1. Loading the NIPS papers

The NIPS conference (Neural Information Processing Systems) is one of the most prestigious yearly events in the machine learning community. At each NIPS conference, a large number of research papers are published. Over 50,000 PDF files were automatically downloaded and processed to obtain a dataset on various machine learning techniques. These NIPS papers are stored in `datasets/papers.csv`. The CSV file contains information on the different NIPS papers that were published from 1987 until 2017 (30 years!). These papers discuss a wide variety of topics in machine learning, from neural networks to optimization methods and many more.

First, we will explore the CSV file to determine what type of data we can use for the analysis and how it is structured. A research paper typically consists of a title, an abstract and the main text. Other data such as figures and tables were not extracted from the PDF files. Each paper discusses a novel technique or improvement. In this analysis, we will focus on analyzing these papers with natural language processing methods.

```
In [2]: # Importing modules
        import pandas as pd

        # Read datasets/papers.csv into papers
        papers = pd.read_csv('datasets/papers.csv')

        # Print out the first rows of papers
        papers.head()
```

Out[2]:

	id	year	title	event_type	pdf_name	abstract	paper_text
0	1	1987	Self-Organization of Associative Database and ...	NaN	1-self-organization-of-associative-database-an...	Abstract Missing	767\n\nSELF-ORGANIZATION OF ASSOCIATIVE DATABA...
1	10	1987	A Mean Field Theory of Layer IV of Visual Cort...	NaN	10-a-mean-field-theory-of-layer-iv-of-visual-c...	Abstract Missing	683\n\nA MEAN FIELD THEORY OF LAYER IV OF VISU...
2	100	1988	Storing Covariance by the Associative Long-Ter...	NaN	100-storing-covariance-by-the-associative-long...	Abstract Missing	394\n\nSTORING COVARIANCE BY THE ASSOCIATIVE\n...
3	1000	1994	Bayesian Query Construction for Neural Network...	NaN	1000-bayesian-query-construction-for-neural-ne...	Abstract Missing	Bayesian Query Construction for Neural\nNetwor...
4	1001	1994	Neural Network Ensembles, Cross Validation, an...	NaN	1001-neural-network-ensembles-cross-validation...	Abstract Missing	Neural Network Ensembles, Cross\nValidation, a...

```
In [3]: %%nose

        import pandas as pd

        def test_papers_exists():
            assert "papers" in globals(), \
                "The variable papers should be defined."

        def test_papers_correctly_loaded():
            correct_papers = pd.read_csv("datasets/papers.csv")
            assert correct_papers.equals(papers), "The variable papers should contain
         the data in papers.csv"

Out[3]: 2/2 tests passed
```

2. Preparing the data for analysis

For the analysis of the papers, we are only interested in the text data associated with the paper as well as the year the paper was published in.

We will analyze this text data using natural language processing. Since the file contains some metadata such as id's and filenames, it is necessary to remove all the columns that do not contain useful text information.

In [4]:
```
# Remove the columns
papers.drop(['id', 'event_type', 'pdf_name'], axis=1, inplace=True, errors='ig
nore')

# Print out the first rows of papers
papers.head()
```

Out[4]:

	year	title	abstract	paper_text
0	1987	Self-Organization of Associative Database and ...	Abstract Missing	767\n\nSELF-ORGANIZATION OF ASSOCIATIVE DATABA...
1	1987	A Mean Field Theory of Layer IV of Visual Cort...	Abstract Missing	683\n\nA MEAN FIELD THEORY OF LAYER IV OF VISU...
2	1988	Storing Covariance by the Associative Long-Ter...	Abstract Missing	394\n\nSTORING COVARIANCE BY THE ASSOCIATIVE\n...
3	1994	Bayesian Query Construction for Neural Network...	Abstract Missing	Bayesian Query Construction for Neural\nNetwor...
4	1994	Neural Network Ensembles, Cross Validation, an...	Abstract Missing	Neural Network Ensembles, Cross\nValidation, a...

