['rbf']}
from sklearn.model_selection import GridSearchCV
grid = GridSearchCV(SVC(),param_grid,refit=True,verbose=3)
grid.fit(X_train,y_train)
grid.best_params_
grid.best_estimator_
grid_predictions = grid.predict(X_test)
```

Principal Component Analysis (PCA)


This line "explains" 70% of the variation.
This line "explains" 28% of the variation.
2% of the variation remains unexplained.
Y
X

Factor analysis - Principal component analysis

https://en.wikipedia.org/wiki/Factor_analysis

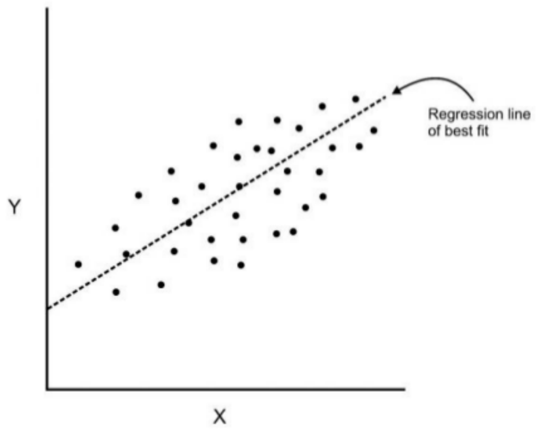
- It is possible that variations in six observed variables mainly reflect the variations in two unobserved (underlying) variables. Factor analysis searches for such joint variations in response to unobserved latent variables. The observed variables are modelled as linear combinations of the potential factors, plus "error" terms.

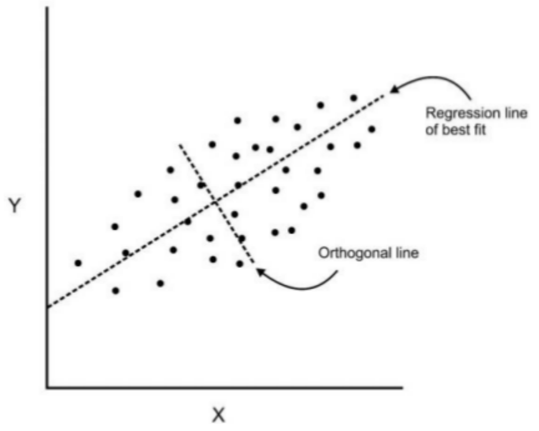
- PCA is an unsupervised statistical technique used to examine the interrelations among a set of DS variables in order to identify the underlying structure of those

- It is also known sometimes as a general factor analysis
-
    - It is a variable reduction technique

PCA Background

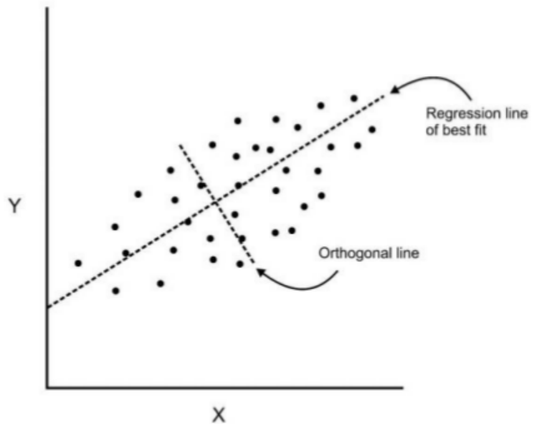
- Where regression determines a line of best fit to a data set, orthogonal factor analysis determines several lines of best fit to the data set.
- Orthogonal means “at right angles” - actually the lines are perpendicular to each other in n-dimensional space.
- n-Dimensional Space is the variable sample space. There are as many dimensions as there are variables, so in a data set with 4 variables the sample space is 4-dimensional.

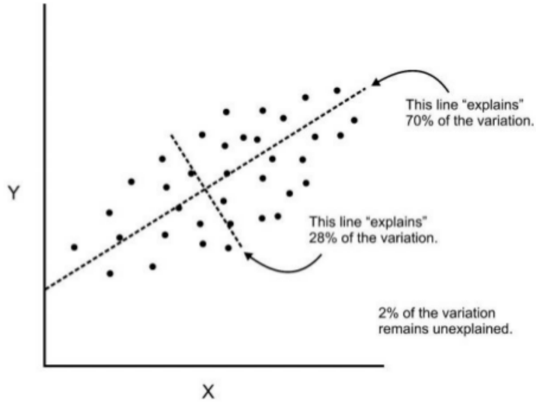




PCA - Components

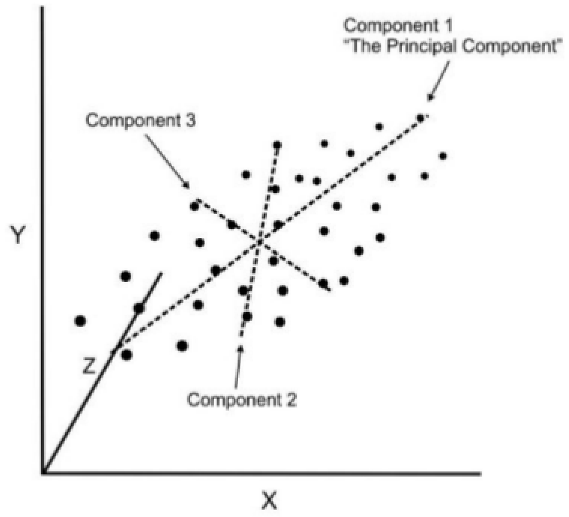
- Components are a linear transformation that chooses a variable system for the data set such that the greatest variance of the data set comes to lie on the first axis
- The second greatest variance on the second axis, and so on …
- This process allows us to reduce the number of variables used in an analysis.
- Note that components are uncorrelated, since in the sample space they are orthogonal to each other.




This line "explains" 70% of the variation.
This line "explains" 28% of the variation.
2% of the variation remains unexplained.
Y
X

PCA higher dimensions

- We can continue this analysis into higher dimensions.
- If we use this technique on a data set with a large number of variables, we can compress the amount of explained variation to just a few components.
- The most challenging part of PCA is interpreting the components.



PCA Hands on in Python

- For our work with Python, we’ll walk through an example of how to perform PCA with scikit learn.
- We usually want to standardize our data by some scale for PCA, so we’ll cover how to do this as well.

- This algorithm is used usually for analysis of data and not a fully deployable model, so there won’t be a 'portfolio' project for this topic.

- Before we do this though, we'll need to scale our data so that each feature has a single unit variance.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(df)
scaled_data = scaler.transform(df)

from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca.fit(scaled_data)
x_pca = pca.transform(scaled_data)

plt.figure(figsize=(8,6))
plt.scatter(x_pca[:,0],x_pca[:,1],c=cancer['target'],cmap='plasma')
plt.xlabel('First principal component')
plt.ylabel('Second Principal Component')
```

Thank you