In [5]:
```
%%nose

import pandas as pd

def test_papers_exists():
    assert "papers" in globals(), \
        "The variable `papers` should be defined."

def test_papers_columns():
    assert papers.columns.size==4, \
        "The variable `papers` does not contain the right amount of columns."
```

Out[5]:
```
2/2 tests passed
```

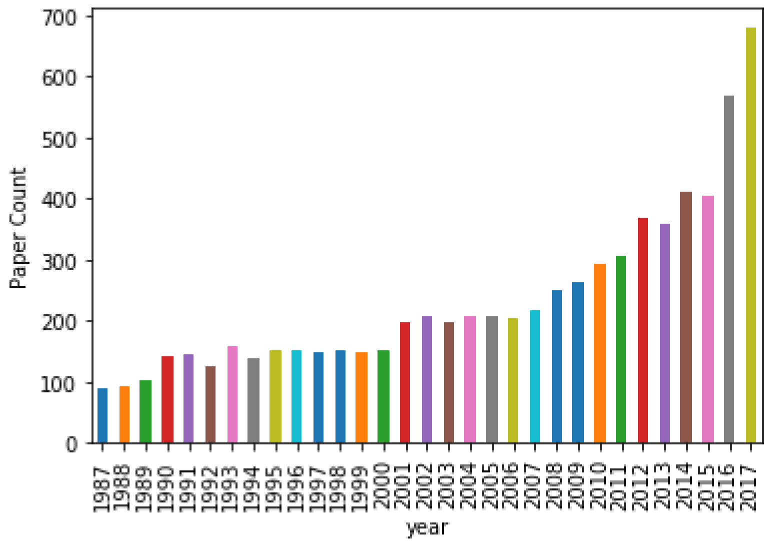
3. Plotting how machine learning has evolved over time

In order to understand how the machine learning field has recently exploded in popularity, we will begin by visualizing the number of publications per year.

By looking at the number of published papers per year, we can understand the extent of the machine learning 'revolution'! Typically, this significant increase in popularity is attributed to the large amounts of compute power, data and improvements in algorithms.

In [6]:

```
# Group the papers by year
groups = papers.groupby('year')

# Determine the size of each group
counts = groups.size()

# Visualise the counts as a bar plot
import matplotlib.pyplot as plt
%matplotlib inline
counts.plot(kind='bar')
plt.ylabel('Paper Count')
plt.show()
```



```
In [7]: %%nose

        # one or more tests of the students code.
        # The @solution should pass the tests.
        # The purpose of the tests is to try to catch common errors and to
        # give the student a hint on how to resolve these errors.

        def test_vars_exists():
            assert "groups" in globals(), \
                "The variable groups should be defined."
            assert "counts" in globals(), \
                "The variable counts should be defined."

        def test_vars_columns():
            correct_groups = papers.groupby('year')
            correct_counts = correct_groups.size()
            assert correct_counts.equals(counts), "The variable counts is not correctl
        y defined."

Out[7]: 2/2 tests passed
```

4. Preprocessing the text data

Let's now analyze the titles of the different papers to identify machine learning trends. First, we will perform some simple preprocessing on the titles in order to make them more amenable for analysis. We will use a regular expression to remove any punctuation in the title. Then we will perform lowercasing. We'll then print the titles of the first rows before and after applying the modification.

```
In [8]: # Load the regular expression library
        import re

        # Print the titles of the first rows
        print(papers['title'].head())

        # Remove punctuation
        papers['title_processed'] = papers['title'].map(lambda x: re.sub('[,\.!?]', ''
        , x))

        # Convert the titles to lowercase
        papers['title_processed'] = papers['title_processed'].str.lower()

        # Print the processed titles of the first rows
        papers['title_processed'].head()

        0    Self-Organization of Associative Database and ...
        1    A Mean Field Theory of Layer IV of Visual Cort...
        2    Storing Covariance by the Associative Long-Ter...
        3    Bayesian Query Construction for Neural Network...
        4    Neural Network Ensembles, Cross Validation, an...
        Name: title, dtype: object

Out[8]: 0    self-organization of associative database and ...
        1    a mean field theory of layer iv of visual cort...
        2    storing covariance by the associative long-ter...
        3    bayesian query construction for neural network...
        4    neural network ensembles cross validation and ...
        Name: title_processed, dtype: object
```

```
In [9]: %%nose

        # one or more tests of the students code.
        # The @solution should pass the tests.
        # The purpose of the tests is to try to catch common errors and to
        # give the student a hint on how to resolve these errors.

        def test_processed_title_column():
            tmp = papers['title'].map(lambda x: re.sub('[,\.!?]', '', x))
            # Convert the titles to lowercase
            tmp = tmp.map(lambda x: x.lower())
            tmp.name = 'title_processed'
            assert tmp.equals(papers['title_processed']), "The column of processed tit
        les is not correctly defined."

Out[9]: 1/1 tests passed
```

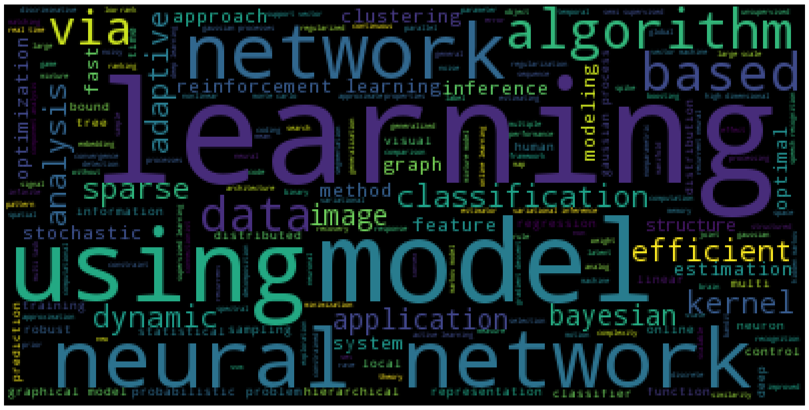
5. A word cloud to visualize the preprocessed text data

In order to verify whether the preprocessing happened correctly, we can make a word cloud of the titles of the research papers. This will give us a visual representation of the most common words. Visualisation is key to understanding whether we are still on the right track! In addition, it allows us to verify whether we need additional preprocessing before further analyzing the text data.

Python has a massive number of open libraries! Instead of trying to develop a method to create word clouds ourselves, we'll use Andreas Mueller's wordcloud library (http://amueller.github.io/word_cloud/).

In [10]:
```
# Import the wordcloud library
from wordcloud import WordCloud

# Join the different processed titles together.
long_string = ' '.join(papers['title_processed'].values)

# Create a WordCloud object
wordcloud = WordCloud()

# Generate a word cloud
wordcloud.generate(long_string)

# Visualize the word cloud
wordcloud.to_image()
```

Out[10]:



In [11]:
```
%%nose

# one or more tests of the students code.
# The @solution should pass the tests.
# The purpose of the tests is to try to catch common errors and to
# give the student a hint on how to resolve these errors.

def test_example():
    assert long_string == ' '.join(papers['title_processed']), \
    'The titles were not parsed correctly to a single long string.'
```

Out[11]:
```
1/1 tests passed
```

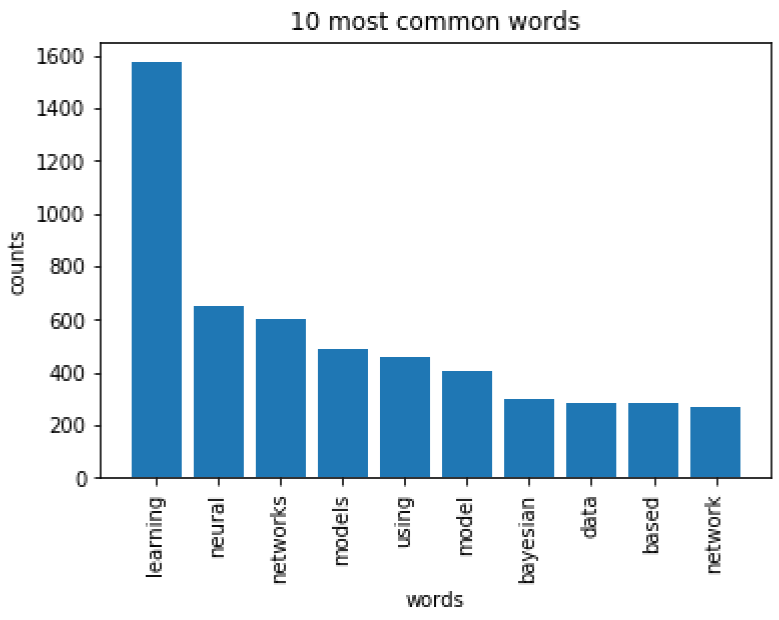
6. Prepare the text for LDA analysis

The main text analysis method that we will use is latent Dirichlet allocation (LDA). LDA is able to perform topic detection on large document sets, determining what the main 'topics' are in a large unlabeled set of texts. A 'topic' is a collection of words that tend to co-occur often. The hypothesis is that LDA might be able to clarify what the different topics in the research titles are. These topics can then be used as a starting point for further analysis.

LDA does not work directly on text data. First, it is necessary to convert the documents into a simple vector representation. This representation will then be used by LDA to determine the topics. Each entry of a 'document vector' will correspond with the number of times a word occurred in the document. In conclusion, we will convert a list of titles into a list of vectors, all with length equal to the vocabulary. For example, *'Analyzing machine learning trends with neural networks.'* would be transformed into `[1, 0, 1, ..., 1, 0]`.

We'll then plot the 10 most common words based on the outcome of this operation (the list of document vectors). As a check, these words should also occur in the word cloud.

In [12]:
```python
# Load the library with the CountVectorizer method
from sklearn.feature_extraction.text import CountVectorizer
import numpy as np

# Helper function
def plot_10_most_common_words(count_data, count_vectorizer):
    import matplotlib.pyplot as plt
    words = count_vectorizer.get_feature_names()
    total_counts = np.zeros(len(words))
    for t in count_data:
        total_counts+=t.toarray()[0]

    count_dict = (zip(words, total_counts))
    count_dict = sorted(count_dict, key=lambda x:x[1], reverse=True)[0:10]
    words = [w[0] for w in count_dict]
    counts = [w[1] for w in count_dict]
    x_pos = np.arange(len(words))

    plt.bar(x_pos, counts,align='center')
    plt.xticks(x_pos, words, rotation=90)
    plt.xlabel('words')
    plt.ylabel('counts')
    plt.title('10 most common words')
    plt.show()

# Initialise the count vectorizer with the English stop words
count_vectorizer = CountVectorizer(stop_words='english')

# Fit and transform the processed titles
count_data = count_vectorizer.fit_transform(papers['title_processed'])

# Visualise the 10 most common words
plot_10_most_common_words(count_data, count_vectorizer)
```



In [13]:
```
%%nose

# one or more tests of the students code.
# The @solution should pass the tests.
# The purpose of the tests is to try to catch common errors and to
# give the student a hint on how to resolve these errors.

def test_count_data():
    assert "count_data" in globals(), \
        "The variable count_data should be defined."
```

Out[13]:
```
1/1 tests passed
```

7. Analysing trends with LDA

Finally, the research titles will be analyzed using LDA. Note that in order to process a new set of documents (e.g. news articles), a similar set of steps will be required to preprocess the data. The flow that was constructed here can thus easily be exported for a new text dataset.

The only parameter we will tweak is the number of topics in the LDA algorithm. Typically, one would calculate the 'perplexity' metric to determine which number of topics is best and iterate over different amounts of topics until the lowest 'perplexity' is found. For now, let's play around with a different number of topics. From there, we can distinguish what each topic is about ('neural networks', 'reinforcement learning', 'kernel methods', 'gaussian processes', etc.).

In [17]:
```
import warnings
warnings.simplefilter("ignore", DeprecationWarning)

# Load the LDA model from sk-learn
from sklearn.decomposition import LatentDirichletAllocation as LDA

# Helper function
def print_topics(model, count_vectorizer, n_top_words):
    words = count_vectorizer.get_feature_names()
    for topic_idx, topic in enumerate(model.components_):
        print("\nTopic #%d:" % topic_idx)
        print(" ".join([words[i]
                        for i in topic.argsort()[:-n_top_words - 1:-1]]))

# Tweak the two parameters below (use int values below 15)
number_topics = 10
number_words = 4

# Create and fit the LDA model
lda = LDA(n_components=number_topics, random_state=720)
lda.fit(count_data)

# Print the topics found by the LDA model
print("Topics found via LDA:")
print_topics(lda, count_vectorizer, number_words)
```

```
Topics found via LDA:

Topic #0:
data image risk analysis

Topic #1:
deep sampling recognition learning

Topic #2:
models bayesian multi inference

Topic #3:
non probabilistic prediction convex

Topic #4:
learning stochastic analysis optimal

Topic #5:
neural networks learning large

Topic #6:
optimization structure neurons memory

Topic #7:
classification gradient learning detection

Topic #8:
learning sparse time efficient

Topic #9:
clustering markov model models
```

In [ ]:

```
%%nose

# No standard testing procedure exists for printing at the moment

def test_nothing():
    assert True, "Nothing to test"
```

8. The future of machine learning

Machine learning has become increasingly popular over the past years. The number of NIPS conference papers has risen exponentially, and people are continuously looking for ways on how they can incorporate machine learning into their products and services.

Although this analysis focused on analyzing machine learning trends in research, a lot of these techniques are rapidly being adopted in industry. Following the latest machine learning trends is a critical skill for a data scientist, and it is recommended to continuously keep learning by going through blogs, tutorials, and courses.

In [18]:

```
# The historical data indicates that:
more_papers_published_in_2018 = True
```

In [ ]:
```
%%nose

# one or more tests of the students code.
# The @solution should pass the tests.
# The purpose of the tests is to try to catch common errors and to
# give the student a hint on how to resolve these errors.

def test():
        assert more_papers_published_in_2018 == True, \
    'The number published papers has been rising the past 10 years!'
```